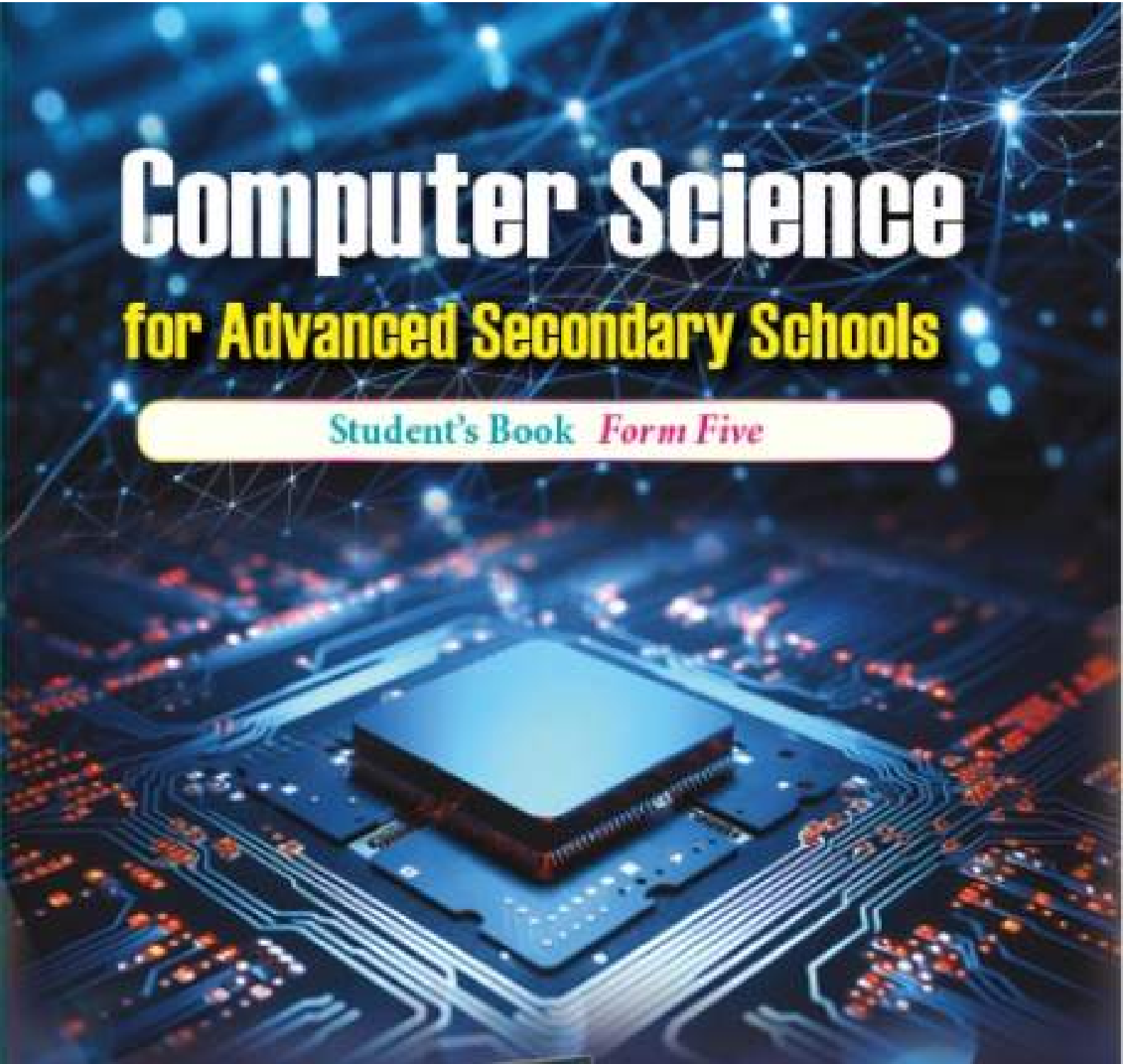
Computer Science

for Advanced Secondary Schools

Student's Book *Form Five*





Tanzania Institute of Education

Computer Science

for Advanced Secondary Schools

Student's Book
Form Five



Tanzania Institute of Education

FOR ONLINE READING ONLY

©Tanzania Institute of Education 2024

Published 2024

ISBN: 978-9912-753-26-6

Mikocheni Area
132 Ali Hassan Mwinyi Road
P. O. Box 35094
14112 Dar es Salaam

Mobile numbers: +255 735 041 170 / +255 735 041 168
E-mail: director.general@tie.go.tz
Website: www.tie.go.tz

FOR ONLINE READING ONLY

All rights reserved. No part of this textbook may be reproduced, stored in any retrieval system or transmitted in any form or by any means whether electronic, mechanical, photocopying, recording, or otherwise without prior written permission of the Tanzania Institute of Education.

Table of Contents

Acronyms and abbreviations vi
Acknowledgements vii
Preface viii

Chapter One: Computer architecture 1
The Concept of computer architecture 1
Boolean algebra and logic gates 10
Truth tables 15
Logic gates 21
Logic circuits 30
Computer memory 35
Instruction set architecture and design 51
Input and Output systems 70
Revision Exercise 1 80
Project 80

Chapter Two: Performance and optimisation of computer processor 82
Pipelining 82
Instruction Level Parallelism (ILP) 95
Thread and data Level parallelism 109
Data Level Parallelism 116
Revision exercise 2 122
Project: Analyzing Website Traffic Data 123

Chapter Three: Advanced Computer Networks 124
Concept of network communication models 124
Open Systems Interconnection Model 125

FOR ONLINE READING ONLY

TCP/IP model 128
Network models 131
Transmission media 178
Straight -Through cable construction 189
Crossover cable construction 190
Designing Local Area Network 191
Network security principles and protocols in computer networks 196
Public Switched Telephone Networks and Mobile telephone system 203
Revision Exercise 3 209
Project: Designing a school Network 210

Chapter Four: Object oriented programming with C++ 211
Concept of Object Oriented Programming (OOP) 211
Introduction to C++ programming language 224
Variables and constants 241
Operators, expressions and assignments 249
Functions 274
Arrays and strings 283
Implementation of key OOP features in C++ 292
Revision Exercise 4 304
Project: Designing and implementing a simple library management system 307

Chapter Five: Object oriented programming with Java 308
Concept of Object Oriented Programming with Java 308
Key principles of OOP in Java 312
Basic Java syntax and constructs 320
Variables, operators and constants 323
Functions, objects and classes in Java OOP 343
Constructors and destructors in Java 348
Implementation of key OOP features in Java 354
Revision Exercise 5 383
Project: Exploring Garbage Collection in Java 384

FOR ONLINE READING ONLY

Chapter Six: Object oriented programming with Python **385**
Concept of Object Oriented Programming with Python 385
Introduction to python OOP 386
Python fundamentals 389
Python Functions and modules 396
OOP Concepts in Python 399
Implementation of key OOP features in Python 402
Revision Exercise 6 407
Project: Developing school Examination Grading System 408

Chapter Seven: Operating Systems **410**
Concept of Operating Systems 410
Impact of operating system design on application system design and performance 416
Operating system design and performance 418
Revision exercise 7 431
Project: Exploring and Comparing Operating Systems 432

Chapter Eight: Databases and Database Management Systems **433**
Basic Concepts of Databases and Database Management Systems 433
Database models and Design 437
Database Management Systems (DBMS) 466
Emerging Database Models, Technologies, and Applications 484
Designing Database Using SQL and PHP 492
PHP Fundamentals 500
NoSQL Databases 517
Revision exercise 8 522
Project: Multi-Functional Library Management System 523
(End of Advanced Level Secondary Education Project): 524

Glossary **525**

Bibliography **529**

Appendices **531**

Indices **533**

FOR ONLINE READING ONLY

Acronyms and abbreviations

OOP	Object-Oriented Programming
JDK	Java Development Kit
JVM	Java Virtual Machine
IDE	Integrated Development Environment
OSI	Open Systems Interconnection
TCP/IP	Transmission Control Protocol/Internet Protocol
HTTP	Hypertext Transfer Protocol
FTP	File Transfer Protocol
SMTP	Simple Mail Transfer Protocol
DNS	Domain Name System
UDP	User Datagram Protocol

FOR ONLINE READING ONLY

Acknowledgements

The Tanzania Institute of Education (TIE) would like to acknowledge the contributions of all the organisations and individuals who participated in designing and developing this textbook. In particular, TIE wishes to thank the University of Dar es Salaam (CoICT-UDSM), Sokoine University of Agriculture (SUA), Mzumbe University (MU), The Open University of Tanzania (OUT), Arusha Technical College (ATC), Institute of Finance Management (IFM), School Quality Assurance (SQA) department, Teachers' Colleges and Secondary Schools. Besides, the following individuals are acknowledged:

Writers:	Watson Gibson Mtewele (TIE), Dr Leonard Peter Binamungu (CoICT, UDSM), Dr Hellen Maziku (CoICT, UDSM), and Onesmo Sedekia (TIE)
Editors:	Dr. Edephonce N. Nfuka (OUT), Dr. Dossa Massa (IFM), Dr.-Eng. Monce Daudi Tegeje (MU), Dr. Adam B. Mtaho (ATC), Dr. Michael Mahenge (SUA), Dr. Msuya (UDSM), Mwl. Hamis Seif (Picha ya Ndege SS).
Designer:	Amani J. Kweka
Illustrators:	Hance Elias Wawar
Coordinator:	Watson Gibson Mtewele

TIE also appreciates secondary school teachers and students who participated in the trial phase of the manuscript of this book. Likewise, TIE would like to thank the Ministry of Education, Science and Technology for facilitating the writing and printing of this textbook.

Dr Aneth A. Komba
Director General
Tanzania Institute of Education

FOR ONLINE READING ONLY

Preface

This textbook, *Computer Science for Advanced Secondary Schools Student's Book*, has been written specifically for Form Five students in the United Republic of Tanzania.

The book is prepared in accordance with the 2023 Computer Science Syllabus for Advanced Secondary Education, Form V-VI, issued by the Ministry of Education, Science and Technology.

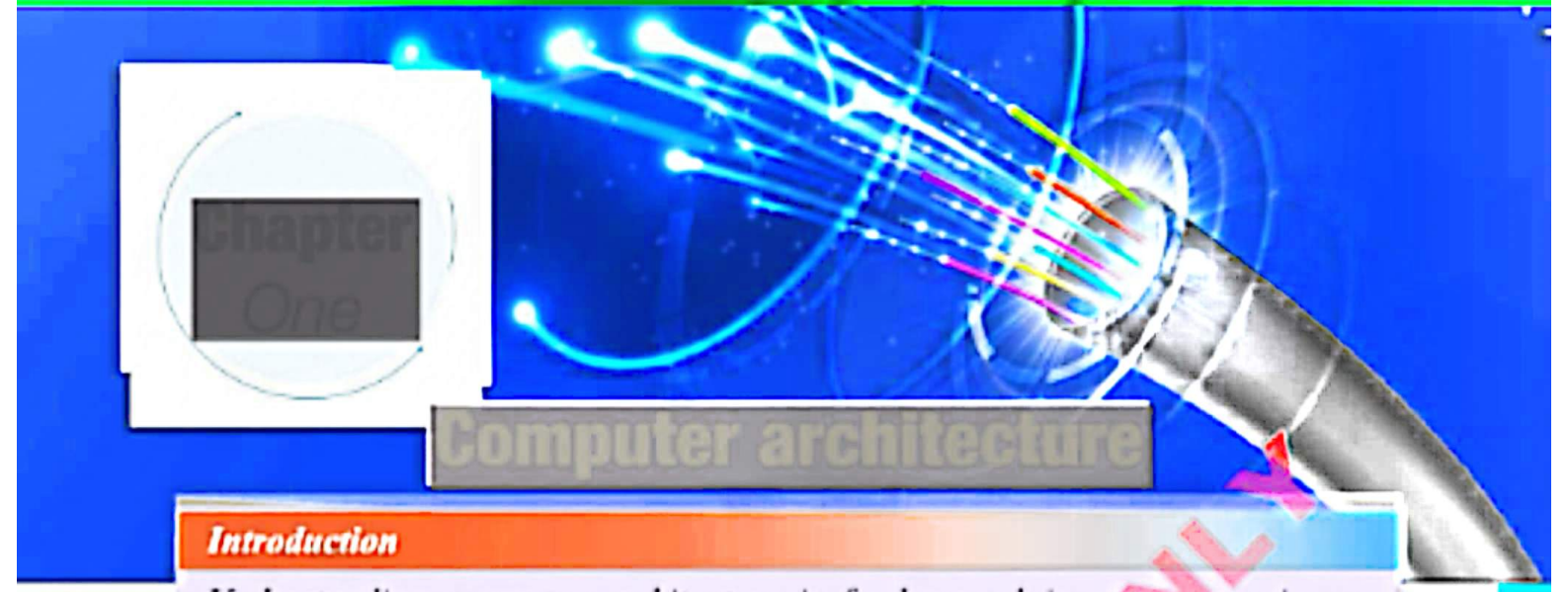
The book consists of eight chapters, which are: Computer architecture; Performance and optimization of computer processor; Advanced computer networks; Object-Oriented Programming with C++; Object-Oriented Programming with Java; Object-Oriented Programming with Python; Operating Systems; and Databases and Database Management Systems. In addition to textual descriptions, each chapter contains illustrations, activities, and exercises. You are encouraged to do all activities and exercises in this book, and other assignments provided by your teacher. You are also required to prepare a portifolio for keeping records of activities performed in different lessons. This will enhance your understanding and development of the intended competencies for this level.

In addition to your studies at this level, you will be required to design and carry out a project in Computer Science, and then submit its final report for assessment. For more details refer to the project guide.

Additional learning resources are available in the TIE e-learning at *https://ol.tie.go.tz* or *ol.tie.go.tz*

Tanzania Institute of Education

FOR ONLINE READING ONLY



Chapter One

Computer architecture

Introduction

Understanding computer architecture is fundamental in computer science because it lays the foundation for how computers process data, execute instructions, and perform tasks efficiently. Computer architecture entails how various computer components interact to create a computer system. Similar to how architects plan, construct, and maintain buildings, computer architects design, construct, and manage the development of computer systems. In this chapter, you will learn the concept of computer architecture, classification of computer architecture, and Boolean algebra and logic gates. You will also learn computer memory, Instruction Set Architecture (ISA), and Input or Output (I/O) systems. The competencies developed will enable you to design and manage computer systems.

Think

How computer architecture shapes our daily technologies

The Concept of computer architecture

Scenario 1.1: *Computer laboratory performance*

The school's new computer laboratory is experiencing performance issues, with slow programs to access educational materials, conducting research and doing assignments. Upgrading every computer may not be possible, so understanding computer architecture becomes crucial. Assume the computers are houses with shared space where the current computers have a single "room" for both instructions and data, causing a traffic jam that slows down processing.

Question:

Based on the scenario, describe the ways you may undertake to improve the computer performance.

FOR ONLINE READING ONLY

Step 1: Understand the components of the simulator (in this case, use open-source tool called Johnny simulator as shown in Figure 1.1. Terminologies used for simulator operations are indicated in Table 1.1.

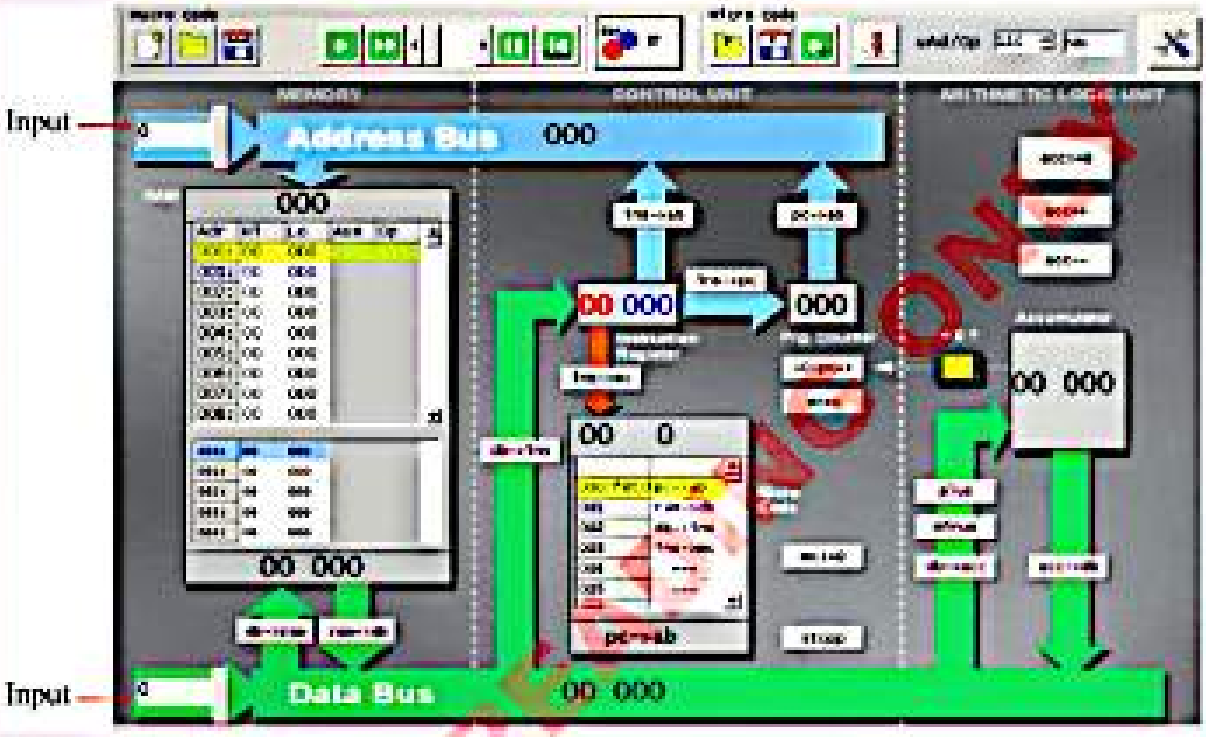


Figure 1.1: *Components of Johnny simulator*

Table 1.1: *Description of the terms and operations used in the simulators*

S/N	Command/term	Description
Memory (RAM)	ram->db	Data from the addressed location in the RAM can be transferred onto the data bus
	db->ram	Data from the data bus can be transferred onto the addressed location in the RAM.
The Control Unit	db->ins	The content of the data bus is transferred to the instruction register.
	ins- >ab	The content of the instruction register is transferred to the data bus.
	ins->pc	The content of the instruction register is transferred to the program counter
	pc->ab	The content of the program counter is transferred to the address bus
	ins->mc	The content of the instruction register is transferred to the micro instruction counter
	stopp	The simulator shows a message that the program execution is finished.

FOR ONLINE READING

The Arithmetic Logic Unit	db->acc	Transports a data word from the data bus into the accumulator
	acc->db	Transports a data word from the accumulator into the data bus
	plus	Data from the data bus can be added
	minus	Data from the data base can substracted

Step 2: Simulate the movement of data from the input devices to other parts of the computer such as Random Access Memory (RAM) as shown in Figure 1.2 using the command given in Table 1.1.

(a) Enter data in the data input box. For this case, enter 100 as shown in the data input box and press ENTER in the keyboard.

(b) Click the arrows with the command and observe how data transfers from one location to another.

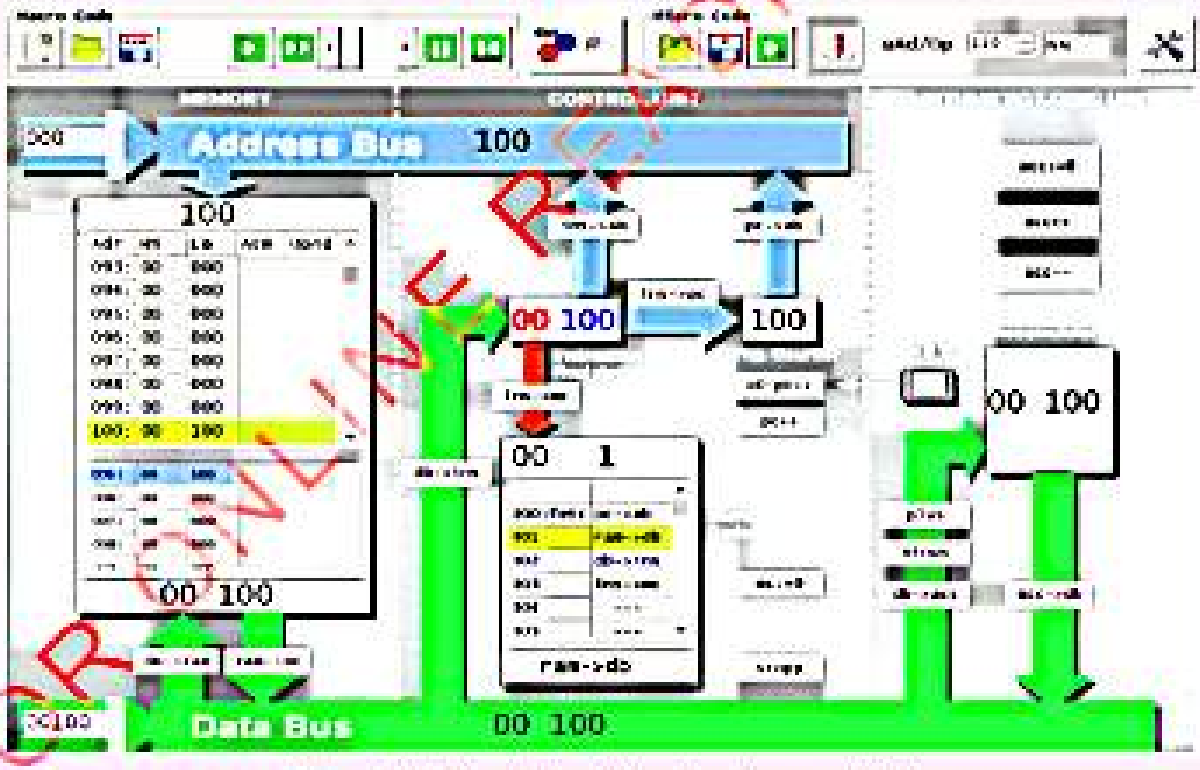


Figure 1.2: *Simulation of data from input to RAM*

Step 3: Repeat the procedures in step 2 with other input data and explore how data moves form one part to another.

Questions

(a) Describe the relationship between the three parts of the computer processor.

(b) Does a bus system used for both data and instructions, allow the CPU to fetch and execute instructions sequentially? Explain.

FOR REVIEW PURPOSE ONLY

Activity 1.2:

Simulation of Von Neumann architecture (Data allocation in memory)

Aim: Understanding of how data is allocated and managed in memory within a Von Neumann architecture.

Materials: Computer, Johnny simulator software, Internet if you don't have a simulator

Instructions: Using Johnny simulator, create a program that adds the values of locations 10 and 11 and stores the result in location 12 to simulate memory allocation. Then answer the questions that follows.

Step 1: Open the simulator and click on one RAM location to launch the window where the numerical value can be changed or a macro instruction can be chosen from a pull-down menu as shown in Figure 1.3.

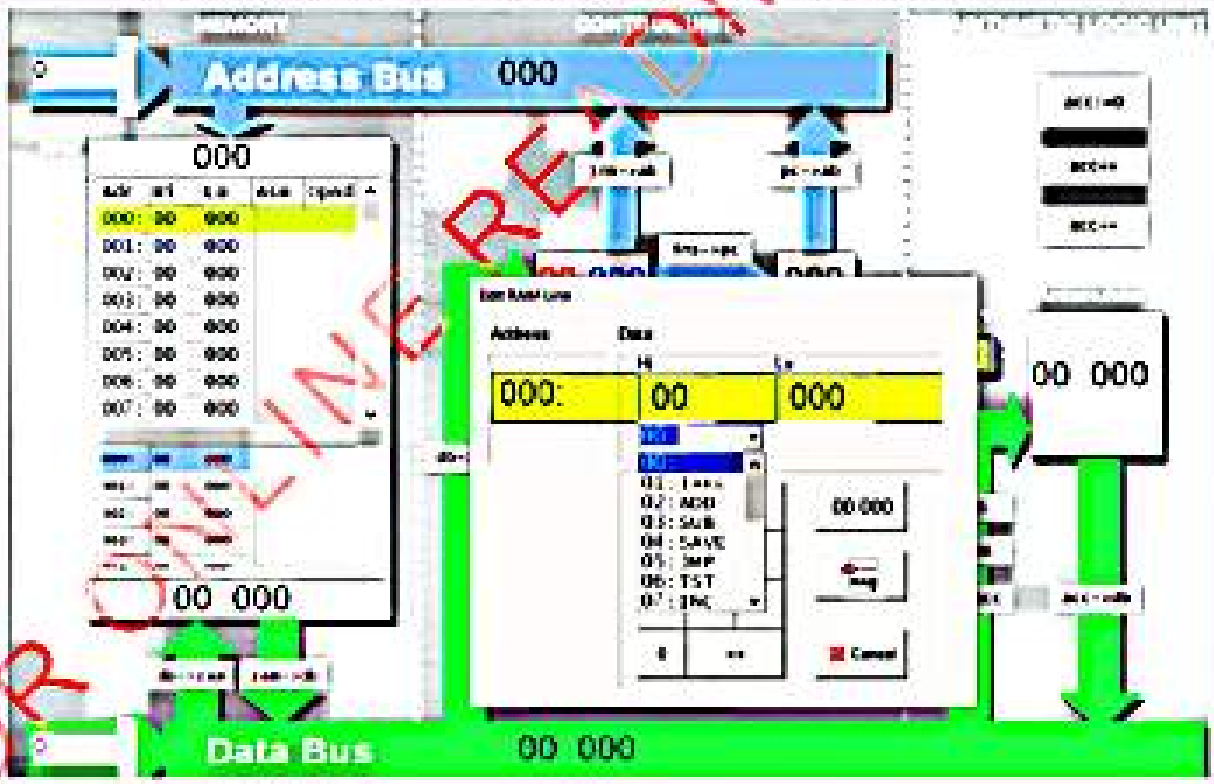


Figure 1.3: *Simulation of data from input to RAM*

Table 1.2: *Description of the standard (macro) instruction set used in the simulator*

Instruction set	Description
TAKE	The value of the location (given by the absolute address) is transported to the accumulator.
SAVE	The value of the accumulator is transported to the location given by the absolute address.

FOR ONLINE READING ONLY

ADD	The value of a location (given by the absolute address) is added to the value in the accumulator.
SUB	The value of a location (given by the absolute address) is subtracted from the value in the accumulator
INC	The value of the location (given by the absolute address) is incremented.
DEC	The value of the location (given by the absolute address) is decremented.
NULL	The value of the location (given by the absolute address) is set to zero.
HLT	The simulator shows a message that the program is finished.

Step 2: Select the instruction from the drop-down menu and enter the address using a real or virtual keyboard.

(a) In the edit RAM line interface, select one instruction i.e. TAKE or ADD or SAVE, etc.

(b) Edit the location (Lo) i.e. for TAKE instruction (010), ADD instruction (011), and for SAVE instruction (012), etc.

(c) For each instruction once the location is edited, click RAM button to write the contents in the RAM.

(d) Finally the contents written in the RAM appear as shown in Figure 1.4 below.

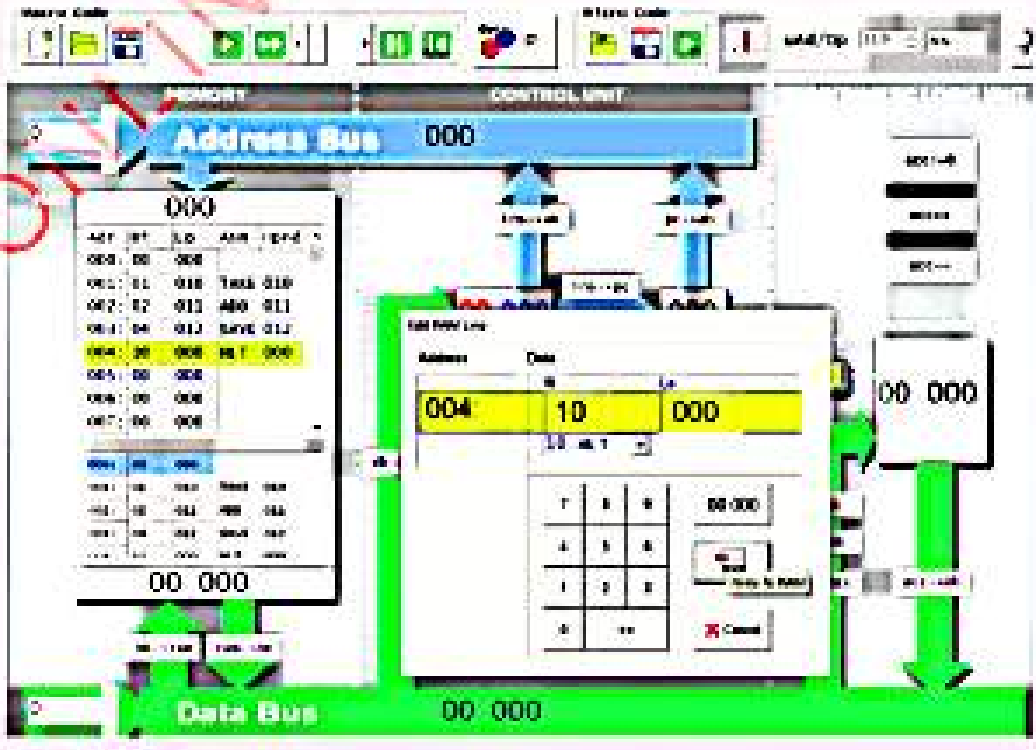


Figure 1.4: *Appearance of allocated data in RAM*

FOR ONLINE READING ONLY

Step 3: Repeat the procedures in step 1 to 3 and document every step.

Questions

Explain how memory allocation of data takes place by interaction between I/O system, RAM and CPU.

Carrying out the steps outlined in Activity 1.2 to simulate this computer architecture involves reading and interpreting the provided instructions, retrieving the necessary data and translating the instructions, executing the specified operations on the data, and storing the results. The visualised process are supported using Figure 1.5, which illustrates a Von Neumann architecture, a common model for the design of modern digital computers.

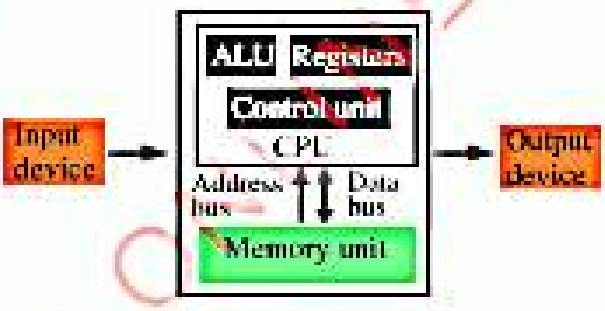


Figure 1.5: *A Von Neumann architecture*

Activity 1.3:

Exploring the Von Neumann architecture

1. Using the Internet or library search, explore:
   (a) The classification of computer architecture;
   (b) The features and applications of Von Neumann architecture and its roles in modern computing systems;
   (c) The advantages and disadvantages of Von Neumann architecture;
2. Document your findings in a portfolio.

Harvard architecture

Harvard architecture is commonly used in embedded systems, digital signal processors (DSPs), and real-time applications that require high speed and efficiency. This design allows for simultaneous access to instruction and data memories, improving performance by enabling parallel fetching and execution of instructions.

Activity 1.4:

Exploring the Harvard architecture

Use the Library or Internet search to perform the following:

1. Explore the features and applications of Harvard architecture and its roles in modern computing systems;
2. Explore the advantages and disadvantages of Harvard architecture;
3. Use any computer architecture simulation tool to simulate a Harvard architecture; and
4. Document your findings in a portfolio

FOR OFFICE USE ONLY

The visualised process for Activity 1.4 can be supported using Figure 1.6, which illustrates a Harvard architecture, a common model for the design of modern digital computers.

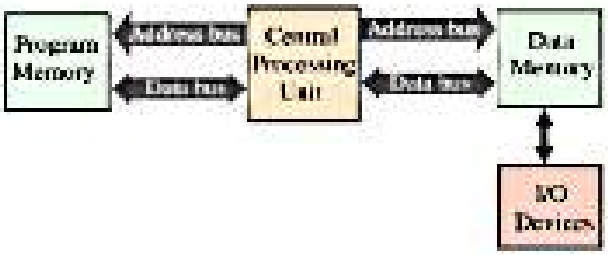


Figure 1.6: *Harvard architecture*

Modified Harvard architecture

The Modified Harvard architecture is a variation of the Harvard architecture that incorporates flexible memory usage. Similar to the Harvard architecture, it retains separate instruction and data memories but allows for unified memory space when necessary, providing flexibility in memory allocation and usage. This architecture strikes a balance between the advantages of separate memory spaces and the flexibility of unified memory, making it suitable for systems with dynamic memory requirements.

Activity 1.5:

Exploring the Modified Harvard architecture

Use the Library or Internet search to perform the following:

1. Explore the features and applications of Modified Harvard architecture and its roles in modern computing systems;
2. Explore the advantages and disadvantages of Modified Harvard architecture;
3. Use any computer architecture simulation tool to simulate a Modified Harvard architecture; and
4. Write down your findings and document them in a portfolio.

The visualised process for Activity 1.5 can be supported using Figure 1.7, which illustrates a Modified Harvard architecture, a common model for the design of modern digital computers.

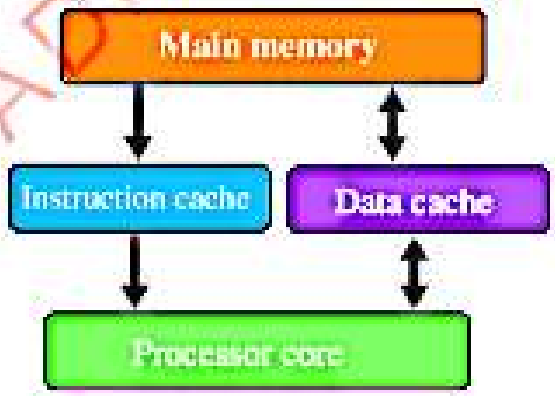


Figure 1.7: *Modified Harvard architecture*

Non -Von Neumann architecture

Non Von Neumann architecture refers to alternative computer architectures that depart from the traditional Von Neumann design. Unlike Von Neumann architecture, which features a single shared memory space for instructions and data, non Von Neumann architectures utilise separate memories for program instructions and data, often employing parallel processing techniques. Examples include distributed computing systems, neural networks, and parallel processing systems, which offer high performance and scalability for specialised applications.

FOR ONLINE READING ONLY

Activity 1.6:

Exploring the Non-Von Neumann architecture

Use the Library or Internet search perform the following:

1. Explore the features and applications of Non-Von Neumann architecture and its roles in modern computing systems;
2. Explore the advantages and disadvantages of Non-Von Neumann architecture;
3. Use any computer architecture simulation tool to simulate a Non-Von Neumann architecture; and
4. Write down your findings and document them in a portfolio.

The explored and visualised process for Activity 1.6 can be supported using Figure 1.8, which illustrates a Non-Von Neumann architecture, a common model for the design of modern digital computers.



Figure 1.8: *Non-Von Neumann architecture*

Flynn's taxonomy architecture

Flynn's taxonomy, proposed by Michael J. Flynn, categorises computer architectures based on the number of instruction streams and data streams processed simultaneously. It defines four categories: Single Instruction, Single Data (SISD), Single Instruction, Multiple Data (SIMD), Multiple Instruction, Single Data (MISD), and Multiple Instruction, Multiple Data (MIMD). Flynn's taxonomy provides a framework for classifying parallel processing architectures, facilitating understanding and comparing different systems based on their processing capabilities and architectural characteristics.

Activity 1.7:

Exploring the Flynn's taxonomy architecture

Use the Library or Internet search to perform the following:

1. Explore features and applications of Flynn's taxonomy architecture and its roles in modern computing systems;
2. Explore the advantages and disadvantages of Flynn's taxonomy architecture;
3. Use any computer architecture simulation tool, simulate a Flynn's taxonomy architecture;
4. Explore the four architecture classes resulting from Flynn's taxonomy architecture, and connect with the simulation in number 3 above; and
5. Write down your findings and document them in a portfolio.

FOR ONLINE READING ONLY

Boolean algebra is a way of representing the relationship between the inputs and outputs of a logic circuit. A logic circuit operates in two values: 1 and 0. Boolean algebra in digital logic circuit differs from ordinary mathematical algebra because a logic circuit involves only two values, which are 1 and 0. In contrast, ordinary mathematics can represent more than two values. Boolean 0 and 1 represent the status of a voltage variable, also known as its logic level, rather than actual numbers. A voltage in a digital circuit is at the logic 0 or logic 1 level depending on the exact numerical values.

Logic expressions and laws

Boolean operations

Boolean algebra has three basic operations: OR, AND, and NOT. The AND and OR gates are linked with addition and multiplication logic operations.

(a) Logical addition

Logical addition uses the logical symbol "+", which can be placed between any number of variables, for example, between A and B which represent the logical value of 1 and 0. The logical addition (+) is represented by the "OR" operator; therefore, the expression A +B reads as "A" OR "B". The possible combinations of logical addition are as follows:
0 + 0 =0, 0 + 1 = 1, 1 + 0 = 1, 1 + 1 = 1.

The logical addition operation represents the same binary addition except for the last entry, whereby when both entries A and B are at logical 1, then A + B = 1. Therefore, the symbol + represents logical addition and not normal binary addition. The logical addition applies to more than one variable, for example, X + Y + Z = P; when values of X, Y and Z are logical values "1", the value of P is "1".

(b) Logical multiplication

Logical multiplication uses the logical symbol "." or AND operator. The possible combination of logical multiplication for variables A and B and the resulting output A.B are as follows:
0.0=0, 0.1=0, 1.0 = 0, 1.1 =1. The logical multiplication represents the same binary multiplication except for the last entry, whereby when both entries A and B are at logical 1, then A.B=1. Therefore, the symbol "." represents logical multiplication. The logical multiplication applies for more than one variable; for example, X.Y.Z = P when X, Y and Z values are logical values "1", the value of P is"1".

Basic laws and rules of Boolean algebra

Laws and rules were invented to help simplify the logic expression and, as a result, minimise the number of logic gates needed to construct various logic circuits. There are different laws of Boolean algebra. We will discuss commutative, associative, distributive, identity, idempotent, annulment, complementary, and double complementary laws.

FOR ONLINE READING ONLY

However, Boolean algebra laws such as commutative, associative, and distributive are all the same as ordinary algebra.

(a) **Commutative law**

The commutative law used in Boolean algebra is the same as ordinary algebra. The addition and multiplication in commutative laws using two variables are as follows:
A+B = B+A and A.B =B.A.

(b) **Associative law**

The associative law used in Boolean algebra is the same as ordinary algebra. The addition and multiplication in associative laws using three variables are as follows:
(A+B) +C = A+(B+C) and (A.B).C = A.(B.C).

(c) **Distributive law**

The distributive law used in Boolean algebra is like ordinary algebra. The addition and multiplication in distributive laws using three variables are as follows:
(A+B).C = (A.C)+(B.C) and A+ (B.C) = (A +B).(A+C)

(d) **Identity law**

The identity law used in Boolean algebra is used as follows:
A+0 = A and A.1 = A

(e) **Idempotent law**

This law applies a variable OR or AND by itself. The output is always the same variable as follows:
A+A=A and A.A= A.

(f) **Annulment law**

Annulment law applies when a variable OR with 1 equals 1 or AND with 0 gives 0 as follows: A.0 =0 and A+1=1

(g) **Complementary law**

Complementary law applies when a variable AND with its complement is 0 and when OR with its complement result is 1.
$\bar{A}.A=0$ and $\bar{A}+A=1$

(h) **Double complementary law**

The double negation law applies when a variable is inverted twice. Its output is the same variable $\overline{\bar{A}}=A$

Boolean algebra expression and simplification

Boolean algebra operations are the core of this simplification of logic circuits. The simplification of Boolean expressions reduces the number of logic gates required to implement the logic circuit. There are several reasons one might want to simplify a logic circuit. In modern circuits the most common reason is to reduce power consumption and/or reduce the amount of heat the circuit generates. This can be important in portable devices and at higher levels of hardware design because power consumption eventually becomes a limiting factor.

Boolean algebra expression

Boolean expressions are algebraic statements that use variables and logical operators, such as AND, OR, XOR, and NOT.

FOR ONLINE READING ONLY

Simplification using Boolean laws

There are different ways used to simplify Boolean expressions. These include the use of the laws of algebra and the Karnaugh map (K-map). This textbook will cover only simplification using the laws of Algebra.

Example 1.1

Using the laws of Boolean algebra, simplify the following expression: (A+ B).(A+C).

Procedure:

(A+B).(A+C)=	
= A.A+ A.C+B.A+ B.C	-Distributive law
= A+A.C +B. A+ B.C	-Idempotent AND law (A.A = A)
= A(1+ C)+ B.A+B.C	-Distributive law
=A+ B.A +B.C	-Annulment OR law (1 + C = 1)
= A(1+ B)+B.C	-Distributive law
= A.1+ BC.	-Annulment OR law (1 + B = 1)
= A+BC.	-Identity AND law (A.1 = A)

Example 1.2

Using laws of Boolean algebra, simplify the following expression: ABC+ A+ AB

Procedure:

$ABC + A\bar{B} + AB\bar{C} = ABC + AB\bar{C} + A\bar{B}$	- Commutative law
$= AB(C + \bar{C}) + A\bar{B}$	- Distributive law
$= AB.1 + A\bar{B}$	- Complimentary law
$= AB + A\bar{B}$	- Identity AND law (A.1 = A)
$= A(B + \bar{B})$	- Distributive law
$= A.1$	- Complementary law
$= A$	- Identity AND law (A.1 = A)

Exercise 1.2

1. Prove the following using Boolean laws and rules:
   (a) AB+AC +ABC = AB +AC
   (b) $A(\bar{A} + B) = AB$
2. Use laws of Boolean algebra to simplify the following expression.
   (a) $ABC + AB\bar{C}$
   (b) $\bar{B}C + BC + AC$
   (c) $B + AB + \bar{A}BC$
   (d) $\bar{A}B\bar{C} + AC + \bar{A}BC$
   (e) $\overline{AC} + AB\bar{C}$
   (f) $\overline{ABC}\bar{D} + \overline{ABCD}$
3. Using the laws of Boolean algebra, verify the Boolean identity $A+(A\cdot B)=AA+(A\cdot B)=A$

FOR ONLINE READING ONLY

Truth tables

A truth table is a table used to represent the Boolean expression of the given logic circuit. It is used to show all possible combinations of the inputs and outputs of a given logic circuit. You must evaluate the Boolean expression for all possible combinations of values for the input variables as part of the method.

Construction of truth tables

The construction of a truth table involves some steps depending on the number of inputs. The following steps are used to construct a truth table:

Step 1: Determine the number of variables for a given Boolean expression, whereby, for n variables, the number of rows will be 2^n. For example, the truth table will need $2^2 = 4$ rows for two variables, A and B. The variables must have all possible combinations of 1 and 0.

Note: When you set up the truth table, always make the first column (whether a 4, 8, or 16-row table) half zeros (0), (1st half) and half ones (1) (2nd half). Before switching to 0's, write half as many consecutive 1's in the next column as you did first. For instance, the truth table that has two variables A and B, as shown in Table 1.3, variable A has two (2) zeros and two (2) ones followed by alternating ones and zeros for variable B.

Table 1.3: *Truth table with two (2) variables*

A	B
0	0
0	1
1	0
1	1

For three variables, A, B, and C, variable A has four (4) zeros and four (4) ones, followed by two (2) ones and two (2) zeros for variable B. Lastly, variable C has alternating zeros and ones as shown in Table 1.4.

Table 1.4: *Truth table with three (3) variables*

A	B	C
0	0	0
0	0	1
0	1	0
0	1	1
1	0	0
1	0	1
1	1	0
1	1	1

Step 2: Identify the variables and every possible combination of 1 and 0 for each variable.

Step 3: Create a new column for each term of the Boolean expression.

Step 4: Compare the two statements.

Note: The two statements are equivalent if their truth tables have the same value.

FOR ONLINE READING ONLY

Example 1.3

Create a truth table for the following expression: X = A + B.

Step 1: Identify the variables in the expression. That is, the two variables are A and B.

Step 2: Determine the number of rows. For two variables, the rows are $2^2 = 4$.

Step 3: Present every possible combination of 1 and 0 for each variable A and B as in Table 1.5

Table 1.5: *Possible combinations of 1 and 0 for variable A and B*

A	B	A+B
0	0	0
0	1	1
1	0	1
1	1	1

Example 1.4

Prove the distributive law (A+B).C = (A+C).(B+C) by using the truth table.

Step 1: Identify variables in the expression. That is, the three variables are A, B and C.

Step 2: Determine the number of rows. For the three variables A, B, and C the number of rows are $2^3 = 8$

Step 3: Present every possible combination of 1 and 0 for each variables A, B, and C as in Table 1.6.

Table 1.6: *Possible combination of 1 and 0*

A	*B*	*C*	*A+B*	*BC*	*AC*	*(A+B).C*	*AC + BC*
0	0	0	0	0	0	0	0
0	0	1	0	0	0	0	0
0	1	0	1	0	0	0	0
0	1	1	1	1	0	1	1
1	0	0	1	0	0	0	0
1	0	1	1	0	1	1	1
1	1	0	1	0	0	0	0
1	1	1	1	1	1	1	1

FOR ONLINE READING ONLY

Writing Boolean expression from the truth table

The truth table is used to describe the Boolean expression of a given logic circuit; therefore, it is possible to write a Boolean expression from the truth table and vice versa. The following steps show how to derive the Boolean expression from the truth table:

Step 1: Write down the Boolean equation for each row in the truth table whose output is 1.

Step 2: To obtain the final Boolean expression, write down the combined Boolean equations with the output one (1) connected by the AND operator and join them with the OR operator.

Step 3: The Boolean expression obtained may not be in the final form. Therefore, it can be simplified using the laws and rules of Boolean algebra.

The following are examples of writing Boolean expressions from the truth table.

Example 1.5

Write the Boolean expression from the truth table as shown in Table 1.7

Table 1.7: *Possible combinations of 1 and 0 for variable A, B, and X*

A	B	x
0	0	0
0	1	1
1	0	1
1	1	1

Step 1: Write down the Boolean equation for each row in the truth table whose output is 1. B, Aand AB

Step 2: To obtain the final Boolean expression, write down the combined Boolean equation whose output is one and by using the OR operator.

$X = \bar{A}B + A\bar{B} + AB$

Step 3: The Boolean expression obtained may not be in the final form, but it can be simplified using the laws and rules of Boolean algebra.

$X = \bar{A}B + A\bar{B} + AB$

$X = \bar{A}B + A\bar{B} + AB + AB$ - Idempotent law

$X = \bar{A}B + AB + A\bar{B} + AB$ - Associative law

$X = (\bar{A} + A)B + A(\bar{B} + B)$ - Distributive law

$X = 1.B + A.1$ - Complementary law

$X = B + A$ - Identity (A.1 = A)

$X = A + B$ - Commutative law

Example 1.6

Write the Boolean expression from the following truth Table 1.8

Table 1.8: *Possible combinations of 1 and 0 for variable A, B, C, and X*

A	B	C	X
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

Solution:

Step 1: Write down the Boolean equation for each row in the truth table whose output is 1.

FOR ONLINE READING ONLY

$\bar{A}BC$, $A\bar{B}C$ and ABC.

Step 2: To obtain the final Boolean expression, write down the combined Boolean equations whose output is one and OR together.

$X = \bar{A}BC + A\bar{B}C + ABC$

Step 3: The Boolean expression obtained may not be in the final form. It can be simplified using the laws and rules of Boolean algebra.

$X = \bar{A}BC + A\bar{B}C + ABC$

$X = \bar{A}BC + A\bar{B}C + ABC + ABC$ - Idempotent law

$X = \bar{A}BC + ABC + A\bar{B}C + ABC$ - Commutative law

$X = (\bar{A} + A)BC + AC(\bar{B} + B)AC$ - Distributive law

$X = 1.BC + 1.AC$ - Complementary law

$X = BC + AC$ - Identity $(A.1 = A)$

$X = C(B + A)$ - Associative law

Activity 1.8:

Designing a car safety buzzer circuit

As an electronic engineer, you are tasked with designing a car safety buzzer circuit. Here are the design specifications: The buzzer should be ON when the door is OPEN or when the key is in the IGNITION, and the seat belt is NOT BUCKLED. Create a truth table to outline these specifications using "highs" (1) and "lows" (0) to indicate sensor or indicator activity.

Exercise 1.3

1. Prove the following using the truth table.
   (a) $AB + AC + ABC = AB + AC$
   (b) $A + (B + C) = (A + B) + C$
2. Create the truth table of the following expression:
   $X = A + \overline{BC}$
3. Given the truth Table 1.9, determine the Boolean expression for Y in terms of A and B.

Table 1.9: *The truth table for A, B, and Y*

A	B	Y
0	0	0
0	1	1
1	0	1
1	1	0

DeMorgan's theorem

There are two DeMorgan's theorems. These are OR sum inverted theorem and AND product inverted theorem. The OR sum inverted version of DeMorgan's theorem states that the complement of two or more variables joined by the OR operator is equal to the complement of individual variables connected by the AND operator. For two variables, A and B, and three variables, A, B and C, the OR sum inverted theorem can be expressed as follows: $\overline{A + B} = \bar{A}.\bar{B}$ and $\overline{A + B + C} = \bar{A}.\bar{B}.\bar{C}$. The AND product inverted version of DeMorgan's theorem states that the complement of two or more variables joined by AND operator is equal to

FOR ONLINE READING ONLY

the complement of individual variables connected by OR operator. For two variables, A and B and three variables, A, B and C, the AND product inverted theorem can be expressed as follows: $\overline{AB} = \overline{A} + \overline{B}$ and $\overline{ABC} = \overline{A} + \overline{B} + \overline{C}$.

Creating a truth table to prove DeMorgan's theorem

The truth table is usually used to prove different Boolean expressions. The truth table in this section is used to prove DeMorgan's theorem. The following examples show the proof of DeMorgan's theorem using the truth table.

Example 1.7

Prove DeMorgan's theorem $\overline{A+B} = \overline{A}\,\overline{B}$, by using truth table.

Solution:

Step 1: The expression has two variables, A and B.

Step 2: Since the expression has two variables, A and B; then, the number of rows are $2^2 = 4$

Step 3: Variables are A and B, and every possible combination of 1 and 0 for each variable is as in Table 1.10:

Table 1.10: *Possible combinations for A and B*

A	B	$\overline{A}$	$\overline{B}$	$A+B$	$\overline{A+B}$	$\overline{A}.\overline{B}$
0	0	1	1	0	1	1
0	1	1	0	1	1	1
1	0	0	1	1	1	1
1	1	0	0	1	0	0

Application of DeMorgan's theorem

DeMorgan's theorem can be used to simplify Boolean expression and create alternative gates from the basic gates. The following steps will guide you to simplify a Boolean expression using DeMorgan's theorem and Boolean algebra laws.

Example 1.8

Simplify the Boolean expression $\overline{A + \overline{BC}}$ using DeMorgan's theorem and Boolean algebra laws.

Solution:

Step 1: Determine the terms you can apply DeMorgan's theorem and treat each term as a single variable. Let $x = A + \overline{BC}$ and $y = \overline{BC}$

Step 2: Using DeMorgan's theorem, the expression $y = \overline{BC}$ will be $y = \overline{B} + \overline{C}$. Therefore, $\overline{A + \overline{BC}} = \overline{A + \overline{B} + \overline{C}}$

Step 3: Using DeMorgan's theorem the expression $\overline{A + \overline{B} + \overline{C}}$ will be $= \overline{A}.\overline{\overline{B}}.\overline{\overline{C}}$

Step 4: Using double negation law, the expression $\overline{A}.\overline{\overline{B}}.\overline{\overline{C}}$ will be $\overline{A}.B.C$

Example 1.9

Simplify the Boolean expression $\overline{\overline{ABC} + \overline{BC}}$ using DeMorgan's theorem and Boolean algebra laws.

FOR ONLINE READING ONLY

Solution:

Step 1: Determine the terms you can apply DeMorgan's theorem and treat each term as a single variable.

Step 2: Using DeMorgan's theorem, the expression $\overline{\overline{A}BC+\overline{B}C}$ will be:

$\left(\overline{\overline{A}BC}\right).\left(\overline{\overline{B}C}\right)$

Step 3: Using DeMorgan's theorem, the expression $\left(\overline{\overline{A}BC}\right).\left(\overline{\overline{B}C}\right)$ will be:

$$\overline{\overline{A}BC+\overline{B}C}=\left(\overline{\overline{A}}+\overline{B}+\overline{C}\right).\left(\overline{\overline{B}}+\overline{C}\right)$$

Step 4: Using laws of Algebra

$$\overline{\overline{A}BC+\overline{B}C}=\left(\overline{\overline{A}}+\overline{B}+\overline{C}\right).\left(\overline{\overline{B}}+\overline{C}\right)$$

$$\overline{\overline{A}BC+\overline{B}C}=\left(A+\overline{B}+\overline{C}\right).\left(B+\overline{C}\right)$$ - Double negation

$$\overline{\overline{A}BC+\overline{B}C}=AB+A\overline{C}+\overline{B}B+\overline{B}\overline{C}+B\overline{C}+\overline{C}\overline{C}$$ - Distributive law

$$\overline{\overline{A}BC+\overline{B}C}=AB+A\overline{C}+\overline{B}\overline{C}+B\overline{C}+\overline{C}\overline{C}$$ - Complementary law

$$\overline{\overline{A}BC+\overline{B}C}=AB+A\overline{C}+\overline{B}\overline{C}+B\overline{C}+\overline{C}$$ - Idempotent law

$$\overline{\overline{A}BC+\overline{B}C}=AB+A\overline{C}+\overline{B}\overline{C}+\overline{C}(B+1)$$ - Distributive law

$$\overline{\overline{A}BC+\overline{B}C}=AB+A\overline{C}+\overline{B}\overline{C}+\overline{C}.1$$ - Annulment law

$$\overline{\overline{A}BC+\overline{B}C}=AB+A\overline{C}+\overline{B}\overline{C}+\overline{C}$$ - Identity law

$$\overline{\overline{A}BC+\overline{B}C}=AB+\overline{C}(A+\overline{B}+1)$$ - Distributive law

$$\overline{\overline{A}BC+\overline{B}C}=AB+\overline{C}(A+1)$$ - Annulment law, $\overline{B}+1=1$

$$\overline{\overline{A}BC+\overline{B}C}=AB+\overline{C}.1$$ - Annulment law, $A+1=1$

$$\overline{\overline{A}BC+\overline{B}C}=AB+\overline{C}$$ - Identity law

FOR ONLINE READING ONLY

Exercise 1.4

1. Use DeMorgan's theorem and laws of Boolean algebra to simplify the following expressions:
   (a) $\overline{ABC+A\overline{B}\overline{C}}$
   (b) $\overline{\overline{B}C+BC+AC}$
   (c) $B+AB+\overline{A}BC$
   (d) $\overline{A}B\overline{C}+AC+\overline{A}BC$
2. Use the truth table to prove the following DeMorgan's theorem:
   $\overline{A.B}=\overline{A}+\overline{B}$
3. Demonstrate DeMorgan's theorem using a truth table for the expression:
   $\overline{ABC}=\overline{A}+\overline{B}+\overline{C}$

Logic gates

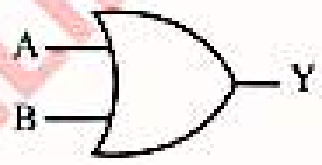
A logic gate is an electronic circuit that involves two or more binary input signals and one binary output signal. These gates are critical when dealing with electronics. They are used to construct the digital system. The logic gates can perform the logic operation in single or multiple binary inputs and give one binary output. Logic gates are considered as a single unit; however, gates comprise one or more transistors.

Standard logic gates symbols

The basic logic gates include OR, AND, NOT, NAND, NOR and XOR gates. The NOT gate has only one input signal and one output signal while other gates have two or more input signals. The standard logic gates symbols and operation of each logic gate are as follows:

(1) OR gate

OR gate is the digital circuit with two or more input signals, and its output signal is the OR combination of its input signal. The logical symbol of the OR gate is shown in Figure 1.12 (a). Figure 1.12 (b) illustrates the truth table for OR gates. The OR gate is expressed mathematically using the (+) sign between the variables; for example, Boolean operations Y= A+B represent the OR gate whereby the output of the OR gate will be 1 when either of the input is 1 or both inputs are 1. The OR gate output will only be 0 if both inputs are 0. The same idea is used for more than two inputs. The output will only be 0 if both inputs are 0; otherwise, the output is 1.



(a)

A	*B*	*A+B*
0	0	0
0	1	1
1	0	1
1	1	0

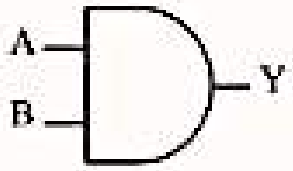
(b)

Figure 1.12: *(a) OR gate, and (b) The truth table for OR gate*

(2) AND gate

AND gate is the digital circuit with two or more input signals, and its output signal is the AND combination of its input signal. The logical symbol of the AND gate is shown in Figure 1.13 (a). Figure 1.13(b) illustrates the truth table for AND gates for Boolean operations Y= A.B whereby the output of the AND gate will be high (1) when both inputs are high (1); otherwise, the output is 0. The same idea is used for more than two inputs. The output will only be one (1) if both inputs are 1; otherwise, the output is 0.

FOR ONLINE READING ONLY



A	B	A.B
0	0	0
0	1	0
1	0	0
1	1	1

(a) (b)

Figure 1.13: *(a) AND gate, and (b) The truth table for AND gate*

(3) NOT gate (Inverter)

Unlike OR and AND gates, the NOT gate is the digital circuit with only one input signal and one output signal, and its output signal complements its input signal. The logical symbol of the NOT gate is shown in Figure 1.14 (a). Figure 1.14 (b) illustrates the truth table for NOT gates. If the input signal A is subjected to NOT gate, its output is A.



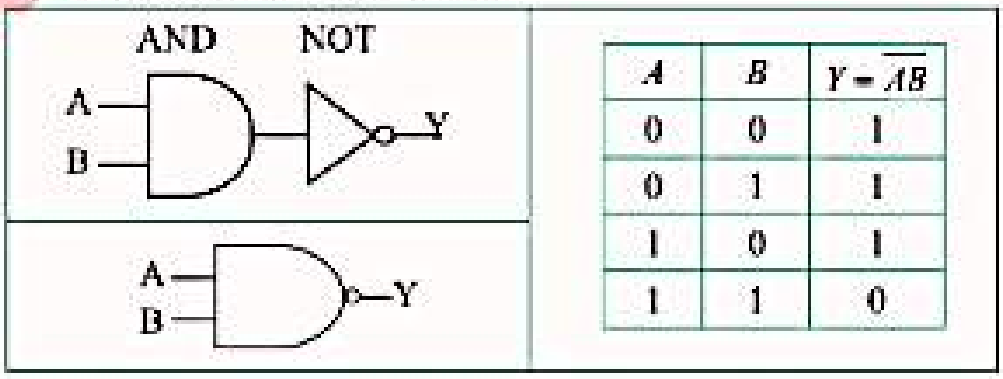
A	$\bar{A}$
1	0

(a) (b)

Figure 1.14: *(a) NOT gate, and (b) The truth table for NOT gate*

(4) NAND gate

The term NAND stands for NOT-AND and refers to the complement of AND gate output. It works oppositely to the AND gate. The NAND gate has two or more input signals, and its output signal complements the AND gate. The logical symbol of the NAND gate is shown in Figure 1.15 (a) . Figure 1.15 (b) illustrates the truth table for NAND gates which shows that the output will be 0 only when both inputs are 1. When any input is 0, the output will be 1.



A	B	$Y = \overline{AB}$
0	0	1
0	1	1
1	0	1
1	1	0

(a) (b)

Figure 1.15: *(a) NAND gate, and (b) The truth table for NAND gates*

FOR ONLINE READING ONLY

Note: Using DeMorgan's theorem, you can create other gates from basic gates. For example, the NAND gate is the same as the OR gate with inverted inputs as shown in Figure 1.16.

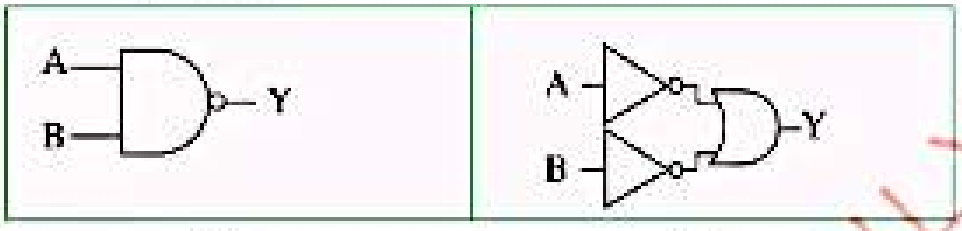
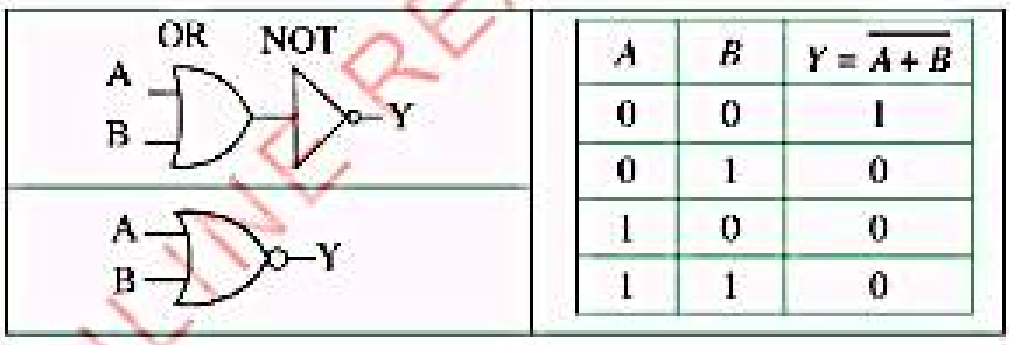


Figure 1.16: *(a) NAND gate, and (b) OR gate with inverted inputs*

(5) NOR gate

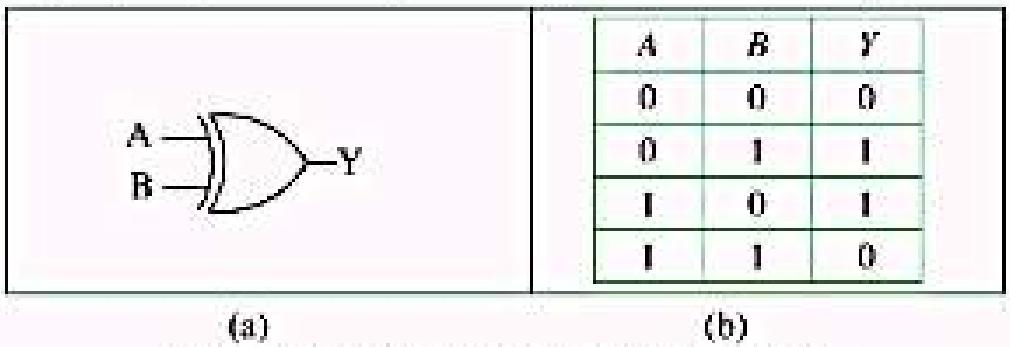
The term NOR stands for NOT-OR, which refers to the complement of OR gate output. It works oppositely to the OR gate. The NOR gate has two or more input signals, and its output signal complements the OR gate. The logical symbol of the NOR gate is shown in Figure 1.17 (a). Figure 1.17 (b) illustrates the NOR gate's truth table which shows that the output will be 0 when any inputs are 1. When both inputs are 0, the output will be 1.



A	B	$Y = \overline{A + B}$
0	0	1
0	1	0
1	0	0
1	1	0

Figure 1.17: *NOR gate and truth table for NOR gate*

The logical symbol of the exclusive OR (XOR) gate is shown in Figure 1.18 (a). The XOR has only two inputs, and its output is one (1) only when two inputs are at opposite inputs, that is, 1 and 0. Figure 1.18 (b) illustrates the XOR gate's truth table.



A	B	Y
0	0	0
0	1	1
1	0	1
1	1	0

Figure 1.18: *XOR gate and truth table for XOR gate*

FOR ONLINE READING ONLY

Activity 1.9:

Simulation of logic gate circuit using Logisim simulator

Aim: To enhance comprehension of digital logic design and Boolean algebra applications in real-world electronic circuits

Materials: Computer, Logisim simulator software or Internet , sample designed logic gate circuits

Instructions:

Step 1: Design a circuit diagram by first, sketching it on piece of paper.

Step 2: Launch Logisim simulator and understand its components, as shown in Figure 1.19.

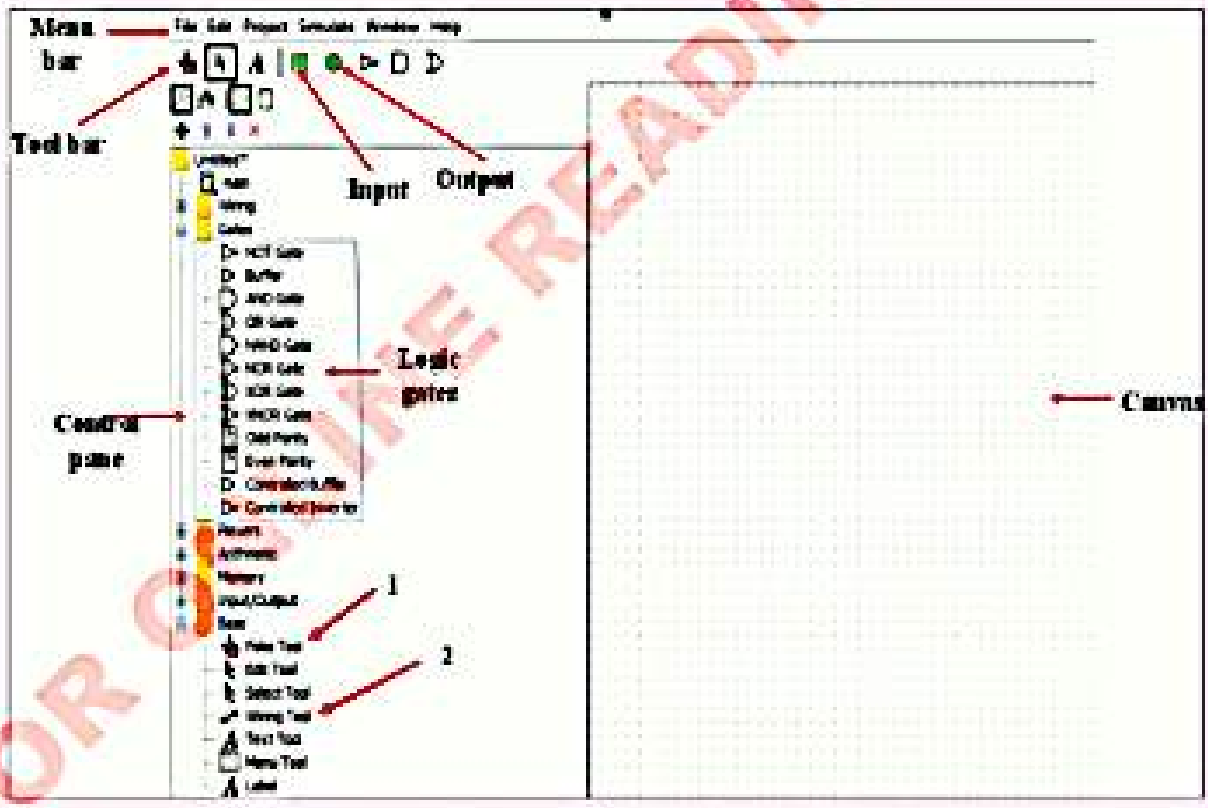


Figure 1.19: *Interface of a Logisim simulator*

Step 3: Adding gates

(a) Add the AND gates by clicking on the AND tool in the toolbar. as shown in Figure 1.20 and drag it into the canvas

(b) Add two inputs and one output by clicking on the input and output tools as shown in Figure 1.20

FOR ONLINE READING ONLY

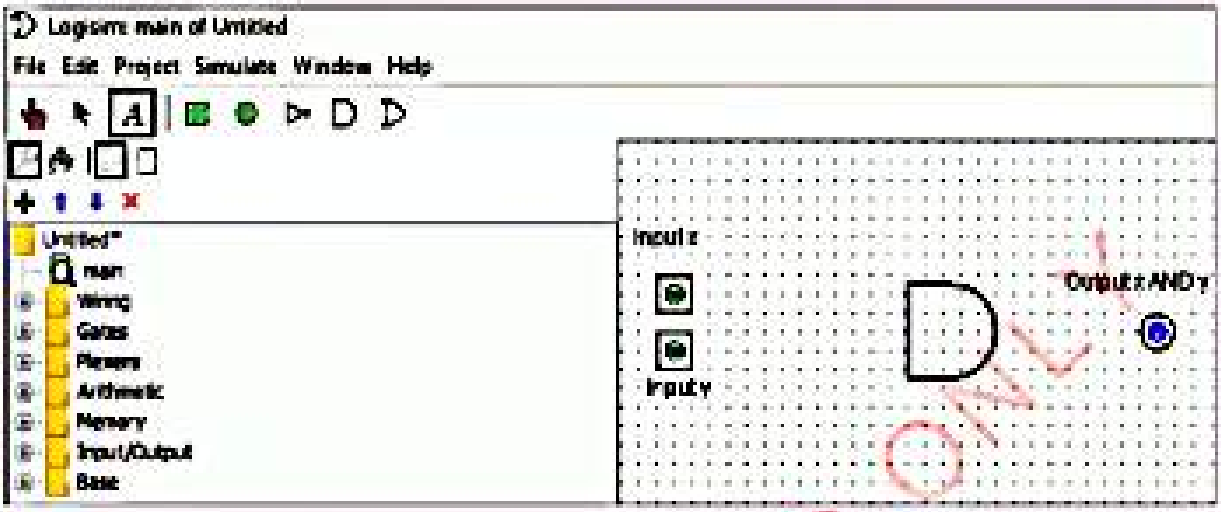


Figure 1.20: *Adding input and output buttons*

Step 4: Add wires (connectors):

(a) Click on the Plus sign in the folder "Base" to expand and display the available tools.

(b) Click and hold "Wiring Tool" (2) as displayed in Figure 1.21and drag to connect Input x and Input y to the AND gate.

(c) Click and hold "Wiring Tool" to drag and connect the AND gate to the output x AND y.

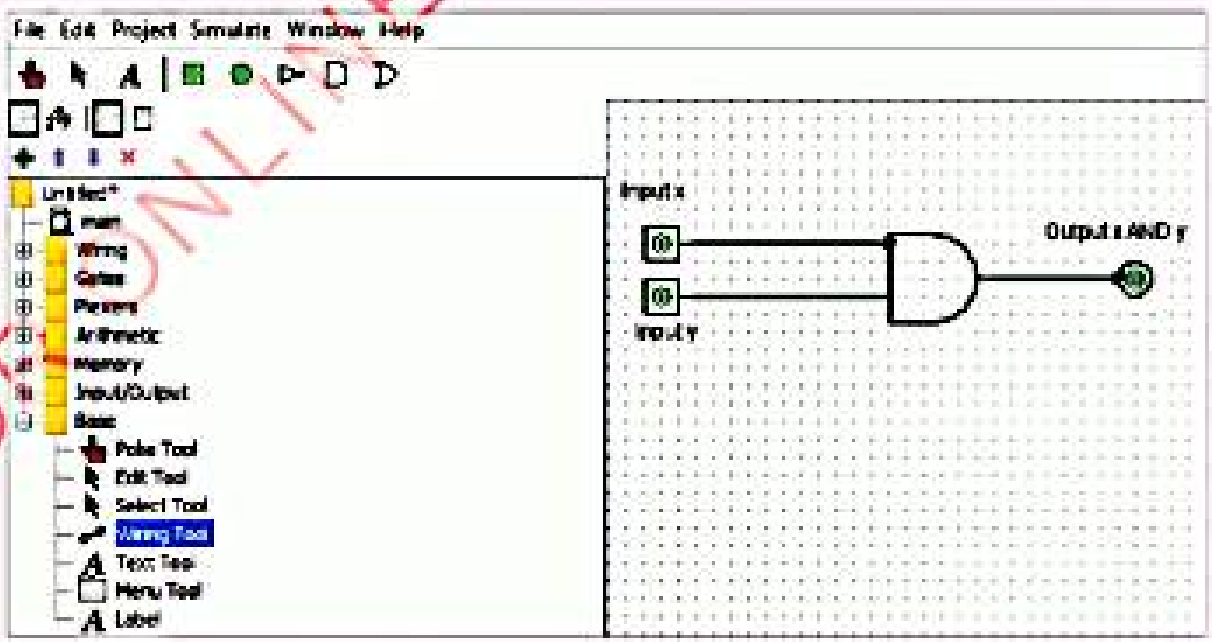


Figure 1.21: *Adding connector*

Step 5: Testing your circuit: select the poke tool (1) as labelled in Figure 1.22 and poke the inputs by clicking on them.

FOR ONLINE READING ONLY

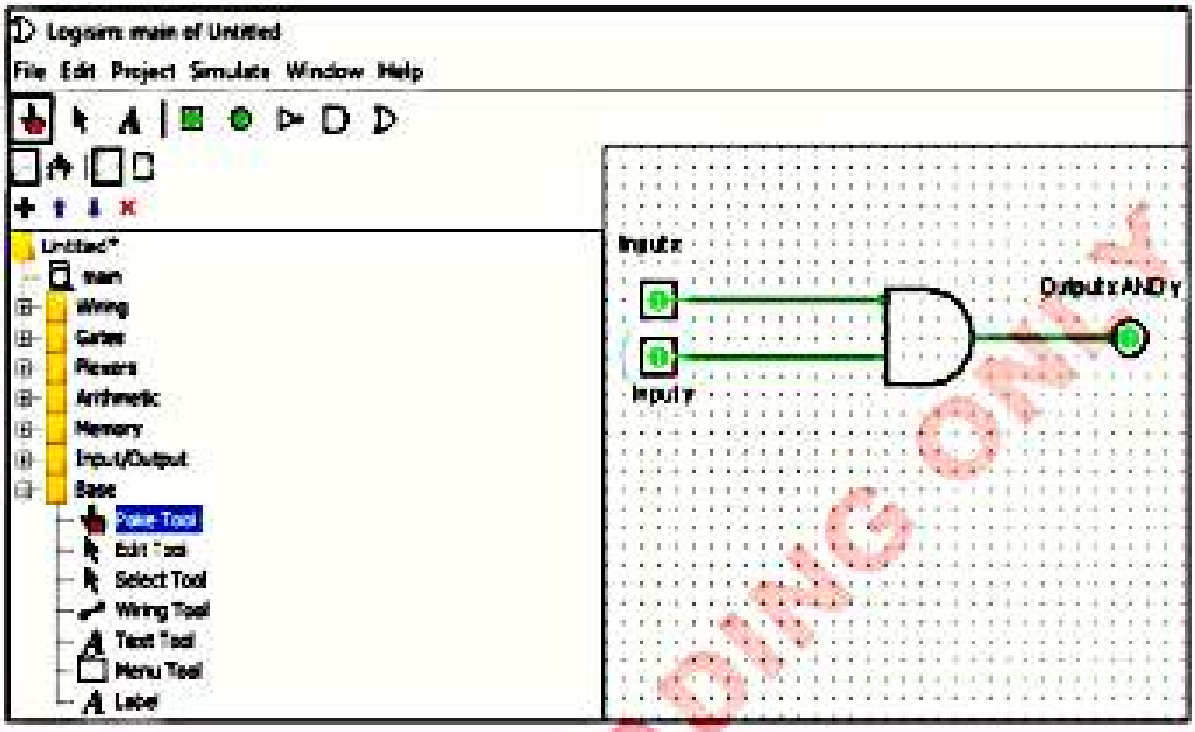


Figure 1.22: *Testing circuit*

Step 6: Compare your test results with the outputs in truth Table 1.11

Both contain 0s, the output as well contains 0. This already tells us that the circuit already computes a 0 when both inputs are 0. Now try another combination of inputs by selecting the poke tool and start poking the inputs by clicking on them. Each time you poke an input, its value will change.

Table 1.11: *Truth table for comparing test results*

x	y	x AND y
0	0	0
0	1	0
1	0	0
1	1	1

Questions:

1. Provide descriptions on what you have observed.
2. Explain if the output of the circuit test from a simulator gives the same results as in Table 1.23.

FOR ONLINE READING ONLY

Step 3: Adding wires (connectors)

Select the wiring tool in the toolbar and drag it from one position to another in the canvas area as shown in Figure 1.25, and a connection will start to appear between the two points.

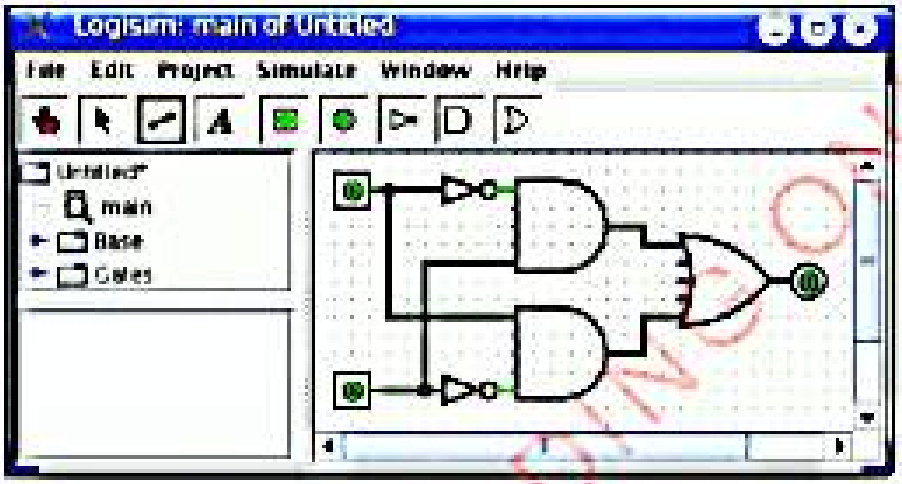


Figure 1.25: *Adding connectors*

Step 4: Testing your circuit, click Light on and Light off

Both contain 0s, the output as well contains 0. This tells us that the circuit already computes a 0 when both inputs are 0. Now, try another combination of inputs by selecting the poke tool shown in Figure 1.26 and start poking the inputs by clicking on them. Each time you poke an input, its value will change.

(a) Light on

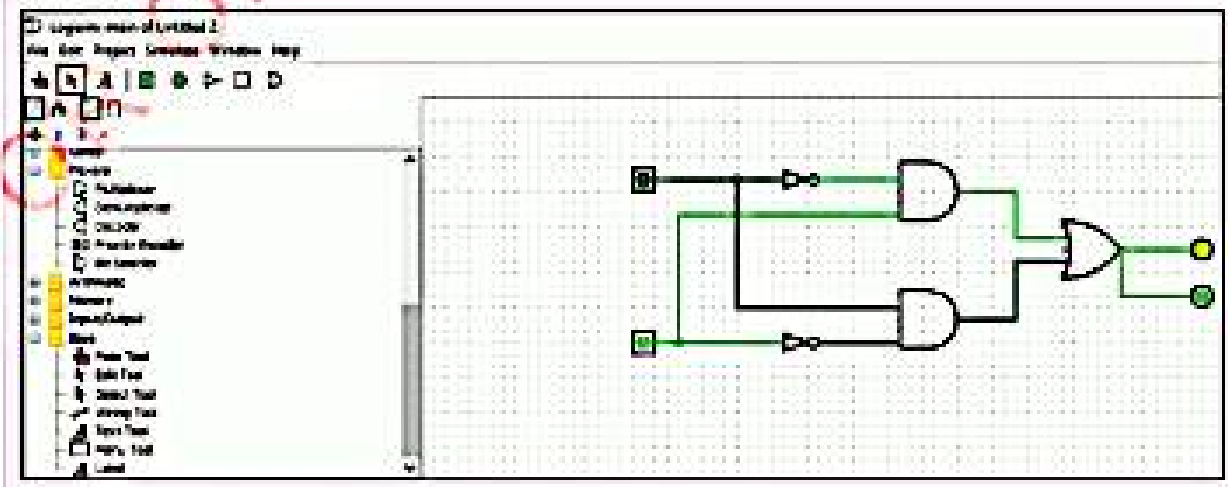


Figure 1.26: *Putting Light on*

FOR ONLINE READING ONLY

(b) Light off

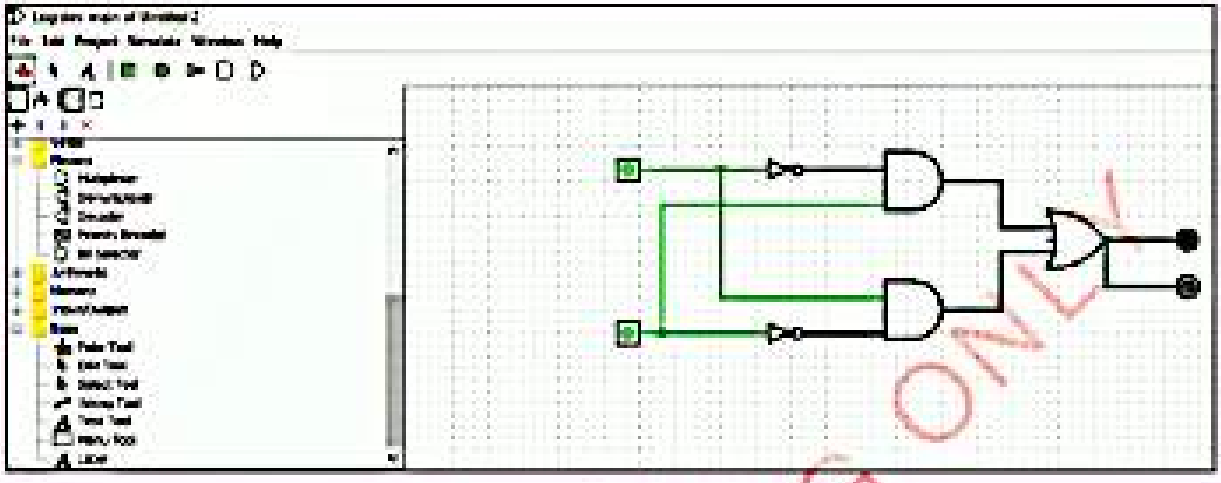


Figure 1.27: Pulling Light off

Question:

1. What did you observe during your exploration?
2. Describe your experience when preparing a truth table, sketching a common circuit diagram, and testing the circuit in a simulator.

Activity 1.11:

The relationship between Boolean operation and logical gates

Two electrical switches, a bulb and a battery are connected as shown in Figure 1.28. As a technical expert, you are required to establish the truth table to reflect the circuit operations that control switches A and B, which will light up the bulb, and then propose the logic gate for the circuit. The following conditions will guide you:

(a) What happens when both switches are OFF? ;

(b) What happens when only one switch is ON (either A or B)? ; and

(c) What happens when both switches are ON?

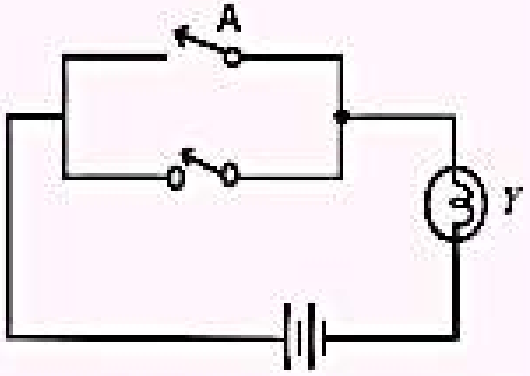


Figure 1.28: Electric circuit with two switches

(d) Simulate this circuit by using a Logisim simulator

Application of logic gates

Logic gates is the fundamental backbone of the digital electronic industry. The design and operation of digital electronic devices are based on digital logic circuits. These devices make use

FOR ONLINE READING ONLY

of one or another type of digital logic circuit for their operation. In digital circuits, the input signals can have only one two values, that is logic 0 (lower voltage level) and logic 1 (higher voltage level). At the most basic level, digital electronics is based on binary number, and it works with '1's and '0's to process the information. The number system used in digital systems is "ON" and "OFF", which is represented by '1's and '0's. Digital circuits are designed using various types of digital components, such as logic gates.

Modern life depends much on the use of digital devices: hence the application of logical gates becomes vital in the design and manufacturing these devices. An example in which logic gates are applied is the design of a logic circuit for the control system for a car engine, computer architecture, and communication systems.

Exercise 1.5

1. Design a logic circuit using AND, OR, and NOT gates to implement the following Boolean expression: $A \cdot (B + \overline{C})$. Draw the circuit diagram and explain each step.
2. Explain how you could use logic gates to create a simple security system that turns on an alarm (output) only when two conditions are met: the door is open (input A) and the motion sensor is triggered (input B). Draw the circuit and describe how it works.
3. Use a logic circuit simulator to create and test a circuit that represents the following logic function: $\overline{(C + B \cdot C}$. Provide screenshots of your circuit design and simulation results, and discuss any challenges you faced.
4. Use TinkerCad to simulate the application of logic gates and verify if the results match the expected behavior according to the principles of logic gates.

Logic circuits

Concept of the logic circuit

A logic circuit combines two or more logic gates, such as OR, AND, and NOT gates. Examples of logic circuits include adders, subtractors, multiplexers, and demultiplexers. These circuits perform complex digital operations by processing binary inputs to produce specific outputs.

Types of a logic circuit

There are two types of logic circuits: combinational and sequential. Combinational logic circuits are made up of logic gates and produce an output based on the current input, without any memory. They are commonly used in CPU components like the Arithmetic Logic Unit. Sequential logic circuits, on the other hand, include both a combinational logic circuit and memory. They are used for designing memory components like registers and memory. However, further details about

FOR ONLINE READING ONLY

sequential logic circuits shall not be discussed in this book. The combinational and sequential logic circuit block diagram is illustrated in Figure 1.29.

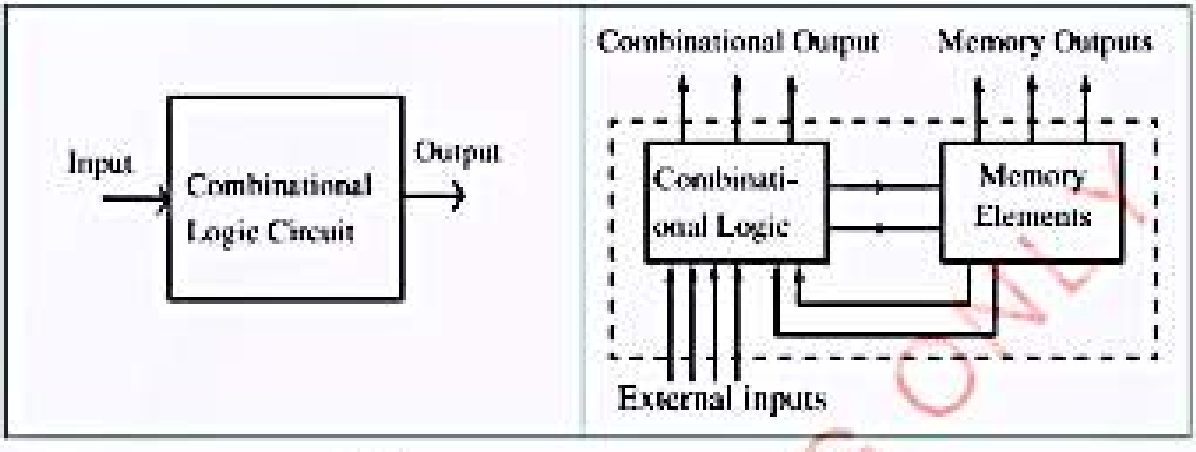


Figure 1.29: (a) Combinational logic circuit, and (b) Sequential logic circuit

Designing a combinational logic circuit for a given Boolean expression

Boolean algebra provides a simple approach to define the operation of a logic circuit which is constructed by a combination of two or more logic gates. The output of the logic circuit is due to a combination of various input values.

Drawing logic circuits from the Boolean expression

The Boolean expressions are usually written to describe the mathematical behaviour of the logic circuit. Therefore, it can determine how the input and output signals are connected in the logic circuit. For example, you will realise that two inputs OR gates are needed to represent an expression Y = A + B. Therefore, using the same analysis, you can create a complex circuit. The essential steps required to create a logic circuit from the Boolean expression include crosschecking it and translating it into logic gates. The following examples illustrate the conversion of a Boolean expression to a logic circuit.

Example 1.10

Create a logic circuit of the Boolean expression Y = A+BC.

Solution:

Step 1: Determine the number of input variables A, B and C.

Step 2: Identify the operators required in the expression.

(a) One AND gate for BC required.

(b) One OR gate is needed to combine A and BC in Figure 1.30.

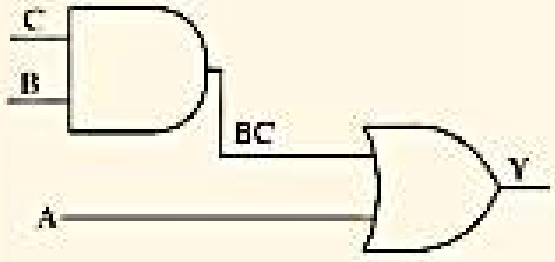


Figure 1.30: Operators for A and BC

FOR ONLINE READING ONLY

Example 1.11

Given the Boolean expression

$Y = AC + ABC + A\overline{BC} + A\overline{B}$.

(a) Simplify the expression using Boolean algebra

(b) Draw the logic circuit of the expression obtained in (a).

Step 1: Simplify the Boolean expression

$$Y = AC(1+B) + A\overline{B}(\overline{C}+1)$$

$$Y = AC.1 + A\overline{B}.1$$

$$Y = AC + A\overline{B}$$

$$Y = A(\overline{B}+C)$$

Step 2: Determine the number of input variables A, B and C.

Step 3: Identify the operators required in the expression.

(a) One NOT gate is required $\overline{B}$.

(b) One OR gate is needed to combine C and $\overline{B}$.

(c) One AND gate is required for A and $\overline{B}+C$, as shown in Figure 1.31.

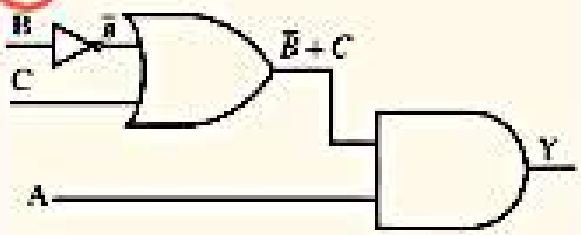


Figure 1.31: *Logic circuit for* $Y = A(\overline{B}+C)$

Creating Boolean expressions from logic circuits

You can create Boolean expressions from the logic circuit by writing the expression for each gate at the leftmost inputs and working to the last output. For example, the Boolean expression of the logic circuit in Figure 1.32, can be determined using the following steps:

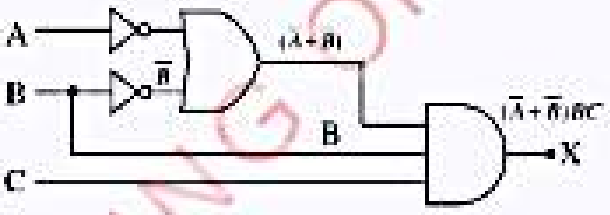


Figure 1.32: *Logic circuit*

Step 1: The expression for left most NOT gate with inputs A and B is $\overline{A}$ and $\overline{B}$, respectively.

Step 2: The output of the left most NOT gates is the OR gate's input. Therefore, the expression for OR gate is $\overline{A}+\overline{B}$.

Step 3: The output of the OR gate is one of the inputs to the rightmost AND gate, and B and C are other inputs. Therefore, the expression for AND gate is $(\overline{A}+\overline{B})BC$. The output expression for the entire logic circuit is $(\overline{A}+\overline{B})BC$.

Exercise 1.6

1. Create a logic circuit of the Boolean expressions:

   (a) $AB + B\overline{C}$

   (b) $\overline{B}C + (B + AC)$

   (c) $AB + \overline{A}BC$

FOR ONLINE READING ONLY

2. Determine the Boolean expression and draw the truth table from the following circuit in Figure 1.33

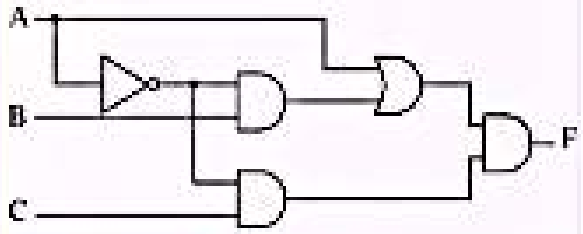


Figure 1.33: *Determining Boolean expression*

3. Design a combinational circuit with three input A, B, and C and three outputs X, Y, and Z. When the binary input is 0, 1, 2, and 3 the binary output is too greater than the input, when the binary input is 4, 5, 6, 7, the binary output is three less than the input.
   (a) Construct a truth table for that scenario.
   (b) Derive the Boolean expression for the output.
   (c) Simplify the Boolean expression for output obtained in part (b).
   (d) Draw a logic circuit for that simplified expression in part (c).

Construction of half and full adder

An adder is a digital circuit that adds binary values together. Adders are used in many computers and other types of processors to execute a variety of computations such as calculating address table indices in the ALU. There are two types of adders, namely half adder and full adder. A half adder is a logic circuit that adds two binary digits together. The half adder has two outputs which are the Sum (S) and Carry (C). This limitation of a half adder can be overcome using a full adder. The full adder is the logic circuit with a condition to add a carry. Therefore, it adds three binary digits together. Adders are the circuits within the ALU in the CPU that perform binary addition. Therefore, they are vital circuits used in computers for number addition.

Construction of half adder

This simple addition consists of four possible operations which are:
$0 + 0 = 0$, $0 + 1 = 1$, $1 + 0 = 1$, and $1 + 1 = 10$.

The first three operations result in a single-digit sum while the fourth operation results in a two-digit sum. A carry is the most significant bit in this operation. As a result, the first three operations carry '0' whereas the fourth produces a carry of '1'. Table 1.12 shows the truth table for the Boolean realisation of the binary addition.

Table 1.12: *Half adder Table*

Input		Output	
A	*B*	Sum(S)	Carry (C)
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

From the truth table, you obtain the Boolean expression for S and C as follows:

$$S = \overline{A}B + A\overline{B} = A \oplus B \text{ and } C = AB$$

The Boolean expressions show that sum and carry are equivalent to XOR and AND gate, respectively. The logic

FOR ONLINE READING ONLY

circuit for the half adder is illustrated in Figure 1.34 which is simplified using the XOR gate as shown in Figure 1.35.

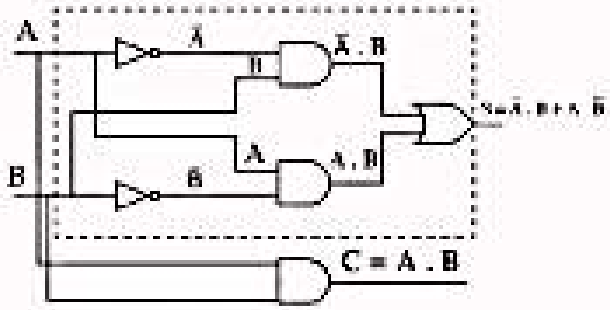


Figure 1. 34: *Half adder logic circuit*

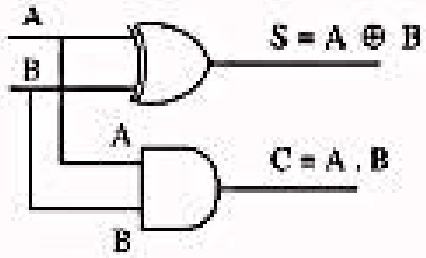


Figure 1. 35: *Half adder logic circuit using XOR and AND gates*

Construction of full adder

A full adder adds three input bits. There are three inputs and two outputs on these circuits. The two significant bits to be added are represented by two inputs while the third input represents the carry from the previous significant position. Table 1.13 shows the truth table for the full adder.

Table 1.13: *Full adder truth table*

Input			Output	
A	B	C_1	Sum(S)	C_2
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

From the truth table, you obtain the Boolean expression for sum (S) and carry (C_2) as follows:

$$S = \overline{A}BC_1 + A\overline{B}C_1 + ABC_1 + A\overline{B}\,\overline{C_1}$$

$$S = \overline{A}BC_1 + ABC_1 + A\overline{B}C_1 + A\overline{B}\overline{C_1}$$

$$S = C_1(\overline{A}B + AB) + \overline{C_1}(\overline{A}B + A\overline{B})$$

$$S = C_1\overline{(\overline{A}B + A\overline{B})} + \overline{C_1}(\overline{A}B + A\overline{B})$$

$$S = C_1(\overline{A \oplus B}) + \overline{C_1}(A \oplus B)$$

$$S = C_1 \ominus (A \oplus B) \text{ and } C_2 = AB + AC_1 + BC_1$$

The Boolean expressions show that sum (S) and carry (C) are equivalent to XOR ($S = C_1 \oplus A \ominus B$) and ($C_2 = AB + AC_1 + BC_1$) gate. The logic circuit for the full adder is illustrated in Figure 1.36.

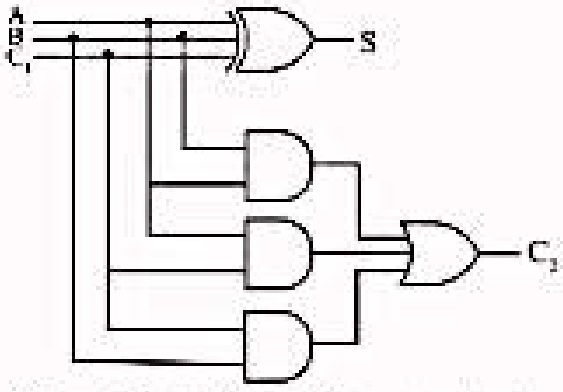


Figure 1.36: *Full adder logic circuit using three inputs XOR, OR and three AND gates*

The Boolean expression for a carry-out (C_2) can be simplified using XOR as follows:

$$C_2 = AB + AC_1 + BC_1$$

$$C_2 = AB + AC_1(\overline{B} + B) + BC_1(\overline{A} + A)$$

$$C_2 = AB + AC_1B + ABC_1 + AC_1\overline{B} + BC_1\overline{A}$$

$$C_2 = AB(1 + C_1 + C_1) + AC_1\overline{B} + BC_1\overline{A}$$

FOR ONLINE READING ONLY

$$C_2 = AB(1 + C_1 + C_1) + C_1(A\overline{B} + \overline{A}B)$$

$$C_2 = AB + C_1(A\overline{B} + \overline{A}B)$$

$$C_2 = AB + C_1(A \oplus B)$$

The full adder can be simplified using the two half adders, as shown in Figure 1.37.

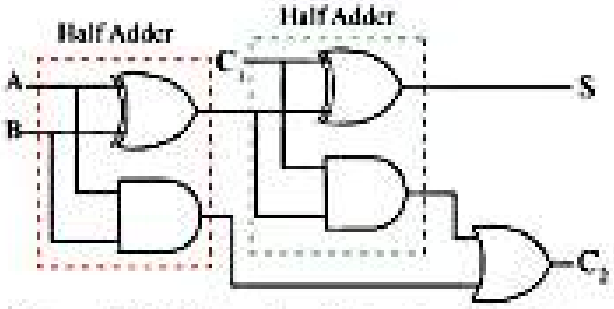


Figure 1.37: *Full adder logic circuit using two half adders and OR gate*

Exercise 1.7

1. Draw the truth table of the half adder and determine its Boolean expression.
2. Explain why Boolean algebra is important in the digital system.
3. Using the laws of Boolean algebra, simplify the following expressions.
   (a) $AB + A\overline{B} + B\overline{C} + \overline{A}C$
   (b) $\overline{A} + AB + \overline{C} + \overline{B}C$
   (c) $A + \overline{A}B$
   (d) $\overline{A}B\overline{C} + AC + ABC$
4. Given the Boolean expression $Y = AB + ABC + A\overline{B}\overline{C} + A\overline{B}$
   (a) Draw the logic circuit of the Boolean expression
   (b) Simplify the Boolean expression
   (c) Draw the logic circuit of the simplified Boolean expression.
5. From the following Boolean expression, simplify and then draw a logic circuit for the simplified expression.
   (a) $Y = A(B + AB) + AC$
   (b) $Y = A(A + B)(\overline{A} + \overline{B})$
   (c) $Y = \overline{A}B\overline{C} + \overline{A}C\overline{C} + ABC$

FOR READING ONLY

Concept of computer memory

Scenario 1.3: *Book storage in the library system*

In any typical library, books are organised on shelves by subjects like Physics, Mathematics, and Computer Science. This is similar to memory hierarchy in computers, where data is stored at different levels based on accessibility and speed. Signs and labels help people locate books easily, similar to memory addresses used to access specific data on computers. Frequently accessed books are placed closer for quick access, much like cache memory in computers storing recently accessed data for faster retrieval. Books on shelves represent primary memory for immediate access, while additional copies in storage rooms or archives represent secondary memory.



Questions:

1. Based on the scenario you have read, what is the importance of signs and labels in the library? Explain by relating this to how memory is organised and accessed in a computer.
2. Explain how useful is the idea of using shelves in the library. Analyse how this idea relates to the organisation and management of computer memory.

The Meaning of Computer Memory

Computer memory is the internal storage of a computer system that holds data, instructions, and processed results during data processing. It is crucial for the proper functioning of a computer, enabling the system to temporarily store and process data as needed. The memory hierarchy includes fast memories like cache, registers, and RAM for temporary data storage, and slower secondary memory devices like magnetic disks for permanent data storage. This organisation aims to balance high performance with minimal cost.

Computer Memory Design Principles

The design of a computer memory is based on two main principles: locality and cost-performance ratio.

Principle of locality

As you sit at the library desk, gathering books for your assignment on computer architecture classification, you realise that while most topics are covered, there is nothing on Flynn's taxonomy. You return to the shelves, find a suitable book, and arrange them on the desk. With this selection, you are likely to find what you need without constant trips to the shelves. This mirrors the *principle of locality*, where frequently accessed data is stored closer for quicker retrieval. Computer programs do not access all data at once, which helps them run faster and use resources better. The principle of locality states that programs access a relatively small portion of their data and instructions at any instant of time, just as you access a very small portion of the library's collection. There are two main types of locality:

(a) *The principle of temporal locality which states that a program is likely to reuse recently accessed data or instructions soon again.*

This principle is like highlighting key points in your textbook and then referring back to them a couple of times to understand a concept. Similarly, a computer program might use a piece of data once for a calculation, then need it again right away for another step. So, the program keeps that data easily

FOR ONLINE READING ONLY

accessible for a short period, just like you keep your highlighted section open for reference.

(b) *The principle of spatial locality* which states that a program is likely to access data or instructions *close to those it just accessed.*

Assume you highlighted a section on computer memory hierarchy and realized you missed something about cache memory. You wouldn't close your book and search the entire library. You would probably just flip back a few pages because the information you need is likely close to what you were just looking at. Similarly, if a computer program needs to access one piece of information in memory, it might also need the information stored right next to it. So, the computer tries to keep such neighbouring data close together for faster access, just like books on similar topics are shelved together in a library.

The principle of cost-performance ratio

As you sit at the library desk, you may wish to instantly access all the books in the library. The library would not be practical if every single book was right next to your desk. A big room filled with individual shelves for each book would make it easier to access any book instantly but it would come at a cost. This concept applies to computer memory as well. We can design a system where all data is readily available for the processor, but it comes at a cost.

The principle of cost-performance ratio states that faster and smaller memories are more expensive and consume more *power.*

Imagine each book being a small, personalised booklet containing only the information you need right now. These booklets would be very fast to access since they are right at your fingertips. However, creating and maintaining individual booklets for every piece of information would be very expensive and impractical. Slower and larger memories are cheaper and more energy-efficient. The big library shelves represent large, slower storage. It takes time to find a specific book, but storing all the information in this way is more cost-effective.

Computer architects use the principle of cost-performance ratio to create a memory hierarchy. This hierarchy consists of different levels of memory, each with its own speed, size, and cost:

(a) Registers: These are the fastest and most expensive memory units, located within the CPU itself, like having a tiny personalized booklet for the most frequently used information.

(b) Cache: This is a smaller, faster memory closer to the processor, like having a selection of relevant books on your desk for easy reference.

(c) Main Memory (RAM): This is the primary memory, like the library shelves readily accessible to you.

FOR ONLINE READING ONLY

(d) *Storage (Hard drive):* This is the largest and slowest memory for permanent storage, like a library's back room where they keep extra books and materials you do not see on the main shelves.

By combining different memory types, computer architects achieve a balance between speed, cost, and power consumption. Frequently accessed data resides in faster, more expensive memory, while less frequently used data is stored in slower, cheaper memory options. This approach ensures efficient use of resources and high performance for your computer programs.

Memory hierarchy and interfacing

Memory hierarchy optimises computer performance by using various memory levels with distinct speeds and capacities, ensuring the processor accesses data as quickly as possible. Interfacing between these levels manages data transfer, starting from the fastest memory (cache and registers) to the slower, larger storage levels.

Computer memory hierarchy

The computer memory hierarchy, shown in Figure 1.38, uses multiple levels of memory with different access times and capacities. Faster memory, closer to the processor, is smaller and more expensive, while slower memory, further away, offers larger, and cheaper storage. This design optimises performance by storing frequently accessed data in faster memory for efficient processing. In summary each type of memory within the hierarchy offers a trade-off between the following performance parameters:

(a) *Speed*: faster memory allows for quicker access to data, but it is typically smaller and more expensive.

(b) *Size:* larger memory holds more data, but it may be slower to access.

(c) *Cost:* faster and smaller memories are generally more expensive per unit of storage compared to larger, slower memories.

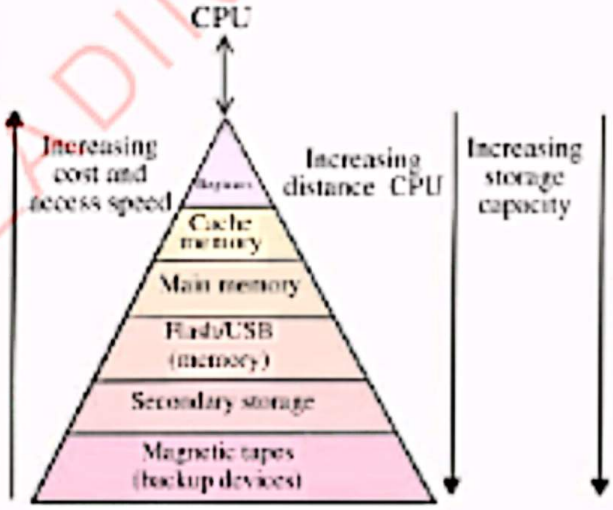


Figure 1.38: *Computer memory hierarchy*

Computer memory interfacing

The computer memory hierarchy uses various memory levels with distinct speeds and capacities to optimise performance. The processor relies on the fastest available memory (cache and registers) for data access, but these levels have limited capacity. Therefore, data must be copied from lower levels when needed by the processor. This communication between memory levels, as shown in Figure 1.39, is managed by memory hierarchy interfacing.

FOR ONLINE READING ONLY

When the processor requests data, the cache controller first checks the cache. If the data is found, a cache hit occurs, and the data is quickly delivered to the processor. If not, a cache miss occurs, and the cache controller then checks the main memory. If the data is found in main memory, a main memory hit occurs, and the data is transferred to the cache and then to the processor. If the data is not in the main memory, a main memory miss occurs. The data is retrieved from storage, transferred to the main memory, then to the cache, and finally delivered to the processor.

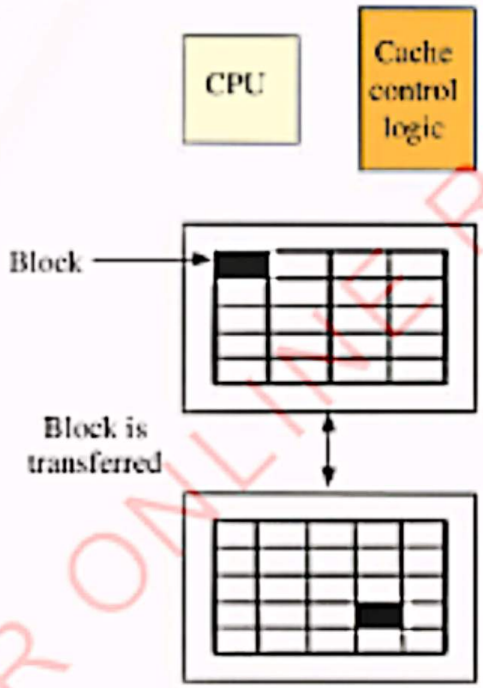


Figure 1.39: *Memory interfacing between cache memory and main memory*

Memory hierarchy and interfacing are enabled by the following key concepts;

Unit of information transfer: Memory is organised into fixed-size blocks. A memory hierarchy can consist of multiple levels, but data is copied between only two adjacent levels at a time. A block is the fixed-size chunk of data that the cache transfers between itself and other memory levels.

Cache line: A unit of data storage within the cache. It acts like a container that holds a block and is specifically designed for efficient data transfer between the cache and other memory levels.

Cache hit: An event in which specific data or instructions are searched and found available in the accessed cache memory. In this scenario, the cache transfers (copies) the requested data to the processor.

Cache miss: Event in which specific data or instructions is searched and not found in the accessed cache memory. In this scenario, the data is fetched from lower levels and transferred to the processor.

Hit rate: the percentage of times the data is found in the cache without needing to access lower levels. A higher hit rate indicates efficient data retrieval.

$$\text{Hit rate} = \frac{\text{Total hits}}{\text{Total hits} + \text{Total misses}}$$

Miss rate: the percentage of times the data is not found in the cache and needs to be fetched from lower levels. A lower miss rate is desirable.

$$\text{Miss rate} = \frac{\text{Total misses}}{\text{Total hits} + \text{Total misses}}$$

Miss penalty: the extra time it takes to access data from a lower level compared

FOR ONLINE READING ONLY

to the cache. Higher levels in the hierarchy have larger miss penalties.

Write operations: when data needs to be written, the write operation might update all copies across different levels (cache and main memory) to maintain consistency (depending on the written policy).

Cache coherence protocols: mechanisms are employed to ensure all copies of data across different levels are consistent, especially when multiple processors are involved.

Average Memory Access Time (AMAT)

Average memory access time is the average time to access memory considering both hits and misses and the frequency of different accesses. AMAT captures the fact that the time to access data for both hits and misses affects performance. This information helps computer architects to examine alternative cache designs.

AMAT = Hit times + Miss rate × Miss penalty

Computer memory technologies

The memory hierarchy within a computer system balances speed, size, and cost using four primary memory technologies: DRAM, SRAM, Flash, and Magnetic Disk.

DRAM (Dynamic Random Access Memory): Forms the main memory foundation, storing data as electrical charges in capacitors that need constant refreshing. It offers a balance between cost-effectiveness and capacity but is slower compared to other memory types.

SRAM (Static Random Access Memory): Used in registers and cache, it has faster access times due to its transistor-based circuits that hold data without constant refreshing. However, its complex circuitry makes it more expensive and limits its capacity.

Flash memory: Is a non-volatile alternative to traditional hard drives, using floating-gate transistors to store data even when the power is off. It is ideal for storing personal files, applications, and operating systems but is slower than DRAM and SRAM.

Magnetic disk: This technology is used in hard disk drives (HDDs), offers the most cost-effective way to store large amounts of data permanently. Despite its vast storage capacities, it has slower access times due to the mechanical nature of data storage.

Exercise 1.8

1. Find the Average Memory Access Time (AMAT) for a processor with a cache hit rate of 95%, a hit time of 4ns, and a miss penalty of 80 ns
2. Given a computer system with varying memory requirements, explain how each level of the memory hierarchy (cache, main memory, and secondary storage) optimises system performance and cost-effectiveness.
3. Discuss how the characteristics of DRAM, SRAM, Flash Memory, and

FOR CLASSROOM READING ONLY

Magnetic disk make them suitable for their respective levels in the memory hierarchy. Use examples to illustrate how each technology's speed, cost, and storage capacity contribute to the overall efficiency of the system

4. Flash memory, characterised by non-volatility, portability and reasonable speed has revolutionised the way we store and access data in the 21st century. Research on applications of Flash memory.

Cache memory

Cache memory is a smaller, faster memory that stores copies of data from the most frequently used main memory locations. Within a computer, cache memory can be designed as either split or unified.

(a) **Split cache**: This design features separate caches for data and instructions. Both caches can work simultaneously, improving bandwidth by fetching data and instructions at the same time. However, it can be less efficient as it rigidly divides the available space.

(b) **Unified cache**: In this approach, data and instructions reside in the same cache. This offers more flexibility as the entire cache space can be used for either data or instructions, potentially leading to a higher hit rate. However, unified caches can limit bandwidth as data and instructions compete for access.

For cache memory, four primary questions must be answered regarding block placement, block identification, block replacement, and write strategy:

(i) *Cache identification:* How do you identify if the information requested by the CPU is available in the cache?

(ii) ***Cache placement:*** When a block is transferred from a lower memory level, where do you place the block in the cache?

(iii) ***Cache replacement:*** When the cache is full, which block will be replaced to make room for an incoming block?

Similar to the cache memory concepts above, imagine you are a student with a small locker (your cache) close to your classroom. This locker holds the books you use most often for quick access during classes, but has limited capacity, just like a cache.

(a) *Finding a spot (Cache placement):* When you get a new book from the library, you must decide where to put it in your locker. Do you put it in the first empty slot, or do you consider which books you will need next and put it closer to the front? Cache placement strategies work similarly, deciding where to put new information in the cache is based on future needs.

(b) ***Checking your locker (Cache identification):*** While studying, if

FOR ONLINE READING ONLY

you need a book to access some information, you check your locker first before rushing to the library. Similarly, the cache needs to identify if the information the processor needs is already inside. This quick check saves valuable time compared to accessing the main memory for everything.

(c) *Making Space (Cache replacement):* If you need a book that is not in your locker, you go to the library to get it. Once you find it, you realise that your locker is full. You need to decide which book to take out to make space for the new one. Cache replacement algorithms face the same dilemma. They decide which existing information in the cache to discard to make room for new data the processor needs.

Types of cache memory

Cache memory is categorized into levels based on its proximity and accessibility to the processor, as shown in Figure 1.40. There are three general cache levels: Level 1 (L1), Level 2 (L2), and Level 3 (L3).

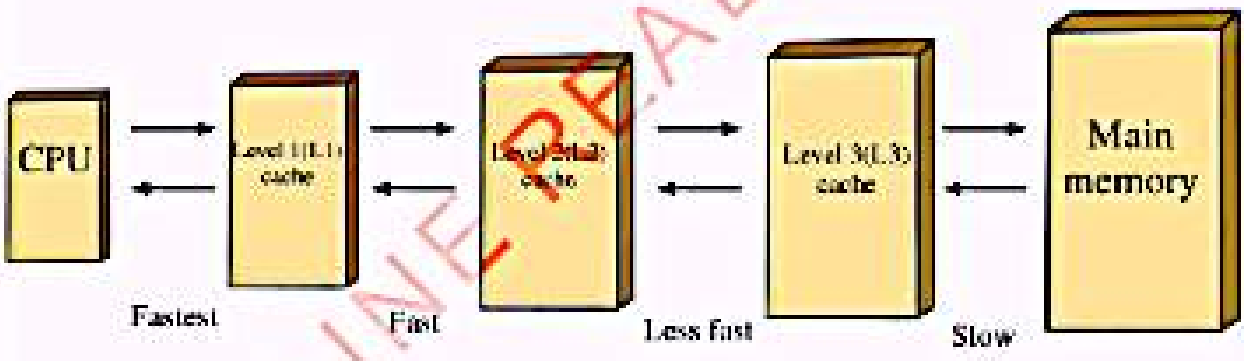


Figure 1.40: *Cache memory levels*

Level 1 cache (L1): The primary cache closest to the CPU, offering the fastest access time. It is small and stores only the most frequently used data and instructions for minimal delay.

Level 2 cache (L2): The secondary cache with larger capacity than L1. It can be on the CPU chip or a separate chip connected via a high-speed bus. It stores a broader range of data and instructions for the processor.

Level 3 cache (L3): This cache further enhances performance with a larger capacity and speed double that of main memory (DRAM). It supplements L1 and L2 caches.

Recently, trends have shifted towards integrating all three levels of cache memory onto the CPU itself, providing faster access times and a more streamlined system design.

FOR ONLINE READING ONLY

Exercise 1.9

1. Explain how cache memory improves the performance of a computer system. Provide an example to illustrate its role in speeding up data access.
2. Describe the difference between cache memory and main memory (RAM). Why is cache memory faster but smaller compared to RAM?
3. How does the principle of locality of reference influence the effectiveness of cache memory? Provide examples of spatial and temporal locality.
4. Discuss the concept of cache hit and cache miss. Why are cache hits desirable, and how do cache misses impact system performance? Provide real-world analogies to explain these concepts

Memory mapping

Memory mapping refers to a technique used to bring data from lower level of memory to a higher level of memory. As seen in Figure 1.41, cache memory mapping is the technique used to bring data from main memory into the cache memory, determining where in the cache the data should be placed. On the other hand, virtual memory mapping is the technique used to bring data from secondary storage to the main memory.

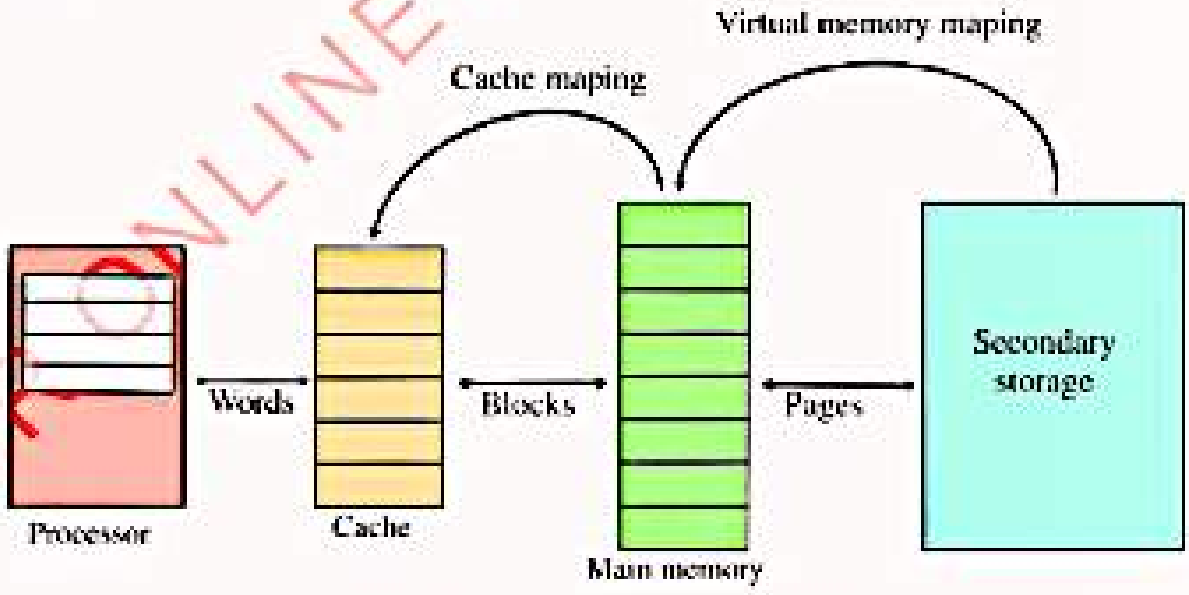


Figure 1.41: *Computer memory mapping*

Cache memory mapping

In cache memory mapping, a block from main memory is directly copied to the cache, facilitating faster data access. This process uses three techniques: direct mapping, set associative mapping, and fully associative mapping.

FOR ONLINE READING ONLY

(a) Direct mapping

Each block of main memory maps to a single, specific cache line. The formula used is:

Cache line number = (Address of the main memory block)modulo (Total number of lines in cache)

For example: For a cache with 8 lines and a main memory block address of 29, the calculation is 29 % 8 = 5. Thus, block 29 maps to cache line 5.

Consider a cache memory in Figure 1.42 with direct mapping. The cache memory is divided into 'n' lines. Then, block 'm' from the main memory can map to line number (m mod n) in the cache.

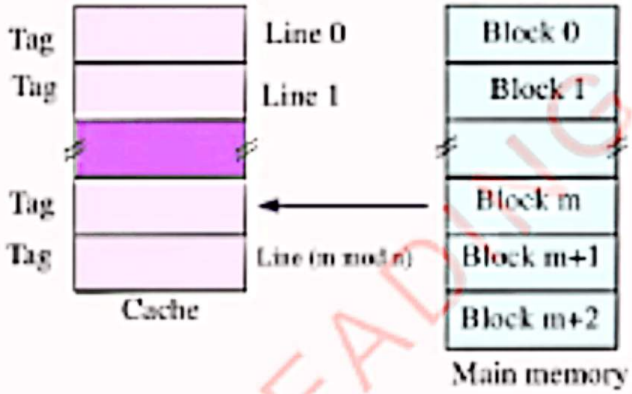


Figure 1.42: *Direct mapping*

In direct mapping, the cache address is divided into three parts: tag, line number, and block offset. As shown in Figure 1.43, the tag uniquely identifies the cache line where the data block resides, indicating if the desired data is in the cache or needs fetching from main memory. The line number locates the cache line, and the block offset pinpoints the exact data location within the block. This process is similar to main memory addressing.

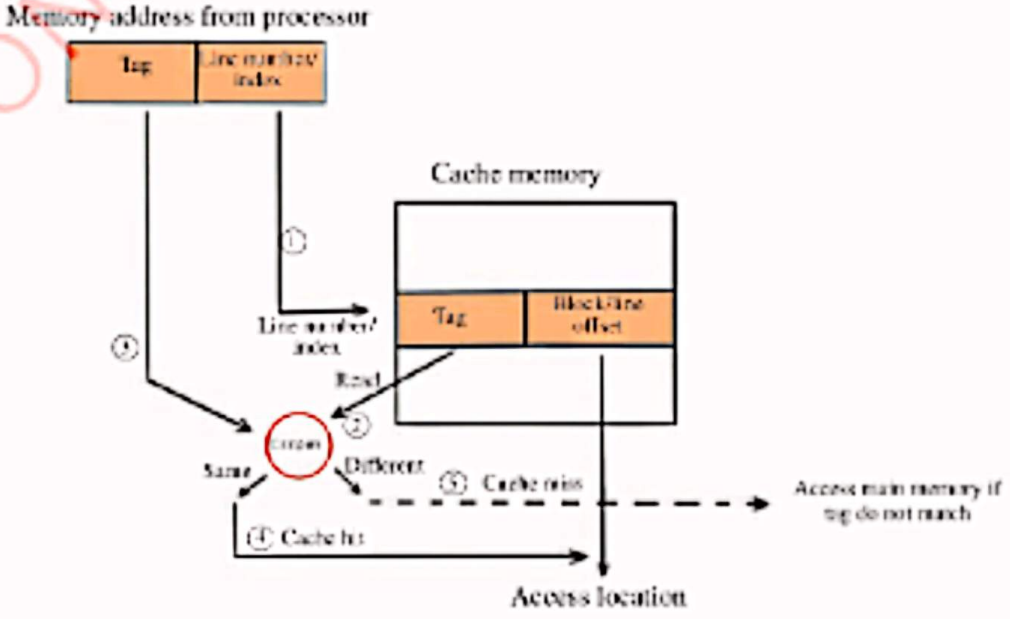


Figure 1.43: *Direct mapping process*

FOR ONLINE READING ONLY

(b) Fully associative mapping

Any block of main memory can be loaded into any line of the cache. If all lines are occupied, a replacement algorithm like Least Recently Used (LRU) or First Come, First Serve (FCFS) are used. The physical address is compared with all stored addresses in the cache. The physical address is divided as shown in Figure 1.44. When data is requested, the incoming memory address is compared with all stored addresses in the cache. If a match is found, data is read; otherwise, it is fetched from the main memory.

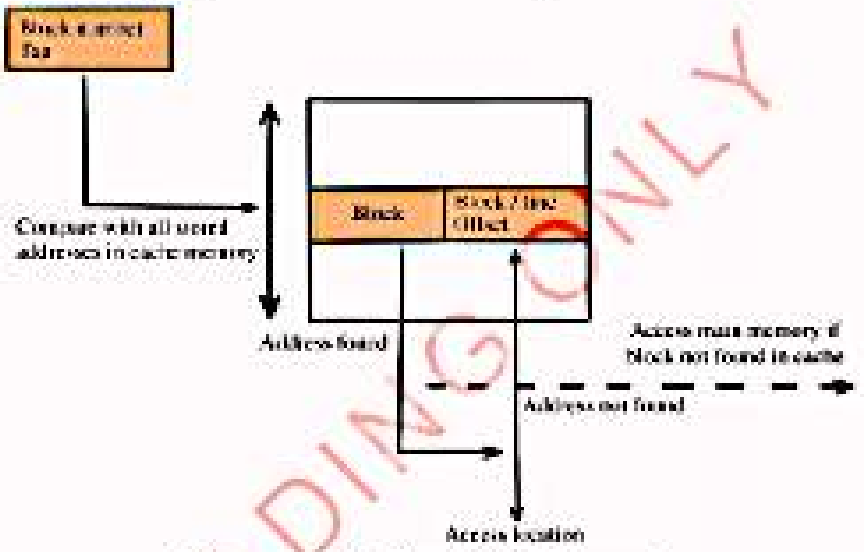


Figure 1.44: *Fully associative mapping*

(c) Set-associative mapping

The cache is divided into sets, each containing a fixed number of lines (k-way set-associative mapping). Each main memory block maps to a specific set, and within that set, it can occupy any line.

For, Example: In a 3-way set-associative cache with 12 lines, there are 4 sets (12/3). A main memory block's set number is determined by:

Cache set number = (Block Address of the main memory) Modulo (Number of sets in cache)

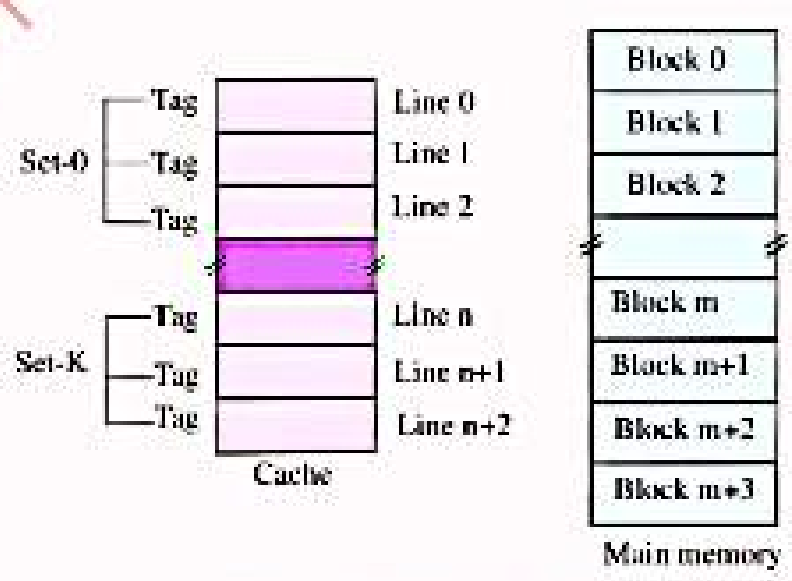


Figure 1.45: *3-way set associative*

FOR ONLINE READING ONLY

The block 'm' from the main memory can map only to set the number (m mod 3) of the cache. Within this set, block 'm' can map to any available cache line. If all cache lines are occupied, one of the existing blocks must be replaced.

In the case of fully k-way set mapping, the division of the physical address occurs as depicted in Figure 1.46.



Figure 1.46: *Physical address of a k-way set associative mapping*

Example 1.12

The modern intel 32-bit intel x86 Core i7 processor that is found in many computers has an 8-way set associative L1 data cache of size 32KB with 64-byte cache blocks. Calculate the number of bits in the tag, set, and offset fields of a main memory address.

(a) Number of bits in the physical address is given = 32 as illustrated in Figure 1.47.

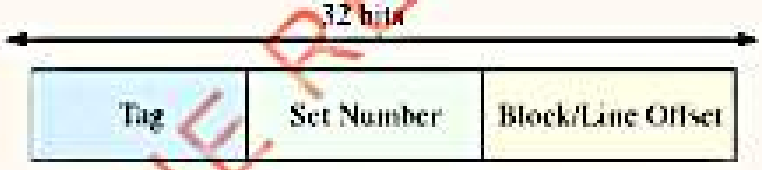


Figure 1.47: *Number of bits in address*

(b) Number of bits in block/line offset

Cache block size given = 64 bytes = 2^6

Number of bits in block offset = 6

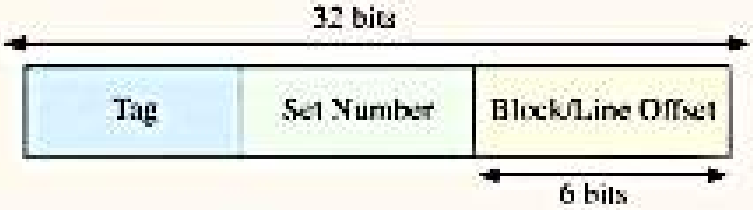


Figure 1.48: *Number of bits in block/line*

(c) Number of lines/blocks in the cache

$$Cache\ lines/blocks = \frac{Cache\ size}{Block\ size} = \frac{32 \times 2^{10}}{64} = 2^9\ blocks/line$$

(d) Number of sets in the cache

FOR ONLINE READING ONLY

Given the number of blocks in a set = the associativity factor = 8

$$Number\ of\ sets = \frac{Cache\ lines/blocks}{Number\ of\ lines/blocks\ in\ a\ set} = \frac{2^9}{8} = 2^6\ sets$$

Number of bits in set number = 6

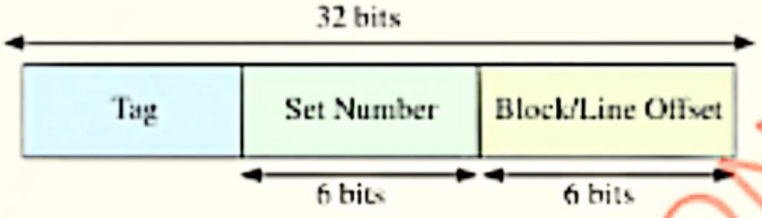


Figure 1.49: *Number of sets in cache*

(e) Number of bits in tag

Number of bits in tag = Number of bits in physical address – (Number of bits in set number + Number of bits in block offset)

Number of bits in tag = 32 bits – (6 bits + 6 bits) = 20

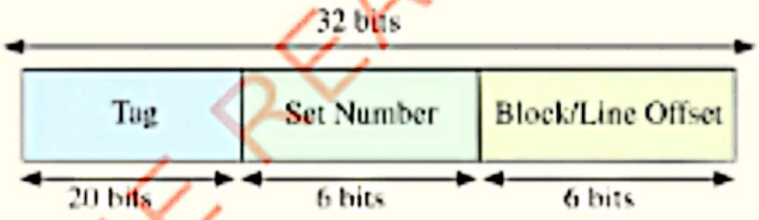


Figure 1.50: *Number of bits in the tag*

Activity 1.12:

Exploring cache mapping techniques

Aim: To understand and apply different cache mapping techniques to optimise data retrieval in computer systems

Materials: Computer, Internet, simulation tool(Such as Logisim), calculator or spreadsheet

Instructions:

Explore different cache mapping techniques by calculating tag, set, and block (or line) offsets. These calculations help locate specific data within the cache by dividing the cache address into three parts: tag, set, and block/line offset.

The Challenge: Work with three common cache mapping schemes-Direct Mapped, k-way Set Associative, and Fully Associative. For each scenario in Table 1.6, you are given:

FOR ONLINE READING ONLY

(a) **Main memory size (M)**: Total number of bytes in main memory.

(b) **Cache size (C)**: Total storage capacity of the cache.

(c) **Block size (B)**: Size of each data block the cache can hold.

(d) **Cache address (A)**: The address to locate within the cache.

Task 1: *calculate the tag, set, and block/line offset for each mapping scheme.*

Scenario	M (Bytes	C (Bytes)	B (Bytes)	A(Bytes)
Direct Mapped Cache	1024	128	16	232
2-way Set Associative Cache	4096	256	32	768
Fully Associative Cache	2048	512	64	1536

Steps to Calculate:

Step1: Calculate direct mapped cache: *Cache line number , Block/line offset, and Tag.*

Step 2: Calculate k-way set associative cache: *Cache set number, Block/line offset, and Tag.*

Step 3: Calculate fully associative cache: *Cache set number, Block/line offset, and Tag*

Task 2: *Explore the advantages and disadvantages of each scheme.*

Deliverables:

(a) Cache set number, Block/line offset and tag for each scheme

(b) Advantages and and disadvantages of each scheme. Document your findings in a potifolio.

FOR ONLINE READING ONLY

Exercise 1.10

1. Describe the LRU replacement algorithm
2. What would the k-way set associative mapping become, if k = 1?
3. How many sets does a fully set associative mapping have?

Primary and secondary memory

Primary memory, also known as main memory, includes RAM and cache memory, which provide fast and temporary storage directly accessible by the CPU for active processes and data. Secondary memory, such as hard drives and SSDs, offers larger, permanent storage for data and programs that are not in immediate use, with slower access speeds compared to primary memory.

Activity 1.13:

Exploring primary and secondary computer memory

Aim:
To explore the differences between primary and secondary memory, understand their functions, and learn about the role of flash memory in modern computing devices

Materials:
Computers, Internet, library or online resources for research

Instructions:

Step 1: Explore primary memory (RAM):

(a) Watch a Video explaining primary memory (RAM). Think how it acts as the computer's main working memory and its volatile nature.

(b) Reflect: Think of the library analogy in previous sections where primary memory is like library shelves with limited but fast access to books.

Step 2: Explore secondary memory: Using one type of secondary storage device (HDD, SSD, Optical Discs), research on their assigned storage type, focusing on capacity, speed, cost, and typical use cases. Write down:

(a) Details on how the device works, its advantages and disadvantages, and its typical applications.

(b) Comparison of their device to other types of secondary storage. Include visual aids, diagrams, or videos to enhance your understanding of your findings.

Step 3: Interactive diagram activity:

Using presentation software, create an interactive diagram showing how primary and secondary memories interact in a computer system.

(a) Show how data moves from secondary storage to primary memory and back.

(b) Explain the role of the CPU in this process.

Step 4: Research and present a case study on the rise of flash memory in mobile devices.

(a) Discuss its advantages, such as solid-state design, lower power consumption, and faster access times.

(b) Explain why flash memory is preferred in smartphones and tablets.

Step 5: Explore a device:

Explore a personal device (such as smartphone, tablet, or laptop) of a parent, school, or any provided by your teacher.

(a) Identify the type of storage used in their device (such as, flash memory in smartphones).

(b) Discuss why this type of memory is suitable for their device.

FOR ONLINE READING ONLY

Deliverable:

Write a one-page report summarizing what you learned about primary and secondary memory, and the importance of flash memory in modern devices. Reflect on how understanding these concepts help in better appreciating the technology they use daily. Document your findings in a portfolio.

Virtual Memory

Virtual memory expands a computer's usable memory by using both primary (RAM) and secondary (HDD/SSD) storage. The operating system creates a larger virtual memory space divided into pages, loading only active pages into RAM while keeping less-used pages in secondary storage.).

Translation Lookaside Buffer (TLB)

Managing virtual memory involves translating virtual addresses used by programs into physical addresses in RAM. To speed up this translation process, the CPU's Memory Management Unit (MMU) utilizes a cache called the Translation Lookaside Buffer (TLB).

The TLB stores recently used virtual-to-physical address translations. When the CPU needs to access data in memory, it checks the TLB first. If the translation is found (TLB hit), the physical address is retrieved quickly. If the translation is not found (TLB miss), the MMU consults the page table in the primary memory or secondary storage, which is slower. By effectively utilising virtual memory and the TLB, the computer can work with a larger memory space while maintaining fast access times for frequently used data.

Exercise 1.11

1. Imagine you are a librarian managing a library with two sections: a reference section with readily available books on shelves and a vast archive room for storing older materials. How is this analogy similar to the concept of primary and secondary memory in a computer? Explain the key differences between primary and secondary memory in terms of speed and capacity.
2. You are working on a critical assignment and need to access a document you saved earlier. However, when you try to open it, you receive an error message. What factors could be preventing you from accessing the document? Explain the role of primary and secondary memory in potentially causing this issue.
3. Imagine you are working on a complex group project. To ensure everyone has access to the latest information, you might utilize a shared online document and a local copy on your device. Explain the concept of virtual memory and the TLB in a way that relates to this collaborative scenario. How do they work together to optimise memory usage?

FOR ONLINE READING ONLY

Instruction set architecture and design

Concept of Instruction Set Architecture



Scenario 1.4: *Cooking instructions in the kitchen*

Imagine a kitchen where a chef prepares various dishes. Each tool, ingredient, and cooking step is organised for efficiency, similar to Instruction Set Architecture (ISA) in computer architecture. In this kitchen, the chef follows recipes with detailed instructions, like chopping vegetables and marinating meat. Similarly, ISA defines the instructions a processor can execute, such as arithmetic operations and control flow.

Just as the chef has specific tools like knives, pots, and pans for different tasks, a processor has registers to hold data temporarily during execution. The instruction execution cycle mirrors the chef's step-by-step process: instructions are fetched, decoded, and executed, then results are stored back in memory. Addressing modes in ISA (direct, indirect, and register addressing) are like different cooking techniques (frying, roasting, or boiling) that determine how instructions access data in memory.

Consider a simplified communication system in the kitchen, where short codes are used to request or transfer ingredients. This resembles Register Transfer Language (RTL), representing data movement within the processor. ARM and x86 architectures, like different restaurant chains, have distinct approaches: ARM focuses on efficiency, while x86 offers a broader range of tools.

In summary, just as an organised kitchen is essential for a chef, understanding ISA is crucial for programmers and engineers to design, optimise, and develop software that runs smoothly on various computer architectures.



Questions:

(a) How does the concept of addressing modes in Instruction Set Architecture relate to different cooking techniques in a kitchen?

(b) In what ways are the roles of registers in a processor similar to the tools used by a chef in a kitchen?

FOR ONLINE READING ONLY

Can you explain the significance of understanding Instruction Set Architecture for programmers and computer engineers in developing software for different computer architectures?

Examples of Instruction Set Architectures (ISA)

(a) **Intel x86:** This is a widely used ISA, particularly in personal computers and servers. It supports both CISC and RISC-like instructions, making it versatile for various computing tasks. Despite its complexity, it remains dominant in the PC market and is continuously evolving to meet modern computational demands. Common examples of Intel x86 power processors are Intel Celeron, Intel Pentium, Intel core family of processors (i3, i5, i7).

(b) **ARMv8 (AArch64):** The successor to ARMv7, offering both 32-bit and 64-bit instruction sets for improved performance and legacy application support. It is increasingly used on high-end mobile devices and servers. ARMv8 powers processors such as; Amazon Graviton2 (server for cloud computing), Samsung Exynos 990/2100, Apple A13/ A14/A15 Bionic and Qualcomm Snapdragon 855/865/888 are all used in smartphones.

(c) **RISC-V:** A relatively new, open-source RISC architecture gaining traction due to its flexibility and customizability. It is well-suited for emerging applications like the Internet of Things (IoT).RISC-V powers processors include NVIDIA Grace CPU and SiFive Core IP

(d) **Custom Instruction Set Architectures (ISAs):** Many companies and research institutions are developing custom ISAs tailored for specific applications like machine learning or network processing. These often combine elements of RISC and CISC philosophies for optimal performance within their niche.

Types of instructions

In most ISAs, you will find the following common types of instructions:

(a) Arithmetic Logic Unit (ALU) instructions

These instructions perform basic mathematical and logical operations on data using the ALU (Arithmetic Logic Unit) within the CPU. Examples include:

(i) *Arithmetic:* add, sub, mul, div (addition, subtraction, multiplication, division)

(ii) *Logical:* and, or, xor, not (bitwise AND, OR, XOR, NOT)

(iii) *Shift:* shl, shr (shift left, shift right)

(iv) *Compare:* cmp, eq, gt, lt (compare, equal to, greater than, less than)

(b) Data transfer instructions

These instructions move data between various locations within the computer system. Examples include:

(i) *Load:* lw (load word), lb (load byte) (transfer data from memory to registers)

FOR ONLINE READING ONLY

(ii) *Store:* sw (store word), sh (store byte) (transfer data from registers to memory)

(iii) *Move:* mov (move data between registers)

(iv) *Input/Output (I/O):* in, out (transfer data between CPU and external devices)

(c) Control flow instructions

These instructions control the flow of program execution by allowing for conditional branching and jumping to different parts of the code. Examples include:

(i) *Branch:* if, else, goto (conditional execution based on register values or conditions)

(ii) *Jump:* jmp (unconditional jump to a specific instruction address)

(iii) *Call:* call (call a subroutine or function)

(iv) *Return:* ret (return from a subroutine or function)

(d) System control instructions

These instructions interact with the operating system or manage system resources. They are typically privileged instructions accessible only by the operating system or specific programs.

Examples include:

(i) *Interrupt handling:* Instructions to handle interrupts from devices or software.

(ii) *Memory management:* Instructions for managing virtual memory or memory protection.

(iii) *Processor control:* Instructions to change processor modes or power management.

(e) Special instructions

Some ISAs might include specialised instructions for specific purposes, such as:

(i) *Floating-point operations:* Instructions for performing calculations on floating-point numbers (numbers with decimal points).

(ii) *String manipulation:* Instructions for working with strings of characters efficiently.

(iii) *Vector processing:* Instructions for performing operations on multiple data elements simultaneously.

Exercise 1.12

1. The passage describes two main approaches to instruction set design: RISC and CISC. Create a table comparing these two approaches. Include details like design philosophy, instruction complexity, efficiency considerations, and common applications.
2. Research a specific example of a modern CPU (Central Processing Unit). Identify the ISA it uses (e.g., Intel x86, ARMv8) and explain why this particular ISA might be chosen for that specific CPU.

FOR ONLINE READING ONLY

Registers

In the heart of a computer processor lies a critical set of high-speed memory locations called registers. Registers act as the CPU's workbench, temporarily storing data that is actively involved in calculations or manipulations. Their close proximity to the processor allows for significantly faster access compared to fetching data from cache and main memory.

Classification of registers:

There are two main classifications of registers within an ISA, each with distinct purposes:

(a) General Purpose Registers (GPRs): These are versatile storage locations that can be used by any instruction in the CPU. They can store a variety of data types like integers, floating-point numbers, or memory addresses during program execution.

(b) Special Purpose Registers (SPRs): These registers are dedicated to specific control functions for the CPU itself. They hold vital information for smooth operation. Examples are;

(i) *Program Counter (PC):* This register keeps track of the next instruction to be executed, ensuring that program flows seamlessly.

(ii) *Stack Pointer (SP):* This register plays an important role in managing function calls and local variables within a program.

(iii) *Status Registers (SR):* These registers hold information about the outcome of recent operations, such as carry flags or zero flags, used for conditional branching and error handling.

Function and operations of CPU registers

(a) *Data storage and manipulation*: Registers act as a temporary holding ground for data being processed. Instructions in the ISA typically reference specific registers for operands (data involved in the operation) and destinations for the results.

(b) *Improving performance:* By keeping frequently accessed data readily available, registers significantly reduce the need to fetch data from slower main memory, leading to faster execution speeds.

(c) *Facilitating function calls:* When a program calls a function, arguments are often passed through registers, enabling efficient communication between different parts of the code.

(d) *Conditional branching:* Status flags set in special purpose registers are used for conditional branching in programs.

Visualizing registers

As depicted in Figure 1.52, a processor houses multiple register files such as GPRs, and SPRs. Data paths, illustrated in the form of arrows, show how data flows between registers, and the Arithmetic Logic Unit (ALU) for performing calculations, and the main

FOR ONLINE READING ONLY

memory. The Control Unit interacts with registers to fetch instructions, decode them, and direct the flow of data to the appropriate registers and the ALU.

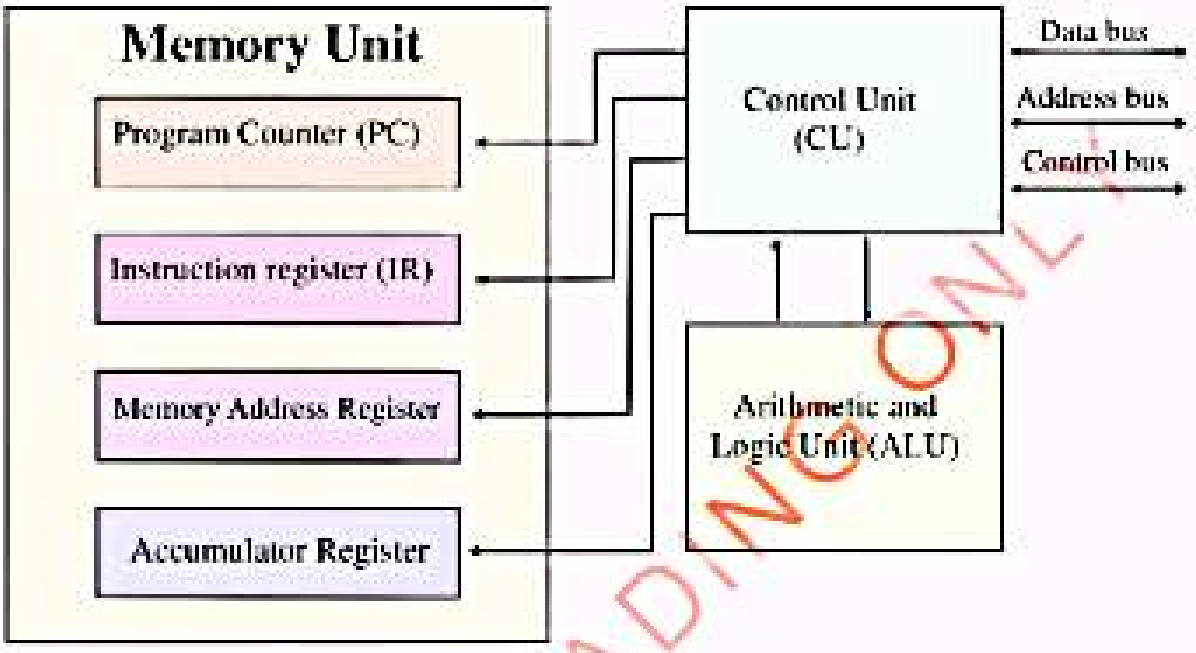


Figure 1.52: *CPU Registers*

Similarities between registers and cache:

Both registers and cache aim to improve performance by providing faster access to frequently used data. However, they have distinct characteristics:

(a) ***Size and location:*** Registers are much smaller in size and reside directly within the CPU for the fastest access. Cache, on the other hand, is larger and sits between the CPU and main memory, acting as a buffer.

(b) ***Data types:*** Registers can typically store various data types, while cache may be optimised for specific data types depending on the architecture.

Examples of register operations

1. **RISC-V (Reduced Instruction Set Computing):**
   (a) Instruction: add a0, a1, a2
   (b) This instruction adds the contents of a1 and a2 and stores the sum in a0. The use of dedicated registers allows for efficient data manipulation within the CPU.
      (i) add: This specifies the operation (addition).
      (ii) a0: This is the destination register where the result is stored.
      (iii) a1 and a2: These are the source registers containing the values to be added.

FOR ONLINE READING ONLY

2. **ARMv8 (AArch64):**
   (a) Instruction: ldr x0, [x1]
   (b) This instruction retrieves the data from the memory location whose address is stored in x1 and loads it in register x0.
      (i) ldr: This specifies the operation (load).
      (ii) x0: This is the destination register where the loaded data will be stored.
      (iii) [x1]: This indicates the memory location to be loaded. The value in register x1 acts as the base address for memory access.

3. **x86 (Complex Instruction Set Computing):**
   (a) Instruction: mov eax, [ebx + 4]
   (b) Explanation: This instruction combines memory addressing and data. It fetches the value from the memory location calculated by adding 4 to the address stored in register ebx and places it in register eax.
      (i) mov: This specifies the operation (move).
      (ii) eax: This is the destination register.
      (iii) [ebx + 4]: This indicates the memory location to be accessed.

Activity 1.14:

Simulation to explore how computer memory is utilised using CPU simulator

Aim:
To enhance individual knowledge and comprehension of computer memory utilization in a practical and interactive manner

Materials:
Computer, CPU simulator software or Internet.

Instructions:
Create a program and instructions to add two numbers. The first number should be stored in a register R00, the second number should be stored in register R01, and the results should be stored in register R02. All instructions should appear in the CPU instructions memory (RAM).

Step 1: Understand the components of the CPU simulator

FOR ONLINE READING ONLY

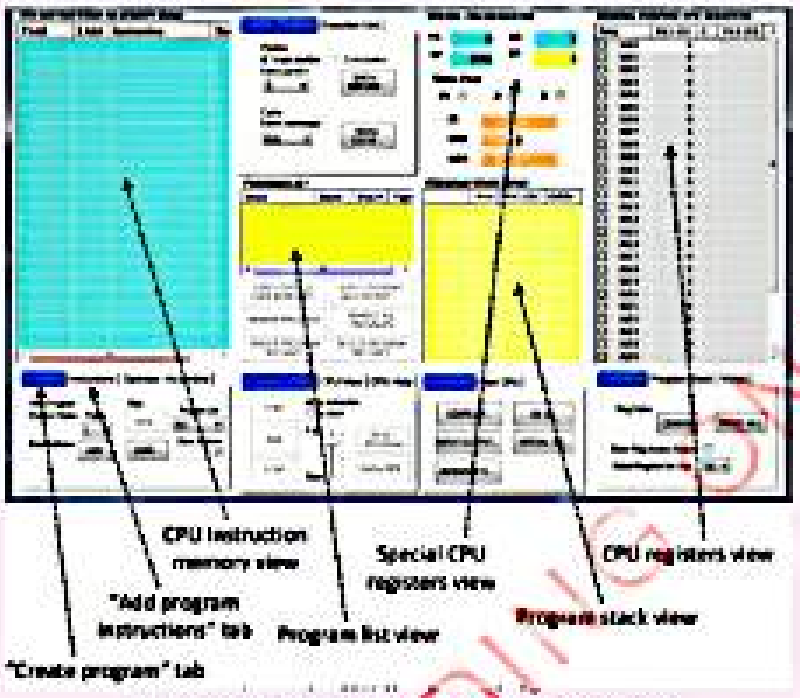


Figure 1.53: *Components of CPU simulator*

Step 2: Create a program

To create a new program, click "program" button, enter the name in the "Program Name" box (for this example enter test1). Also enter the "base address" in the "Base Address box" (You can enter any number, for this example enter **100**) then click on the "ADD button". The new program's name will appear in the "Program List view"

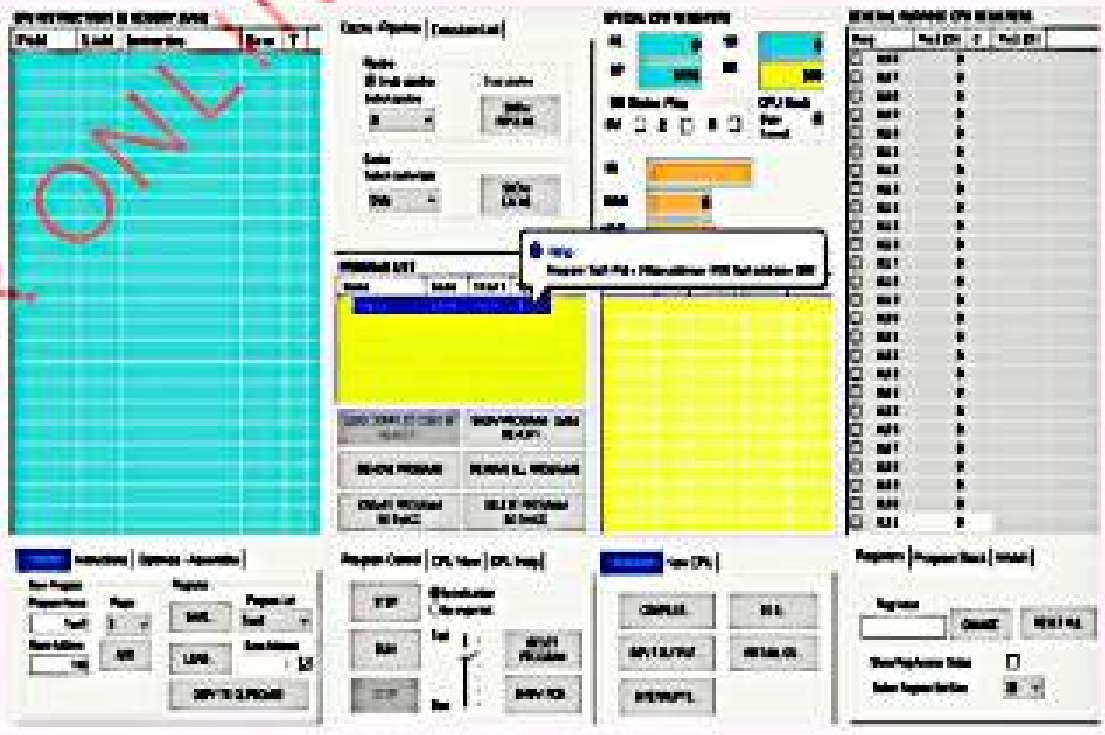


Figure 1.54: *Creating a program*

FOR ONLINE READING ONLY

Step 3: Add program instructions

(a) Select "instructions" from the CPU simulator interface. Click "ADD NEW..." button to add a new instruction, and use "EDIT..." button to edit the selected instruction.

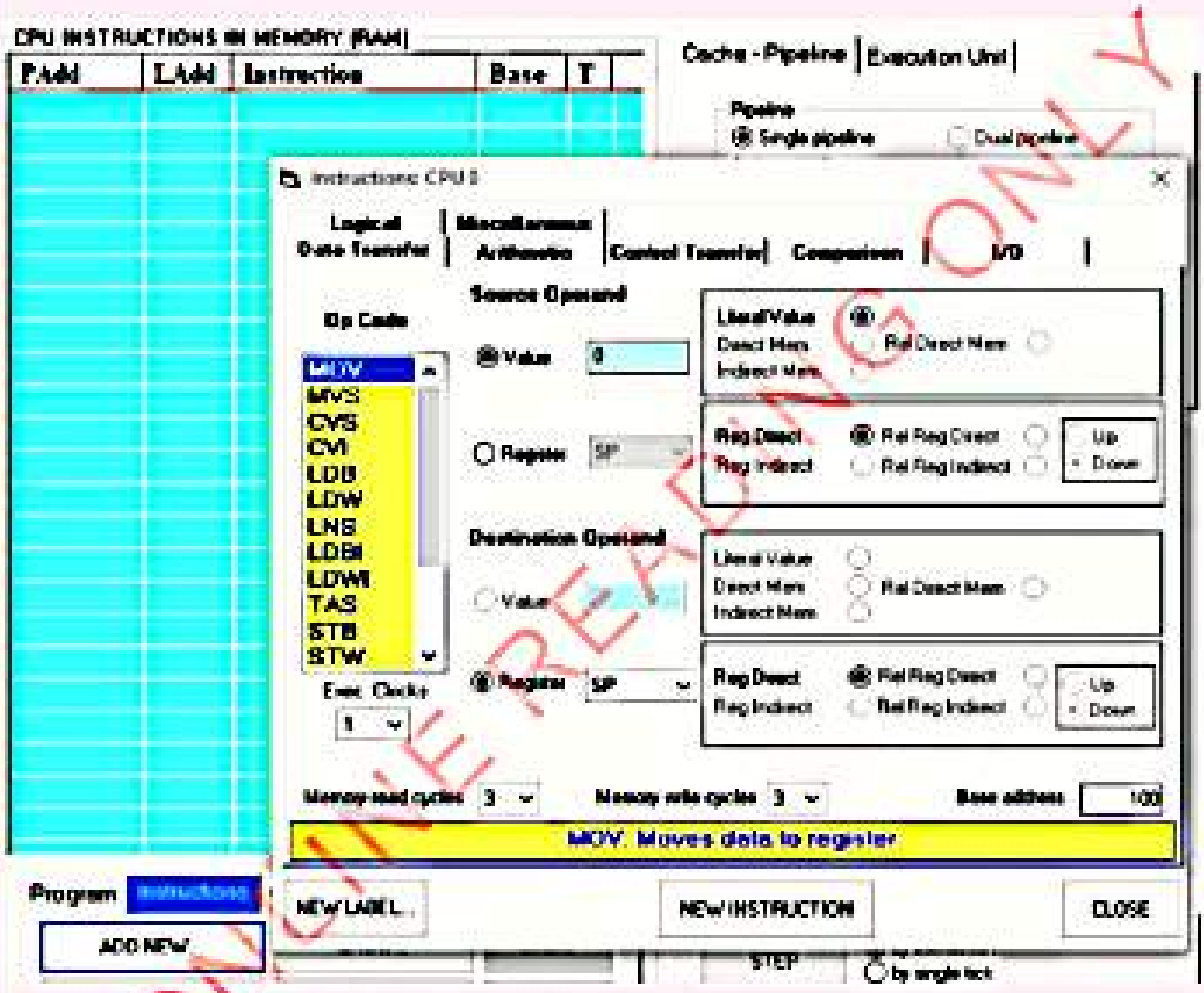


Figure 1.55: *Adding a program instruction*

(b) Create an "instruction", which moves the "first number" (in this example use 15) to register R(0. To do this, click the "ADD NEW...button" in the instruction window and select "Data Transfer", then select "MOV function", add first number, that is 15, in the "source operand" value box and select "R0(" from the drop down menu of the register in the "Destination Operand". Click "NEW INSTRUCTION" for the instruction created to appear in the "CPU INSTRUCTIONS IN MEMORY(RAM)".

FOR ONLINE READING ONLY

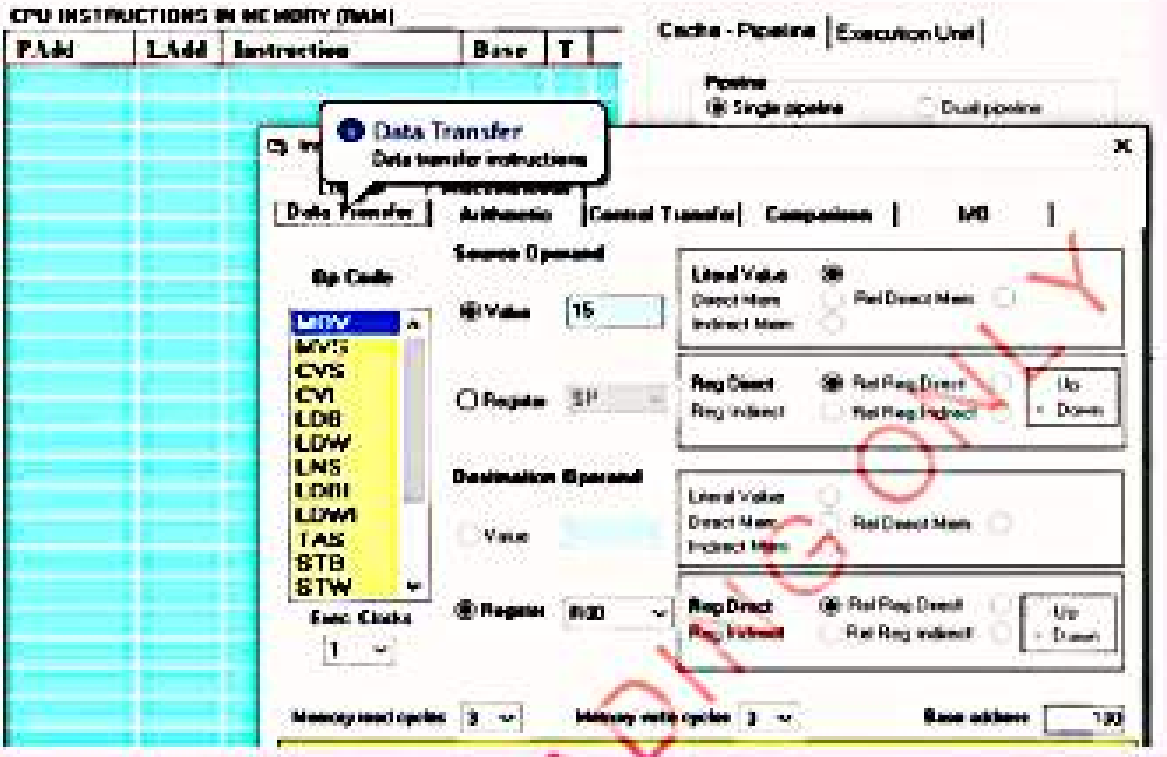


Figure 1.56: *Creating instruction*

(c) Execute the instruction in (b) by "double-clicking" on the instruction in the "CPU INSTRUCTIONS IN MEMORY(RAM)". Observe the result in the "CPU Registers" view.

(d) Create an "instruction", that moves the "second number" (in this example use "20") to register R01. Repeat the procedures in (b), and add the second number, that is 20 in the "source operand" value box and select "R01" from the drop down menu of the register in the "Destination Operand".

(e) Execute the instruction in (d) by "double-clicking" on the instruction in the "CPU INSTRUCTIONS IN MEMORY (RAM)". Observe the contents of "R00" and "R01" in the CPU Registers view (see the figure below).

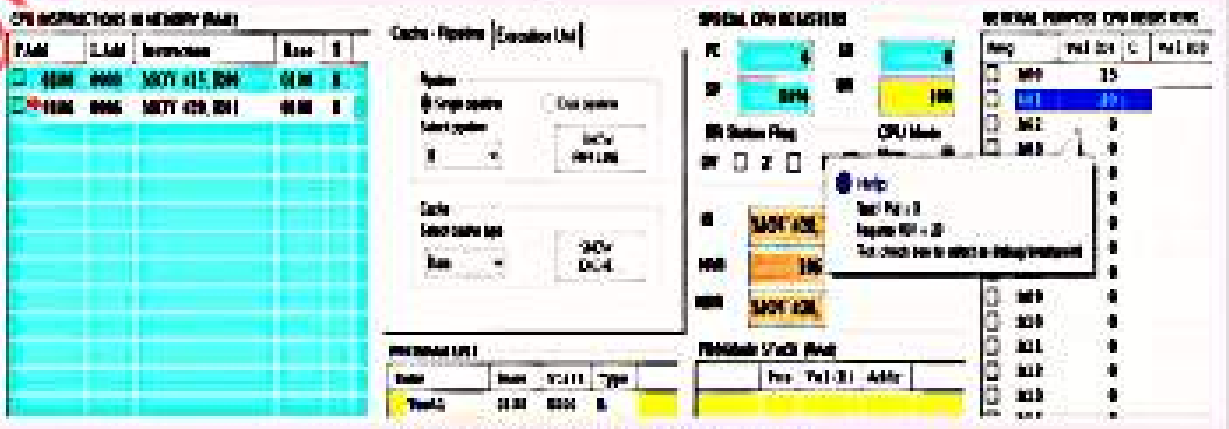


Figure 1.57: *Instruction execution*

FOR ONLINE READING ONLY

(f) Create an instruction, which adds the contents of R00 and R01 and store the results in R02. In the instruction interface, click "Arithmetic" then select "ADD function". Select R00 in the register of the Source Operand and R02 in the register of the Destination Operand and click "NEW INSTRUCTION".

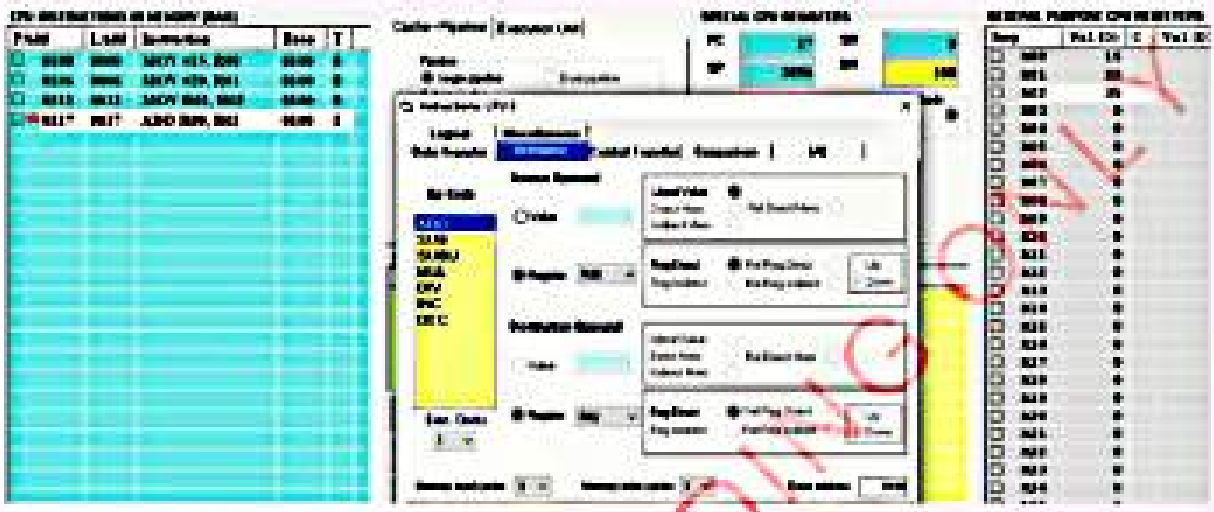


Figure 1.58: *Creating instructions for adding and storing contents*

(g) Create an instruction, that pushes the result value in the register to the "program stack(Random Access Memory)", and then "execute" it. Click the Data Transfer button and select "PSH function". Then, select R02 in the register of the Source Operand, click "NEW INSTRUCTION" and observe the value in the "Program Stack(RAM)".

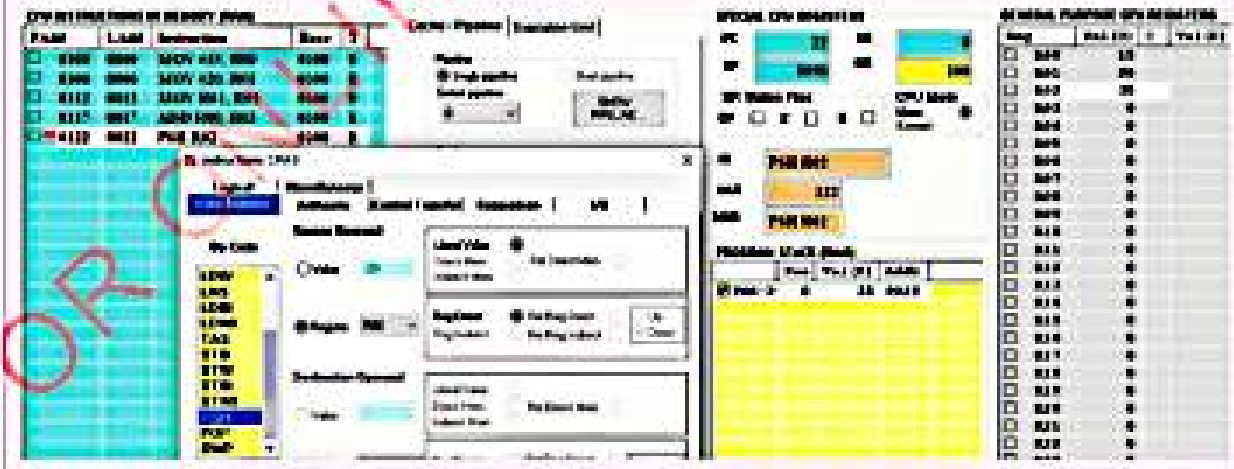


Figure 1.59: *Creating a program that pushes the result value in the register to the program stack(RAM) and then execute it*

FOR ONLINE READING ONLY

Activity 1.15:

Register and ISA interaction

Aim:
To enhance individual comprehension of the inner workings of a CPU and how registers facilitate efficient data processing and manipulation within the context of the ISA

Materials:
Computer with Internet access, Open source CPU simulator (MARS or QEMU).

Instructions:
Explore how registers interact with the Instruction Set Architecture (ISA) within a CPU. Use CPU simulator or any other tool, observe register usage during program execution so as to understand the interplay between registers and ISA instructions

Step 1: Download the MARS CPU simulator which is designed for MIPS and RISC-based ISA from https://courses.missouristate.edu/kenvollmar/mars/ (RISC-based ISAs) and install.

Step 2: Load a sample program as shown below which is a basic example written in MIPS assembly language for the MARS simulator. It demonstrates the use of General Purpose Registers (GPRs) for calculations and storing data.

```
# Define constants
ADD_VALUE  equ 10
SUB_VALUE  equ 5

# Load values into registers
li $t0, ADD_VALUE  # Load 10 into $t0
li $t1, SUB_VALUE  # Load 5 into $t1

# Perform operations and store results
add $s0, $t0, $t1  # Add values in $t0 and $t1, store in $s0
sub $s1, $t0, $t1  # Subtract values in $t0 and $t1, store in $s1

# Program terminates here (implicit in MARS)
```

Step 3: Open MARS and create a new file. Paste the provided code into the editor window.

FOR ONLINE READING ONLY

Step 4: Run the program and observe the following:

(a) The values loaded into registers $t0 and $t1.

(b) The result of the addition stored in register $s0.

(c) The result of the subtraction stored in register $s1

Questions

1. Identify which registers were used as temporary registers and which were used for storing results.
2. Explain how the li instruction (load immediate) is used to load values into registers.
3. Discuss the role of the add and sub instructions in manipulating data within registers.
4. Discuss how specific ISA instructions (add, li, load, etc) interact with registers.
5. Explain how registers contribute to the overall efficiency of program execution.

Instruction execution cycles

Datapath and control

As you may recall, the CPU is the active part of the computer that processes data and makes decisions. The CPU is made up of datapath and control. Datapath consists of the functional units of the processor. Datapath includes: elements that hold data (such as Program counter, register file, and instruction memory), elements that operate on data (such as ALU, and adders) and buses for transferring data between elements. The Control part (brain) of the processor commands the datapath regarding when and how to route and operate on data.

Instruction execution cycles

The instruction cycle represents a series of steps that a CPU goes through to execute a single machine instruction. As shown in Figure 1.60 in an instruction cycle, the processor (1) fetches an instruction from instruction memory, (2) decodes the instruction, (3) reads the addresses from memory, (4) executes the instruction, and (5) writes the result back to memory or register.

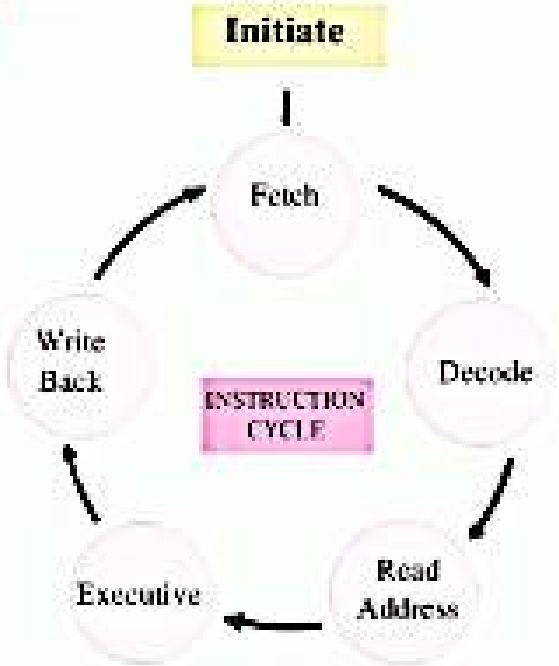


Figure 1.60: *Instruction cycle*

Figure 1.60 shows which datapath functional units are utilized at each phase. An arithmetic instruction add r1, r2, r3 (r1= r2+r3) has been used as an example to explain the concepts at each

FOR ONLINE READING ONLY

phase. Assume register r2 contains the value 5 and register r3 contains the value 6. Therefore, at the end of the cycle, register r1 should contain the value 11.

(a) Fetching of instruction

In this phase, the CPU retrieves an instruction from memory. The Program Counter (PC) register holds the address of the next instruction to be fetched. The Control Unit instructs the datapath to read the memory location pointed to by the PC and then load the fetched instruction (add r1, r2, r3) into the Instruction Register (IR). Datapath elements involved in this phase are; PC, Memory, Instruction Register (IR), and Control Unit.

(b) Decoding the instruction

The Control Unit analyses the instruction in the IR. It breaks down the instruction into its key parts. Opcode - this specifies the operation to be performed (in this case, "add"). Operands - these are the data elements involved in the operation, identified by their register names.

(i) *Source register (rs):* this operand will be used as the first input to the ALU r2 in this instruction).

(ii) *Destination register (rt):* this operand will be used as the second input to the ALU and will also store the result of the operation (r3 in this instruction).

Based on this information, the Control Unit generates control signals to activate specific datapath elements for the next phase. Datapath elements involved in this phase are; Control Unit, and Instruction Register (IR).

(c) Read Operands

This phase is necessary for instructions that require operands (data to be manipulated). Example of instructions that require operands are arithmetic and logic instructions (such as add, subtract, and multiply), and data transfer instructions (such as load, store, move). Examples of instructions that do not require operands include; jump instructions, and branch instructions. Therefore, **add r1, r2,** and **r3, requires** operands.

The Control Unit uses the register names (rs and rt) identified during decode to locate the corresponding registers within the register file. The values from these registers (r2 and r3 in this example) are loaded into temporary holding registers within the CPU for the ALU to access during execution. Datapath elements involved include; the Control Unit, Register File and data memory (if the operand is in memory).

(d) Execute

This is where the actual calculation happens. The Control Unit activates the ALU (Arithmetic Logic Unit). The ALU takes the operands from the temporary registers and performs the addition based on the opcode ("add"). The result of the addition (11) is stored in a a temporary register. Datapath elements involved in this phase are the Control Unit, ALU, and Registers file.

FOR ONLINE READING ONLY

(e) Write back

This phase involves storing the result (11) of the operation to the destination register (rt). The Control Unit directs the data from the temporary register holding the result to the register specified by rt (r1 in this example). Not all instructions require a write-back (such as, conditional branches). Datapath elements involved are the Control Unit, Register File, and Data memory (if the result is written back to memory).

Once this cycle finishes, the PC is incremented to point to the next instruction in memory. The entire fetch-decode-read-execute-write back cycle repeats to process the next instruction and continue program execution.

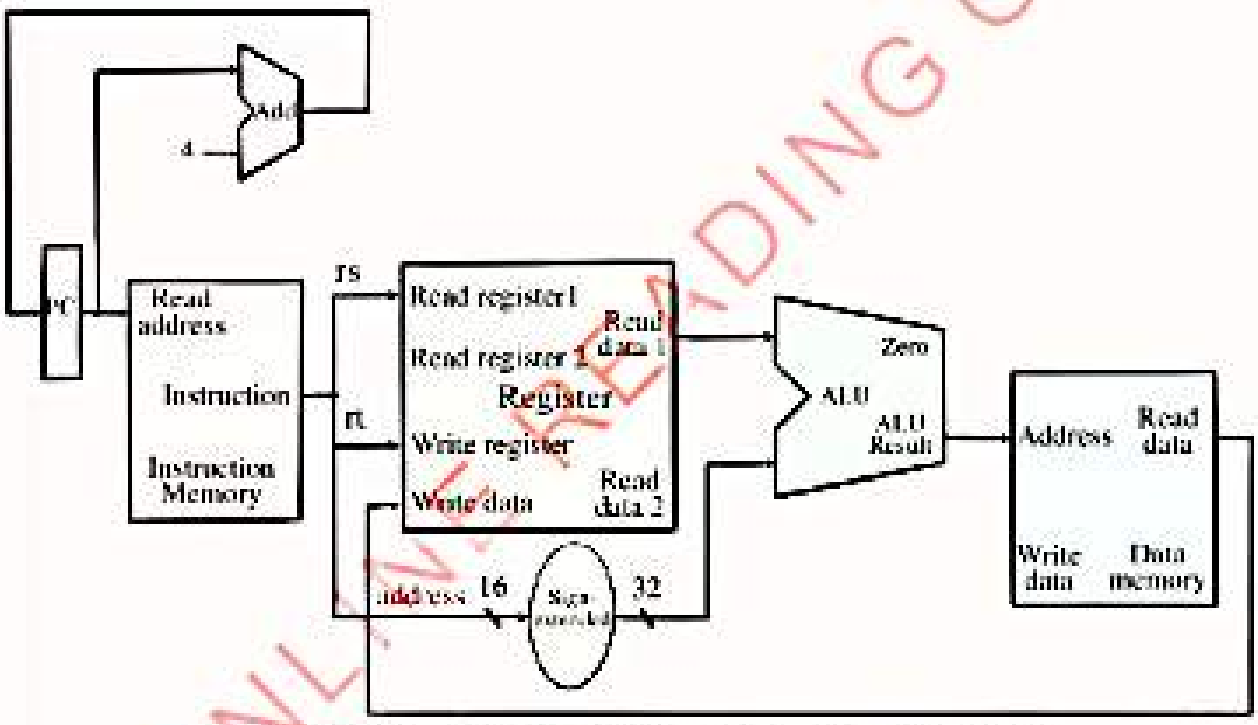


Figure 1.61: *Datapath for load arithmetic instructions in MIPS ISA.*

FOR ONLINE READING ONLY

Exercise 1.13

1. With the help of diagrams, describe the following datapath functional units: instruction memory, program counter, adder, register file, ALU and Sign extension.
2. The instruction execution cycle described in 1.4.2 is for arithmetic instructions. Using the library and online resources, write the instruction execution cycle for load/store instructions.
3. Research and explain the purpose of the sign extension element in the datapath. When would sign extension be necessary during arithmetic operations involving immediate values in MIPS instructions?

Addressing modes

In the previous section, we learned about the instruction cycle. We saw how instructions are fetched, decoded, and their operations are carried out. Addressing modes are special instructions that help the CPU find data.

Recall the kitchen-ISA analogy introduced at the beginning of this chapter. Imagine you are a chef following a recipe (the program) in a kitchen (the CPU). The recipe tells you what ingredients (data) to use and what steps (instructions) to take. But how does the recipe tell you exactly where to find those ingredients? This is where addressing modes come in – they are like special instructions within instructions, telling the CPU how to locate the data it needs to work with. Common addressing modes are; register addressing, immediate addressing, direct addressing, indirect addressing and relative addressing.

Register addressing

Data is stored in registers within the CPU, providing fast and efficient access. *Example (MIPS):* add $t1, $s2, Ss3 (Adds values in registers Ss2 and Ss3, storing the result in St1).

Immediate addressing

The instruction includes the actual data value.

Example (MIPS): addi St0, Ss1, 5 (Adds 5 to the value in register $s1, storing the result in St0).

Direct addressing

The instruction specifies a memory address where the data is stored.

Example (MIPS): lw St2, 10($s0) (Loads data from the memory location 10 bytes away from the address in $s0 into St2).

Indirect addressing

The instruction specifies a memory address that points to another memory location with the actual data.

Example (MIPS): lw St4, (Ss5) (Loads data from the memory location pointed to by the address in Ss5 into St4).

Relative addressing

The instruction specifies an offset from the current instruction's address.

Example (MIPS): beq St6, Ss7, Label (Branches to "Label" if the values in $t6 and Ss7 are equal, using an offset relative to the current instruction's address).

Exercise 1.14

Imagine you are a programmer working on a game engine for a mobile racing game. The game uses various data structures to store information about the race track, car physics, and player input. Answer the following questions, considering the most appropriate addressing mode (register, immediate, direct, indirect, or relative) for each scenario and explain your reasoning.

FOR ONLINE READING ONLY

(a) *Shujaa power activation:* The game allows players to activate "Shujaa Power," which grants a temporary speed boost. This activation is triggered by a button press. A flag value (0 for not pressed, 1 for pressed) is stored in a specific memory location. Which addressing mode would be most efficient for the instruction that checks this flag value, and why?

(b) *Car position update:* During the race, the car's position on the track is constantly updated based on various calculations. The new position data needs to be stored in a specific variable within a complex data structure representing the car object. Which addressing mode would be suitable for the instruction that stores the new position, and how would you communicate this addressing mode choice to another programmer working on the same code?

(c) *Dynamic difficulty adjustment:* The game can adjust difficulty based on player performance. An array stores pre-defined difficulty settings. Based on the player's score, the program needs to access a specific difficulty setting within the array. Which addressing mode would be appropriate, and how could you modify the code to dynamically adjust the difficulty based on an external factor (player score)?

(d) *Collision detection:* To check for collisions, the game needs to compare the car's position data with the positions of obstacles stored in an array. How would you use addressing modes to efficiently compare the car's position with multiple obstacle positions stored in memory? Explain your approach to a teammate who might be less familiar with addressing modes.

(e) *Optimizing performance:* The game needs to run smoothly on various mobile devices. How can your understanding of addressing modes help you optimise the game code for performance? Explain your approach considering the trade-offs between different modes.

(f) Research another addressing mode not covered in this section (such as, base displacement addressing). Explain a scenario in the game engine where this addressing mode might be useful and why.

Register Transfer Language

Register Transfer Language (RTL) is a symbolic notation used to describe the flow of data between registers and other functional units within a digital system, like a CPU. It acts as an intermediate representation (IR) between high-level programming languages and the actual machine code instructions executed by the processor. RTL helps us understand how data is manipulated within the CPU at a register level, providing a more detailed view compared to machine code. RTL uses the common addressing modes discussed in the previous sections.

FOR FULL FREE READING ONLY

Micro-operations

The basic operations are performed on data in registers or between registers and other units (ALUs, memory). Table 1.13 shows common symbols in RTL operations.

(a) *Arithmetic operations (add, subtract)* - **Example:** R1 + R2 -> R3 (Add the contents of R1 and R2, store the result in R3)

(b) *Logical operations (AND, OR, NOT)* - **Example:** R1 AND R2 -> R3 (Perform a logical AND operation on the contents of R1 and R2, store the result in R3)

(c) *Data transfer (move):* The movement of data between registers or between a register and another unit. RTL uses notations to specify the source and destination of the data transfer, along with control signals that enable the transfer. **Example:** R2 <- R1 (Move the contents of R1 to R2). For example Figure 1.71 indicates that if T=1, then the content of R1 is transferred to R2.

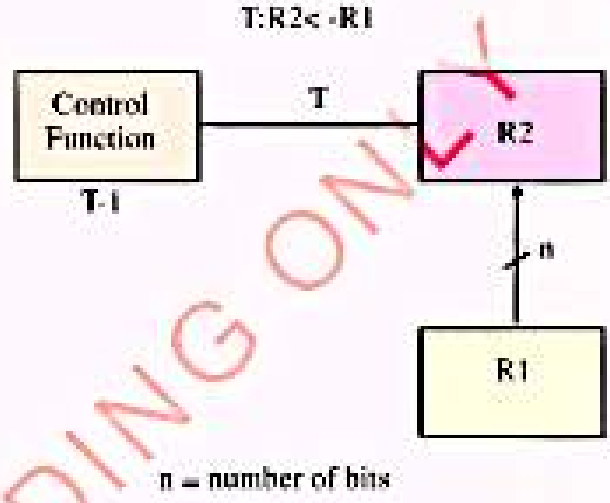


Figure 1.62: *Data Transfer in RTL operations*

(a) Shift operations (left, right)

(b) Control operations (compare, jump)

Table 1.13: *Common symbols of RTL*

Symbol	Meaning	Example
Letters	Registers	MAR, PC, IR
<-	Transfer of Information	R3 <- R2, PC <- PC + 1
,	Two micro-operations	R3 <- R2, R5 <- R4
:	Conditional operations	T:R2<-R1 if T=1
()	Part of a register	R1(0-7)

Computer processor architecture (ARM- Advanced RISC Machine and x86 architecture)

There are two famous architectures, namely Advanced RISC Machine (ARM) and x86.

Advanced RISC Machine (ARM): Just like cheetahs, which are the fastest land animals on Earth but can not sustain their top speed for long distances, ARM follows a similar principle of careful energy usage. ARM is a Reduced Instruction Set Computing (RISC) architecture, meaning it has simpler and fewer instructions. This

FOR ONLINE READING ONLY

simplicity makes ARM processors highly energy-efficient, making them ideal for mobile devices like smartphones and tablets.

x86: A powerhouse for performance: Think of a heavyweight boxer with a wide range of powerful punches. That is the x86 architecture, known for its Complex Instruction Set Computing (CISC) design. x86 offers a larger set of complex instructions, making it a powerhouse for demanding tasks on desktops and laptops. Table 1.14 provides a comparison between ARM and x86 architectures.

Table 1.14: *Characteristics of architecture of ARM Cortex-A8 compared to Intel Core i7*

Characteristic	ARM Cortex-A8	Intel Core i7
Architecture classification	Von Neumann architecture	Von Neumann architecture
Processor/cores	One processor	Four processors
Instruction Set Architecture (ISA)	ARMv7	x86-64
Main Memory and technologies	Physical address: 32 bits	Physical address: 44 bits DDR4 SDRAM for main memory Levels of cache (L1, L2, L3) implemented with SRAM for faster access.
Cache memory	L1 cache organisation: Split instruction (I) and data (D) caches L1 cache size: 32 KiB each for instructions/data L1 cache associativity: 4-way (I), 4-way (D) set associative L1 replacement algorithm: Random //Similar to L2 L1 hit time: 1 clock cycle	L1 cache organisation: Split instruction (I) and data (D) caches L1 cache size: 32 KiB each for instructions/data L1 cache associativity: 4-way (I), 8-way (D) set associative L1 replacement algorithm: Approximated Least Recently Used (LRU) //Similar for L2, L3 L1 block size: 64 bytes // similar for L2,L3

FOR ONLINE READING ONLY

	L2 cache organisation: Unified (instruction and data) L2 cache size: 128 KiB to 1 MiB L2 cache associativity: 8-way set associative L2 hit time 11 clock cycles	L1 hit time: 4 clock cycles, pipelined L2 cache organisation: Unified (instruction and data) per core L2 cache size: MiB 256 KiB (0.25 MiB) L2 cache associativity: 8-way set associative L2 hit time: 10 clock cycles L3 cache organisation: Unified (instruction and data) L3 cache size: 8 MiB, shared L3 cache associativity: 16-way set associative L3 hit time 35 clock cycles
Instruction pipelining	Shallow pipeline	Deeper pipeline with more stages
Power consumption	Lower power consumption	Higher power consumption
Typical applications	Mobile devices (smartphones, tablets)	Desktop computers, laptops, high-performance computing

FOR ONLINE READING ONLY

Exercise 1.15

1. You are designing a new wearable fitness tracker. Would you choose an ARM or x86 processor for this device? Give reasons.
2. Both ARM and x86 architectures are constantly evolving. Research and describe a recent development in either ARM or x86 technology. How does this advancement impact the capabilities of processors?
3. You are participating in a debate about which processor architecture is "superior." Would you argue for ARM or x86? Develop a persuasive argument highlighting the strengths and ideal applications for your chosen architecture.

Input and Output systems

Scenario 1.3: *Restaurant system for customer service*

To understand the concept of input output system in computer architecture, imagine a restaurant with a well-coordinated system for handling orders, deliveries, and customer needs. This system is analogous to the Input/Output (I/O) systems in computer architecture, facilitating communication between the processor (the kitchen) and external devices (the outside world). Think of Direct Memory Access (DMA) as a dedicated waiter assigned to handle large food deliveries. DMA allows devices to transfer data directly between memory and external devices without involving the processor, improving efficiency for bulk data transfers. Just as the dedicated waiter frees up the kitchen staff to focus on cooking, DMA frees up the processor for other tasks. Imagine a fire alarm going off in the restaurant. This unexpected event disrupts the normal workflow, requiring immediate attention from the staff. Similarly, interrupts and exceptions are signals from external devices or errors within the system that require the processor's immediate attention, temporarily pausing ongoing tasks. To understand Privileged and Non-Privileged Instructions, imagine only the head chef has access to the master control panel for adjusting ovens.

This control panel represents privileged instructions, which require a higher level of access and can only be executed by the operating system or specific programs. Non-privileged instructions, like those used for basic cooking steps, are accessible to any program running on the system. The restaurant's kitchen relies on a well-stocked pantry and an efficient delivery system to function. Memory acts as the "pantry" for the processor, storing data and instructions needed for running programs. The I/O system is like the "delivery system," handling the flow of data between the processor and external devices like keyboards, printers, disks, and networks.

Questions:

1. While DMA is great for large deliveries, how would the restaurant handle individual customer orders from their tables? Is there an analogous concept in computer I/O systems for dealing with smaller data transfers? Give reasons.
2. Imagine multiple events happening at once - a fire alarm goes off while a large food delivery arrives. How would the restaurant staff prioritize handling these urgent situations? How does this relate to the way processors handle interrupts and exceptions with varying priorities?

Why might the head chef have restricted access to the master control panel? How does this concept translate to privileged instructions in computer architecture? Provide any security risks associated with non-privileged programs having access to certain functions.

Direct Memory Access

Direct Memory Access (DMA) is a crucial technology for optimising data transfer in computer systems.

DMA relies on a dedicated hardware component called the DMA controller. This controller acts as the manager, coordinating the transfer of blocks of data between memory and external devices, as depicted in Figure 1.63. The Direct Memory Access (DMA) controllers optimise data transfer in computer systems by sharing the system bus with the processor. The concept of a "bus master" dictates which device controls data transfers at any given moment.

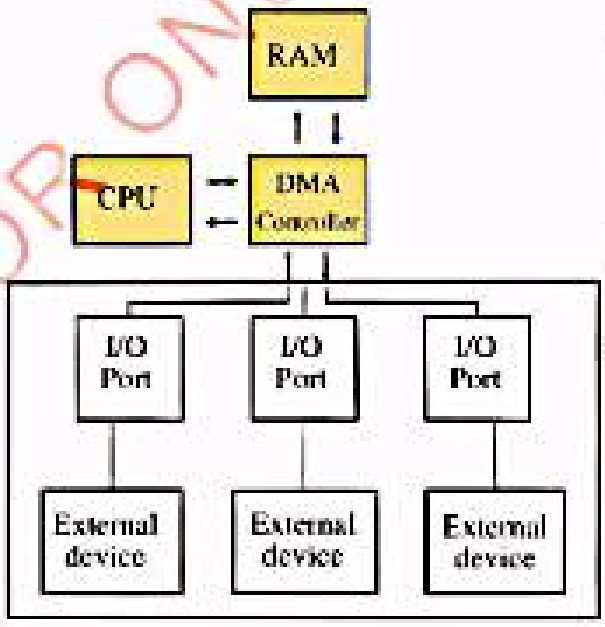


Figure 1.63: *DMA controller*

As seen in Figure 1.63, the DMA controller coordinates data transfer in the following manner;

(a) To initiate a DMA transfer, the processor first sends a request to the DMA controller, specifying details like memory addresses, data size, and transfer direction. This information travels along the address bus. The processor then suspends its current program execution, increments the program counter, and saves its current state so that it can get back to this state after DMA stansfer is completed . Finally, it signals the DMA controller's availability by asserting a DMA select signal.

(b) The DMA controller, if free, responds by requesting control of the bus through a bus request signal. Upon receiving a bus grant signal from the processor, the DMA controller becomes the bus master. This allows the processor to resume program execution. Now in control, the DMA controller interacts directly with memory and the external I/O device, transferring data blocks efficiently according to the previously received instructions.

(c) Once the transfer is complete, the DMA controller gives up control by deactivating the bus request signal. The processor recognizes this and regains control of the system bus by deactivating the bus grant signal. In addition to this, I/O devices can also start transfers by sending a DMA request to the controller. If

FOR ONLINE READING

the controller is available, it acknowledges the request and then takes control of the bus to facilitate data transfer from the device.

(d) Multi-channel DMA controllers further enhance efficiency by enabling connections to multiple external devices (up to the number of channels). This allows for concurrent data transfers from various devices, significantly improving overall system throughput.

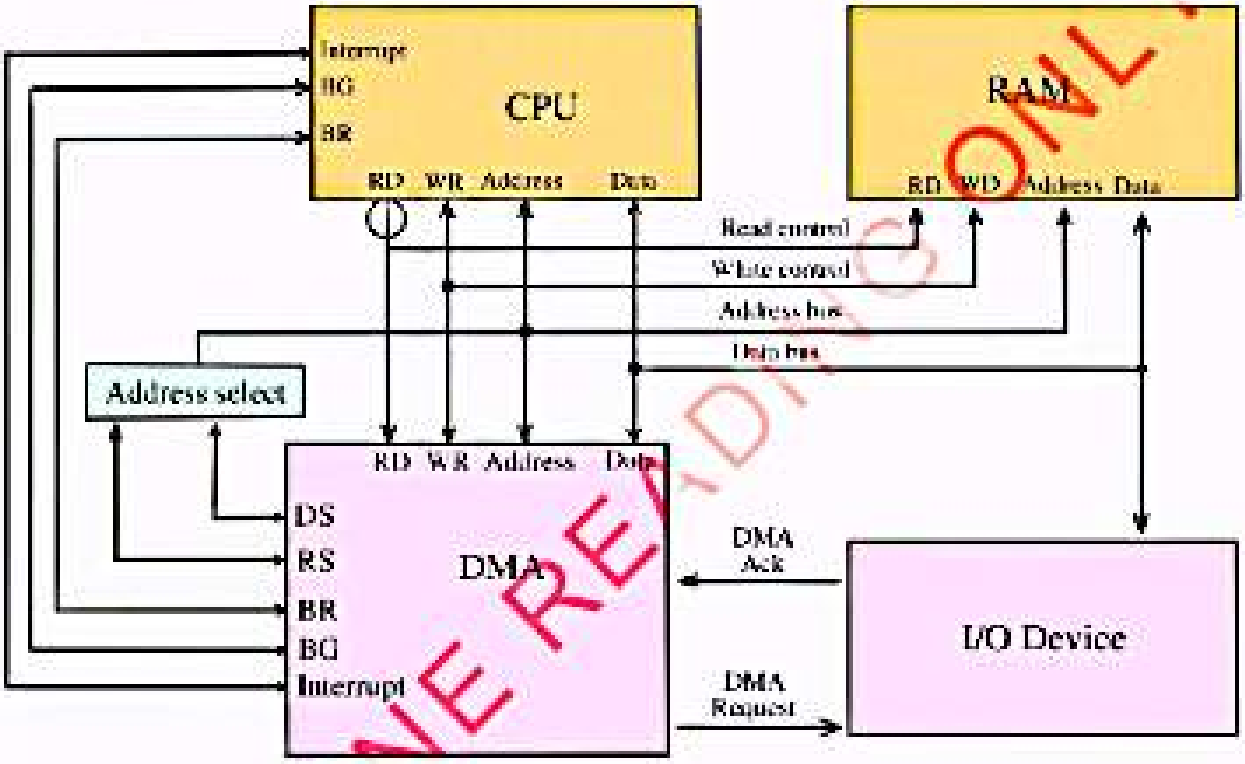


Figure 1.64: *Transfer of data by DMA controller*

FOR ONLINE READING ONLY

Types of DMA transfers

There are different types of DMA transfers, each with its own advantages:

(a) *Burst mode:* In this mode DMA handover the buses to the CPU only after the completion of the whole data transfer. Meanwhile, if the CPU requires the bus it has to stay ideal and wait for data transfer. In the burst mode, DMA transfers data in larger blocks, minimising processor interruptions and improving efficiency. This is like the waiter making multiple trips with a large tray full of dishes, rather than going back and-forth for each individual plate.

(b) *Cycle stealing mode:* In this mode, DMA gives control of buses to CPU after the transfer of every byte. It continuously issues a request for bus control, makes the transfer of one byte and returns the bus. This way, the CPU does not have to wait for a long time if it needs a bus for higher priority task. While efficient for short bursts, it can introduce delays in processor execution.

(e) *Transparent mode:* Here, DMA transfers data only when the CPU is executing the instruction which does not require the use of buses.

(d) *Scatter-gather DMA:* Useful for transferring non-contiguous data blocks in memory to a contiguous location in the external device and vice versa. Imagine the waiter collecting ingredients from different shelves in the kitchen (scatter) and bringing them together for the chef (gather)

Benefits of DMA Technology

(a) *Improved performance:* Reduced processor involvement in data transfers translates to faster data movement between memory and external devices, leading to a noticeable performance boost for tasks like video streaming and disk I/O operations.

(b) *Increased system throughput:* Processors remain free to handle other tasks while DMA manages data transfers, resulting in higher overall system throughput. Imagine the chefs in the restaurant being able to prepare more dishes simultaneously because the waiter efficiently handles deliveries.

(c) *Reduced processor load:* Offloading data transfer responsibilities from the processor allows it to focus on core computational tasks and reduces overall workload.

Real-world applications of DMA: DMA plays a critical role in various everyday tasks:

(a) *High-speed network data transfers:* DMA is essential for efficient network card interactions, enabling smooth data flow across networks.

(b) *Video and audio data streaming:* DMA ensures smooth streaming of video and audio content by managing data transfers between memory and graphics/audio cards efficiently.

(c) *Disk I/O operations:* DMA accelerates data transfer speeds between storage devices and memory, improving hard drive and disk operation efficiency.

(d) *Scientific computing:* DMA is crucial for data-intensive scientific computing applications, ensuring fast and efficient transfer of large datasets.

(e) *Real-time data processing:* Real-time processing systems rely on DMA to handle high volumes of data transfers effectively.

Exercise 1.16

1. DMA allows direct memory access, bypassing the processor's cache. How might this impact data consistency between the processor and memory? Propose potential solutions to mitigate this challenge.
2. While DMA offers performance benefits, it adds hardware

FOR ONLINE READING ONLY

complexity. Research the cost implications of implementing DMA controllers in computer systems. How does this cost factor into decisions about using DMA?

3. Research and discuss specific examples of how DMA technology is utilised in various fields like networking, multimedia, scientific computing, or real-time data processing. Prepare a short presentation highlighting these real-world applications and their significance.
4. Simulate a simple DMA transfer scenario using a flowchart or pseudocode, outlining the steps involved from processor initiation to data transfer completion.

Interrupt and exception

The restaurant analogy we used for I/O systems also applies to interrupts and exceptions in computer architecture. Imagine the well-coordinated restaurant staff working diligently. While the chefs (processor) focus on preparing meals (computations), unexpected events can occur, requiring their immediate attention. These events are analogous to interrupts and exceptions in a computer system.

Interrupts

An interrupt is like a fire alarm going off in the restaurant. This disrupts the normal workflow and demands the staff's immediate response (handling the fire). Interrupts are signals generated by external devices or internal hardware components that require the processor's immediate attention. These signals temporarily halt the processor's current task and divert its focus to handling the interrupt. Examples of interrupts are; a key being pressed on the keyboard and a network card receiving data. Different interrupts can have varying priorities. Just as a fire alarm takes precedence over a customer placing an order, high-priority interrupts (like hardware failures) preempt lower-priority ones (like keyboard input).

Exceptions

Exceptions are like the head chef noticing smoke coming from the oven (an error condition). This unexpected situation requires the chef's attention to diagnose and fix the problem (handle the exception). Exceptions are internal events generated by the processor itself during program execution. These events can be errors (like division by zero), privilege violations (attempting to access unauthorised memory), or other unexpected conditions that require the processor's intervention. The processor has built-in mechanisms to handle exceptions. These mechanisms typically involve saving the processor's state, identifying the exception type, and executing specific code (exception handler) to address the issue.

Context switching

Both interrupts and exceptions involve context switching, where the processor

FOR ONLINE READING ONLY

saves its current state (registers, program counter) and switches to handling the interrupting event. This context switching can introduce slight performance overhead, but it is essential to ensure proper system responsiveness and error handling.

Interrupt Service Routines (ISRs)

Each type of interrupt has a corresponding Interrupt Service Routine (ISR) that defines the specific actions the processor takes in response to that interrupt. ISRs are typically short code segments that handle the interrupting event and then return control back to the interrupted program.

Exercise 1.17

1. MIPS utilises the cause and status registers to manage interrupts and exceptions. Research the specific bit positions and corresponding causes within the Cause register as seen in Table 1.15 and 1.16. Investigate the status register and its role in interrupt handling within MIPS ISA. Explain how these registers work together to differentiate between interrupts and exceptions.

Table 1.15: *Codes for MIPS interrupt/exception cause register*

Number	Name	Description
00	INT	External Interrupt
01	IBUS	Instruction Bus Error (Invalid Instruction)
10	OVF	Arithmetic overflow
11	SYSCALL	System call

Table 1.16: *Mask bits for MIPS Status register*

Bit	Interrupt/Exception
3	INT
2	IBUS
1	OVF
0	SYSCALL

2. While MIPS utilises polled interrupts, many modern systems employ vectored interrupt tables. Research the concept of vectored interrupt tables and compare them to polled interrupts. Discuss the advantages and disadvantages of each approach. Identify real-world applications where interrupt and exception handling in MIPS ISA plays a crucial role (such as, device interactions, and system calls).

FOR ONLINE READING ONLY

Privileged and non-privileged instruction

Imagine a well-organised restaurant kitchen called CPU, where tasks are divided for efficiency and security. The head chef represents the operating system, having complete control over ingredients (resources) and appliances (hardware). Other cooks represent user programs and are responsible for specific dishes (tasks), but they need permission to access certain ingredients and appliances. This analogy explains privileged and non-privileged instructions in ISA.

Again, consider a real-world example: a word processing program (user program) wants to save a document to the hard drive (a privileged operation). However, it can't directly access the disk. Instead, it uses a system call (like asking the head chef for permission) to request the operating system to perform the save operation on its behalf. The operating system then uses privileged instructions to interact with the disk controller and write the data. Privileged instructions are like secret recipes or master control panels in the kitchen. They ensure smooth operation and protect valuable resources. By limiting access to these important functions, the operating system maintains a stable and secure environment for user programs to carry out their tasks

Privileged instructions

Like the head chef's control panel for adjusting ovens, privileged instructions require a higher level of access (Ring 0) and can only be executed by the operating system or specific programs with authorisation. Examples of such instructions in x86 architecture are;

(a) *MOV to/from Control Registers (such as CR0 and CR3):* These instructions configure essential CPU behavior, like memory protection and paging. Tampering with them could lead to system instability. Example: MOV CR0, eax - This instruction moves the contents of the EAX register to the Control Register 0 (CR0), potentially enabling features like paging or write-protection.

(b) *I/O instructions (IN, OUT):* Direct interaction with hardware devices like disks or network cards requires privileged access to prevent conflicts and data corruption.

(c) *HLT (Halt):* This instruction puts the processor into a low-power state, effectively stopping all program execution. Only the OS should have the authority to do this.

Non-privileged instructions

These instructions are like the everyday cooking tasks assigned to regular cooks. They can be used freely by user programs (Ring 3) without special permission. The operating system monitors non-privileged instructions to prevent unauthorised access to critical resources through privilege checks. An attempt to execute a privileged instruction without permission results in a protection fault exception (like setting off an alarm in the kitchen). Examples of such instructions in x86 architecture are;

FOR ONLINE READING ONLY

(a) Arithmetic and logical operations (ADD, SUB, AND, OR): Performing calculations on data in registers is a fundamental task for any program.

(b) Data transfer instructions (MOV, LEA): Moving data between registers and memory locations is essential for program logic and data manipulation.

(c) Register-to-register operations (CMP, TEST): Comparing and testing register values are crucial for control flow and decision-making within programs.

There are several benefits of separating instructions into privileged and non-privileged. These include;

(a) Security: Prevents user programs from causing system crashes or security breaches by restricting access to sensitive hardware and resources.

(b) Stability: Ensures the operating system maintains control of essential resources, promoting overall system stability.

(c) Organisation: Provides a structured environment for program execution, allowing for efficient resource management and multitasking.

Exercise 1.18

(a) Complete the following privileged instructions (x86) given real-world kitchen equivalents.

Table 1.17: Privileged instructions (x86) and real-world kitchen equivalents

Instruction	Kitchen equivalent
	Changing the recipe book (Paging table): This instruction modifies the Control Register (CR3), which acts like the head chef's master recipe book (paging table) in our analogy. It tells the CPU where to find specific data in memory, similar to how the chef uses the recipe book to locate ingredient instructions. Tampering with this could lead to confusion and wasted resources (ingredients).
	Checking the oven temperature (input): This instruction reads a byte of data from a port specified in the DX register and stores it in the AL register. Imagine the DX register as the oven number, and the AL register stores the current temperature reading. This is like the cook checking the oven temperature using a special thermometer (privileged instruction).

FOR ONLINE READING ONLY

Turning off the stove (halt): This instruction puts the processor into a low-power state, effectively stopping all program execution. It is like the head chef turning off the stove (halting operations) when the kitchen is closed.

(b) Why is it important to restrict access to privileged instructions? Think about potential consequences if junior cooks had unrestricted access to all ingredients and appliances.

(c) A new recipe (user program) requires deep-frying (privileged operation) a specific ingredient. How can the junior cooks (user program) achieve this task securely and efficiently, following the access control protocols of the kitchen (CPU)?

Function of memory and I/O system

Recall the previous analogy of a restaurant kitchen (CPU) where the head chef (operating system) leads a team of cooks (user programs) to prepare delicious meals (processed data). To function smoothly, this kitchen relies on two critical systems: memory and the Input/Output (I/O) system.

Memory: The well-stocked pantry

Memory acts like the kitchen's well-stocked pantry, storing all the ingredients (data) required for the cooks to prepare dishes (programs). The head chef carefully organises the pantry (memory) to ensure ingredients (data) are easily retrievable when needed. This organisation often involves techniques like paging and segmentation, similar to categorising ingredients on shelves or in labelled containers. Just like cooks cannot prepare a meal without the necessary ingredients, user programs cannot execute their tasks without accessing data stored in memory.

I/O System: The delivery entrance and exit

The I/O system functions like the kitchen's delivery entrance and exit. It handles the flow of ingredients (data) from external sources (like refrigerators and freezers) and delivers prepared meals (processed data) to customers (external devices). Similar to how cooks would not have everything they need readily available in the pantry, user programs often require data from external devices like hard drives or network cards. The I/O system facilitates this communication, ensuring a smooth exchange of information. Additionally, the I/O system is crucial for delivering the final results (processed data) generated by the user programs. This could involve sending data to a display screen (like serving a dish to a customer) or storing it on a storage device (like packing leftovers for takeaway). In essence, memory and the I/O system work together seamlessly

FOR ONLINE READING ONLY

to provide the essential ingredients (data) and communication channels for the user programs (cooks) to function effectively within the CPU kitchen (processor), ensuring the successful preparation and delivery of processed information.

Activity 1.16:

Building a simple computer

Aim:

To enhance understanding of how to optimise the design to efficiently meet the specified requirements.

Materials:

Components for the computer system: Simulation tool (such as Logisim and simul IDE), Central Processing Unit (CPU), Memory (RAM), Storage device (such as hard drive or solid-state drive); Input devices (such as keyboard, mouse); Output devices (such as monitor, speakers); Power supply, Cables and connectors; Basic tools like screwdrivers.

Instructions:

ABC Company sponsoring young innovators who have innovative ideas. Assume your school teacher assign you and your team a task to design a simple computer system from scratch to showcase to the sponsor. The computer needs to be capable of performing basic arithmetic operations (addition, subtraction, multiplication, and division), storing and retrieving data. Your school has limited resources and budget, you need to optimise your design to meet the requirements efficiently.

Step 1: Demonstrate the key requirements you will consider to design a simple computer

Step 2: Use simulation tool, visualise how each parts (such as Input/Output, CPU, memory) of a computer you designed works and show how different parts relate.

Step 3: Present the model or the blue print of a simple computer you designed

Deliverable:

A comprehensive document outlining the design proposal for the simple computer system, simulation of a simple designed computer.

Exercise 1.19

1. What is the primary function of the memory system in a computer?
2. How does the I/O system contribute to the overall functionality of a computer?
3. What are the key responsibilities of the memory system in managing data and instructions?
4. How does the I/O system facilitate communication between the computer and external devices?
5. What are some common challenges faced by the memory and I/O systems, and how are they addressed in modern computer architectures?

FOR ONLINE READING ONLY

Chapter Summary

Reflect on what you have learnt from this chapter, summarise and keep it in your portfolio.

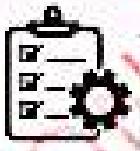
Revision Exercise 1

1. Artificial intelligence (AI) and machine learning (ML) are rapidly evolving fields. How can you design your computer architecture to accommodate the ever-growing processing demands of AI and ML algorithms, considering factors like scalability and energy efficiency?
2. Ethical considerations are paramount in technology development. Research potential security risks and privacy concerns associated with advanced computer architectures. Propose solutions to mitigate these risks, ensuring the responsible and ethical development of your future computer system.
3. Modern processors employ multiple cache levels. Analyse the trade-offs between increasing the cache size (L1, L2, L3) and the impact on AMAT.
4. How would increasing associativity (number of ways) within a cache level affect performance? Consider factors like hit rate, miss penalty, and overall access time.
5. How might the traditional memory hierarchy need to adapt to efficiently handle large, complex datasets? Propose potential modifications or new memory technologies that could be beneficial for big data and Machine Learning applications.
6. Cache misses can introduce vulnerabilities in certain scenarios. Explain how a malicious program might exploit cache misses to gain unauthorised access to sensitive data.
7. Discuss potential mitigation strategies that computer architects or software developers can implement to improve cache security.
8. Flash memory has become a prominent storage solution, particularly in mobile devices. Brainstorm three creative ways flash memory technology could be further utilized in various aspects of everyday life beyond just storing data on smartphones and tablets.
9. As technology evolves, so does memory technology. Research emerging memory technologies that scientists are exploring. Briefly describe one such technology and discuss its potential advantages

FOR ONLINE READING ONLY

and disadvantages compared to traditional memory solutions.

10. Imagine you are designing a new type of handheld gaming device. Considering factors like battery life, processing power, and game complexity, what type of ISA (RISC or CISC) would be most suitable for this device? Explain your reasoning.
11. Discuss the the advantages and disadvantages of having a standardised ISA for all computers.
12. The passage mentions the emergence of custom ISAs for specific applications. Research and describe one example of a custom ISA and its intended use case. How might custom ISAs impact the future of computing?
13. Provide an example of a MIPS instruction that utilises the immediate data input (immed) as shown in the datapath in figure 1.14 and explain how it is used in operation.
14. Imagine explaining the differences between ARM and x86 processors to a friend who is not familiar with computer architecture. Break down the key concepts in an easy-to-understand way, using analogies or real-world examples.
15. Think about the environmental impact of different processor architectures. How does the lower power consumption of ARM processors contribute to a greener future for technology? Research and share some information on sustainable computing practices.



Project

Design simple traffic light controller using logic gates. Simulate your design in Logisim simulator

FOR ONLINE READING ONLY

Chapter Two

Performance and optimisation of computer processor

Introduction

This chapter focuses on techniques that can be applied to improve the performance of a processor and reduce access time. In this chapter, you will learn the concept of pipelining, Instruction Level Parallelism (ILP), Thread and data Level parallelism.

Knowledge regarding the performance and optimisation of computer processor is fundamental in computer science, as it will equip you with the ability to analyse processor performance bottlenecks, measure and quantify processors performance, and parallelise programs/instructions effectively. The competencies developed in this chapter will prepare you to design, build and use efficient, cost-effective and high performing computer systems.

FOR ONLINE READING ONLY



Think:

How a finely tuned engine boosts a car forward.

Pipelining

Concept of pipelining



Scenario 2.1: *Kitchen as a simple processor*

Imagine Mama Chapati, a popular stall known for its delicious flatbreads where hungry students line up for their orders. However, during peak hours, the wait times can be frustratingly long. Let us take a closer look at Mama Chapati's kitchen and see how it works, using the concept of pipelining to understand how efficiency can be improved. Think of Mama Chapati's kitchen as a simple processor, with each step acting like a stage in the process. Assume each customer orders only one chapati. Here is what happens in a non-pipelined approach:

1. Order taking (5 mins): Mama chapati greets the customer and takes their order for a chapati.
2. Dough preparation (10 mins): She measures flour, water, and salt, then kneads the dough by hand. This can be quite time-consuming.
3. Rolling (5 mins): Once the dough is kneaded, she rolls it out into a thin circle. While rolling, she also heats the frying pan on the cooking stove.
4. Cooking chapati (5 mins): The rolled-out dough is carefully transferred to the hot frying pan and cooked until golden brown.
5. Serving (5 mins): Finally, the cooked chapati is presented to the customer, and Mama chapati is ready to take the next order.

The problem: This non-pipelined approach creates a bottleneck at the kneading and rolling stages. Mama chapati can only move on to the next order after finishing the entire chapati for the previous customer. This leads to long wait times, especially during peak lunch hour. Just like in a modern computer processor, Mama chapati's kitchen can be optimised using pipelining. Mama chapati can have kitchen assistants at each stage; taking orders, preparing dough, rolling, cooking and then serving. Therefore, once an order for customer 1 moves to the second stage of dough preparation, Mama chapati can concurrently receive order from customer 2. When the order of customer 1 moves to the third stage of rolling, the order of customer 2 can move to stage two of dough preparation; hence Mama chapati can concurrently take the order from customer 3. By the time a cooked chapati is served to customer 1, Mama chapati will be taking the fifth order from customer 5. Pipelining offers many benefits such as increased throughput: by working on multiple stages simultaneously, Mama Chapati can prepare chapatis much faster, reducing customer wait times significantly.



Questions:

(a) How much speedup is gained by introducing a pipeline into the process of cooking chapatis?

(b) Think about the challenges of pipelining in Mama Chapati's kitchen. What if the frying pan is already occupied when a new chapati needs to be cooked? How can Mama chapati address this data dependency?

FOR ONLINE READING ONLY

Meaning of pipelining

Pipelining is an implementation technique in which multiple instructions are overlapped in execution. A pipeline is divided into stages and these stages are connected with one another to form a pipe like structure. Instructions enter from one end and exit from another end. In a pipelined processor, there are two main ends: the input end and the output end. Between these ends, there are several stages or segments, each of which is responsible for a specific operation. The output of one stage connects to the input of the subsequent stage. Interface registers are used to hold intermediate output between two stages. These interface registers are also known as latches or buffers.

While the specific names and number of stages can vary between Instruction Set Architectures (ISAs) like RISC and CISC, many modern processors, including those based on RISC and MIPS principles, often utilise a 5 or 6 stage pipeline. Here is a breakdown of a common 5-stage pipeline:

1. *Instruction Fetch (IF):* Fetches the next instruction from memory.
2. *Instruction Decode (ID):* Decodes the instruction and reads the required registers.
3. Execute (EX): Performs the arithmetic or logic operation specified by the instruction.
4. *Memory Access (MEM):* Accesses memory, if necessary, for load or store operations.
5. *Write Back (WB):* Writes the result back to the register file.

Start one instruction per clock cycle as depicted in Figure 2.1. Each instruction still takes 5 cycles, but instructions now complete every cycle: Cycles Per Instructions (CPI) =5/5 = 1.

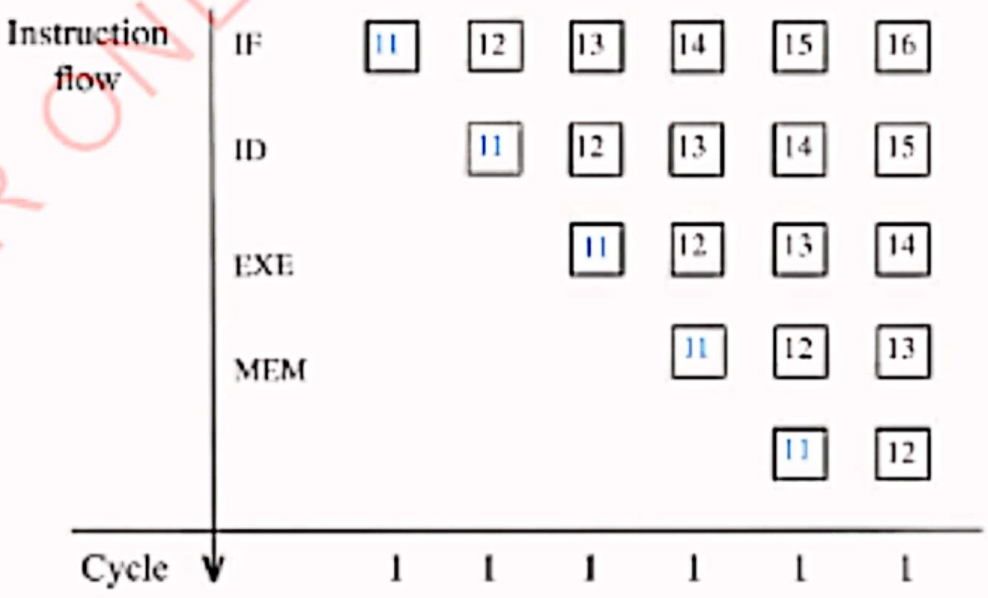


Figure 2. 1: *A pipelining Diagram*

FOR ONLINE READING ONLY

Activity 2.1:

Simulation of the pipeline stages and movement of instruction in Computer Memory and Cache using CPU Simulator

Aim:
To visualise and understand the pipeline stages and instruction movement in CPU, RAM, and Cache Memory using a CPU Simulator.

Material:
Computer, CPU simulator software or Internet

Instructions:
Visualise the pipeline stages and the movement of instructions in Memory (RAM) and Cache Memory

Step 1: Study the components of the CPU

Step 2: Create a program

To create a new program, click the "program" button, and enter the name in the "Program Name" box (for this activity enter "CacheTest1"). Also enter the "base address" in the "Base Address box" (You can enter any number, or for this activity, enter "101") then click on the "ADD button". The new program's name will appear in the "Program List view"

Step 3: Add program instructions

(a) Enter instructions as they appear in the "CPU INSTRUCTION IN MEMORY(RAM)" in the top left of the screenshot in Figure 2.2.

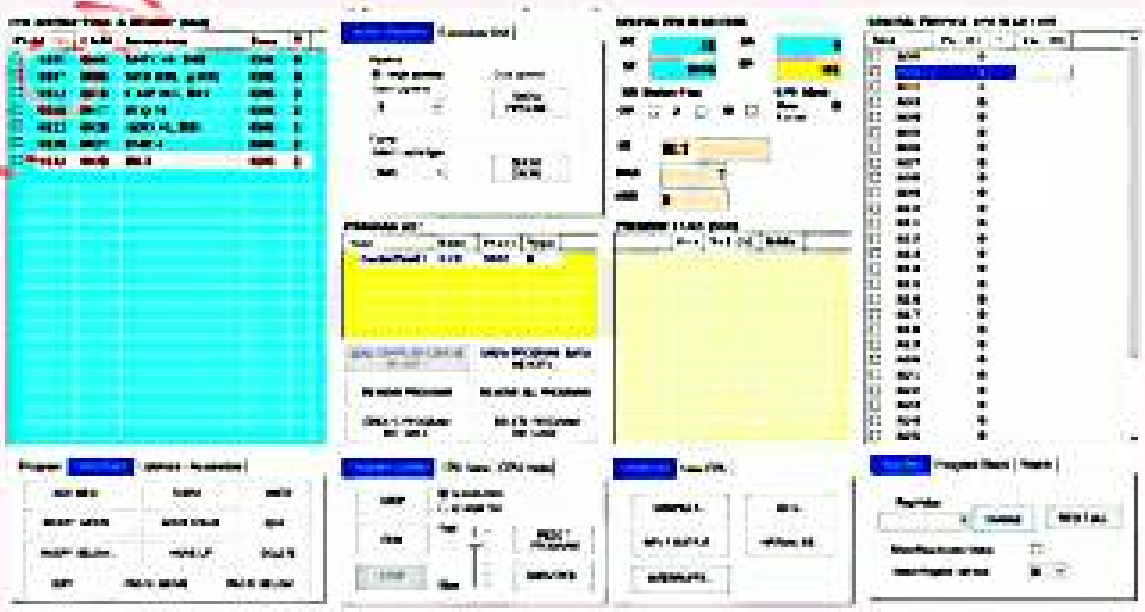


Figure 2.2: *Add program instructions*

FOR ONLINE READING ONLY

(b) Observe the pipeline stages of the instructions created in the following screenshot

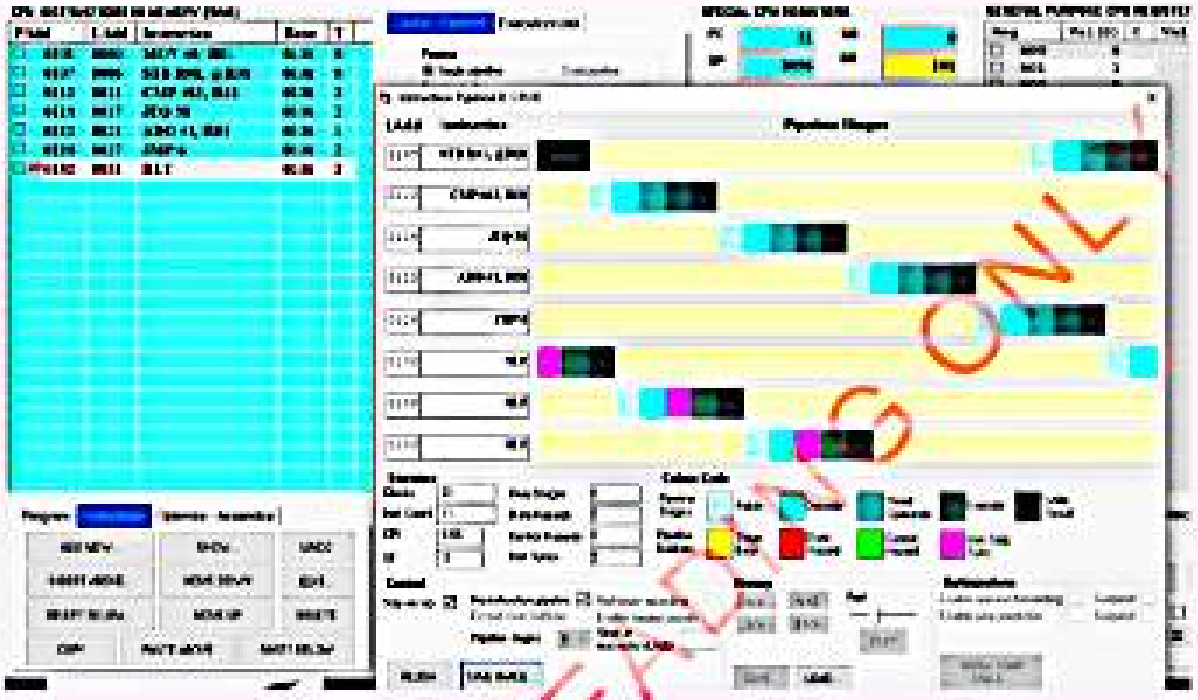


Figure 2.3: *Pipeline stages*

(c) Add the program instructions as they appear in the following screenshot. Run the simulation and observe the results.

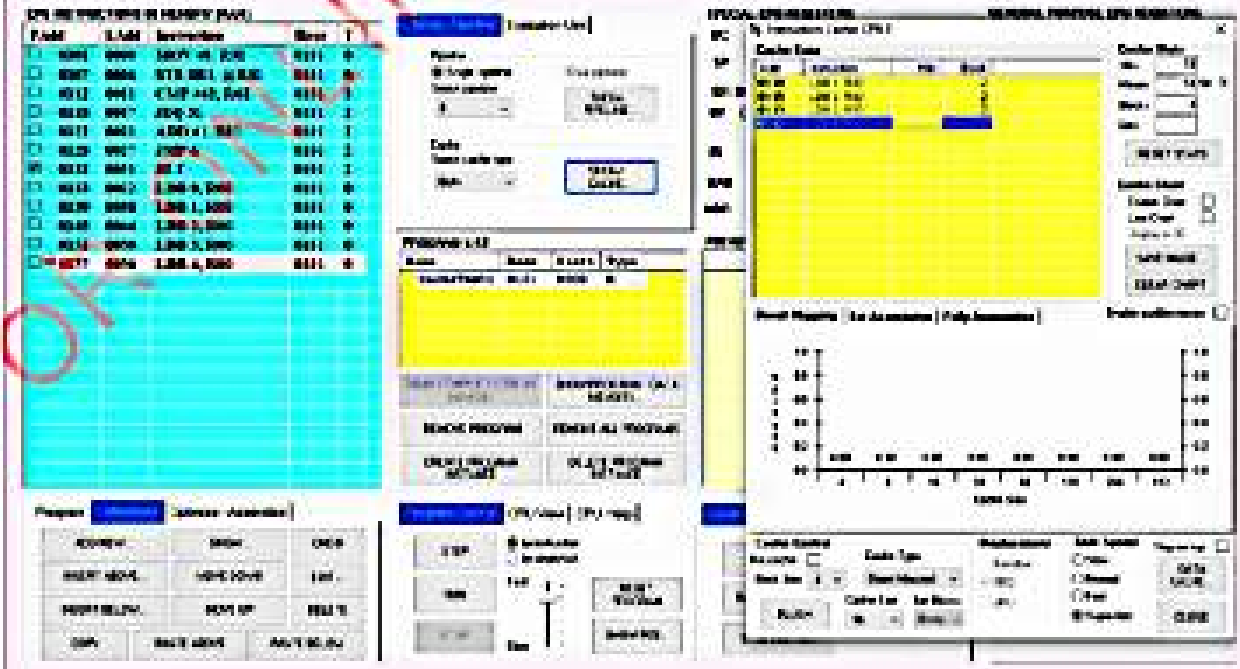


Figure 2.4: *Add the program instructions*

FOR ONLINE READING ONLY

Activity 2.2:

Simulation of the pipeline stages and movement of instruction in Computer memory and cache using CPU Simulator

Aim:
To visualise and understand analysis of LDB instruction in CPU Simulator by simulating pipeline stages and instruction movement

Material: Computer, CPU simulator software or internet

Instructions:
Conduct a cache performance analysis of LDB Instruction in CPU Simulator by repeating steps in Activity 2.1 with the LDB values between 0 to 10

Step 1: Record the experiment results using the parameters in the following table. The first row is an example.

Addr	Instruction	Hits (%)	Misses (%)	Block
0139	LDB1, R00	50	50	2

Step 2: Analyse the experiment results and report your observation

Step 3: Calculate the miss rate and Average Memory Access Time (AMAT)

Pipelining and increase in throughput

Consider the analogy of the Mama chapati kitchen, which breaks the process of cooking and serving chapatis in five stages. Stage 1: Receiving an Order (O), Stage 2: Kneading dough (N), Stage 3: Rolling (R) Stage 4: Cooking (C) and Serving (S). The average time taken to serve one chapati with or without pipelining is as follows:

Without pipelining: Remember that taking an order takes 5 minites, preparing and kneading dough takes 10 minutes, rolling takes 5 minutes, cooking takes 5 minutes and serving takes 5 minutes.

```
O N R C S | | | | | | | | | | | | | | | |
| | | | | O N R C S | | | | | | | | | |
| | | | | | | | | | O N R C S | | | | |
| | | | | | | | | | | | | | | O N R C S (30*4 minutes)
```

Figure 2.5: *Average time for one chapati = 120/4 = 30 minutes*

FOR ONLINE READING ONLY

With pipelining: if this process is broken into pipelines, the pipeline stages must be able to accommodate the slowest operation, as shown in Figure 2.6 For Mama chapati, the longest stage is kneading which takes 10 minutes.

```
O N R C S | | |
| O N R C S | |
| | O N R C S |
| | | O N R C S (10*8 minutes)
```

Figure 2.6: *Average time = 80 /4 = 20 minutes*

$$Speedup = \frac{Average\ time\ without\ pipelining}{average\ time\ with\ pipelining} = \frac{30\ minutes}{20\ minutes} = 1.5$$

Therefore, pipelined operation improves performance by increasing instruction throughput. Higher throughput means more instructions can be executed in a given unit of time.

The performance of a pipeline is judged by two key metrics: ***throughput*** *and* ***lateney***. Throughput, measured as instructions completed per unit of time, reflects the overall processing speed. A higher throughput indicates faster instruction processing. Factors like pipeline length, clock frequency, efficiency of individual instructions, and the presence of stalls or hazards can all impact throughput.

$$Throughput = \frac{Number\ of\ instructions\ executed}{Execution\ time}$$

On the other hand, latency measures the time taken for a single instruction to finish execution. It essentially represents the delay or travel time an instruction experiences as it progresses through the pipeline stages. Lower latency signifies better performance. Pipeline length, depth (number of stages), clock cycle time, dependencies between instructions, and pipeline hazards all influence latency.

$$Latency = \frac{Execution\ time}{Number\ of\ instructions\ executed}$$

Excution time can be reduced, as shown in Figure 2.7, where a 5-stage pipeline has reduced the execution time for 6 instructions from 30 time units to 10 time units.

FOR ONLINE READING ONLY

Time →

	1	2	3	4	5	6	7	8	9	10
Instruction 1	IF	ID	EX	MEM	WB					
Instruction 2		IF	ID	EX	MEM	WB				
Instruction 3			IF	ID	EX	MEM	WB			
Instruction 4				IF	ID	EX	MEM	WB		
Instruction 5					IF	ID	EX	MEM	WB	
Instruction 6						IF	ID	EX	MEM	WB

Figure 2.7: *Timing diagram for instruction pipeline operation*

Exercise 2.1

1. You are analysing a program running on a MIPS processor with a 5 stage pipeline. The program has a mix of instructions, with the following breakdown: **add:** 40%, **sub:** 20%, **lw:** 20% and **sw:** 20%. All instructions take one cycle each in the execute stage. Memory access for `lw` and `sw` takes an additional cycle in the Memory Access stage.
   (a) Assuming a clock cycle time of 1 ns (nanosecond), calculate the theoretical throughput (instructions completed per second) of this program on a pipelined processor.
   (b) How would the throughput be affected if the program had a higher percentage of `lw` and `sw` instructions compared to arithmetic instructions? Explain your reasoning.
2. Imagine two MIPS processors with different pipeline depths: a 5-stage pipeline (as assumed earlier) and a deeper 8-stage pipeline. Discuss the potential benefits and drawbacks of using a deeper pipeline in terms of performance.

Hazards

Pipelining aims to improve processor performance by allowing multiple instructions to be processed simultaneously. However, there are situations, called hazards, that prevent the next instruction in the instruction stream from executing during its designated clock cycle. Hazards reduce the performance from the ideal speedup gained by pipelining. There are three classes of hazards; structural hazards, data hazards and control hazards.

Structural hazards

These arise from limitations in the hardware. For example, if multiple instructions require the same hardware component (e.g., memory unit) at the same time, a structural hazard can arise.

FOR ONLINE READING ONLY

Imagine fetching data from memory in one stage while another stage also needs to access memory. Without separate memory units, a structural hazard would occur. The solution would be to use separate memory units or add extra hardware to resolve the conflict.

Data hazards

Data hazards occur due to dependencies between instructions, where one instruction depends on the result of a previous instruction that is still in the pipeline. We must ensure that the next instruction does not attempt to access data before the current instruction, because this will lead to incorrect results, as shown in Program Example 2.1.

Program Example 2.1:

Data hazard

add $r1, $r2, $r3 (result is stored in $r1)

sub $r4, $r1, $r5 (reads from $r1)

Here, the sub-instruction depends on the result of the add instruction stored in $r1. Data hazards can be solved through stalling or forwarding.

Control hazards

One example of the common types of instructions in ISAs is the control flow instructions such as branch, jump, etc. To understand branch instructions, let us revisit the mama chapati analogy. Imagine Mama Chapati's kitchen which has grown now implements a ticket system where customers write down their orders on their order slip. This slip acts like a program with instructions for Mama chapati to follow. Consider the two scenarios in the Mama chapati kitchen, one that follows a normal program order and the other where a customer makes a special request.

(a) **Normal order processing (Linear Instructions)**

(i) The customer gives their order slip (such as "chapati").

(ii) Mama Chapati reads the slip and prepares the chapati (takes dough, rolls it, cooks it).

(iii) Mama Chapati serves the chapati and takes the next customer's slip.

(b) **Branching with special requests**

(i) Customer gives their chit (e.g., "chapati with butter").

(ii) Mama Chapati reads the chit and checks for a special instruction (like "butter").

Branch if Butter: If there»s a «butter« instruction, Mama Chapati jumps to a different set of instructions:

Apply butter to the rolled dough

Otherwise (No Butter): If there is no "butter" instruction, she continues with the regular chapati preparation.

(i) Mama Chapati cooks the chapati (with or without butter) and serves it.

(ii) Mama Chapati takes the next customer's slip.

FOR ONLINE READING ONLY

This is similar to a branch instruction. The chit acts like the program, and the "butter" instruction is the condition. The condition is the presence or absence of the "butter" instruction on the chit. If there's "butter," Mama Chapati jumps (branches) to the butter application step before cooking. The specific step for applying butter becomes the target address for the branch.

A control hazard, also known as a branch hazard, occurs when the pipeline makes the wrong decision on a branch prediction and therefore brings instructions into the pipeline that must subsequently be discarded. Imagine planning a picnic based on the weather forecast, only to have it change drastically. The processor needs to adjust its course based on the actual branch outcome, as shown in Program Example 2.2.

Program Example 2.2:

beq $r2, $r1, Label	//Branch to Label if Sr1=$r2
add $r3, $r2, $r1	
lw $r3,4($r2)	
sw $r1,0($r2)	
Label: sub $r4, $r3, $r2	

In Program Example 2.2, the pipeline must wait until after the branch decision to determine the correct instruction address. Control hazards can be solved through stalling or through a more intelligent approach called predictive branching.

In predictive branching, the processor predicts the branch outcome in advance and fetches the predicted instruction. We will cover branch predictions in another section.

Exercise 2.2

1. Use resources such as the internet and the library to research on the following types of data hazards: Read After Write (RAW), and Write After Read (WAR).

2. The following instructions represent a program to be processed by a simplified processor pipeline. Identify the type(s) of hazards for each instruction, if any. Write "None" if no hazard is present.

   add $r1, $r2, $r3

   sub $r4, $r1, $r5

   lw $r6, 100 ($r6)

   add $r7, $r4, $r8

   beq $r7, $r2, Loop

   add $r9, $r10, $r1

FOR ONLINE READING ONLY

Stalling and forwarding

Stalling

In the previous section, we discovered that pipeline hazards can be resolved by stalling or forwarding. Recall the previous example of two instructions that cause data hazard; add $r1, $r2, $r3 and sub $r4, $r1, $r5. Figure 2.8 shows the pipeline behaviour. The add instruction does not update register $r1 until the end of stage 5, which occurs at clock cycle 5 (Write back stage). However the sub instruction needs that value from $r1 at the beginning of stage 2 so that the subtraction can be executed at clock cycle 4. Therefore, the pipeline must stall, which means staying idle or doing nothing for two clock cycles until the add instruction updates register $r1 at clock cycle 5.

	Clock cycle									
	1	2	3	4	5	6	7	8	9	10
add $r1, $r2, $r3	IF	ID	EX	MEM	WB					
sub $r4, $r1, $r5		IF	ID	Stall	Stall	EX	MEM	WB		
Instruction 3			IF			ID	EX	MEM	WB	
Instruction 4						IF	ID	EX	MEM	WB

Figure 2.8: *Stalling to resolve a data hazard*

Forwarding

As we have seen in the previous example, the pipeline stalls for two cycles to ensure add finishes writing to r1 before sub reads from it. Forwarding on the other hand involves bypassing the pipeline stages and delivering the result directly from the source (such as, ALU output) to the instruction that needs it. Forwarding (bypassing) helps the processor to use early results from the pipeline to resolve dependencies without waiting. This eliminates the stall and improves performance. This is how forwarding works:

(a) **Hardware modification:** The pipeline is modified to include hardware for detecting data dependencies and forwarding results.

(b) **Dependency detection:** During the Decode/Execute stage, the processor identifies dependencies between instructions. For example, it recognises that the sub needs the outcome of adding stored in $r1.

(c) **Forwarding unit:** A forwarding unit is introduced in the pipeline. This unit checks for dependencies and the availability of results from previous stages.

FOR ONLINE READING ONLY

(d) **Forwarding the result:** If a dependency is detected and the result is available (e.g., from the ALU output in the EX stage), the forwarding unit directly sends the result to the dependent instruction.

While forwarding offers performance benefits, it can be more complex to implement compared to stalling. Processors often use a combination of both techniques. Processors prioritise forwarding for frequent data hazard types (such as, ALU output to dependent instructions). Stalling is used for less frequent or complex dependencies since it is a simpler solution.

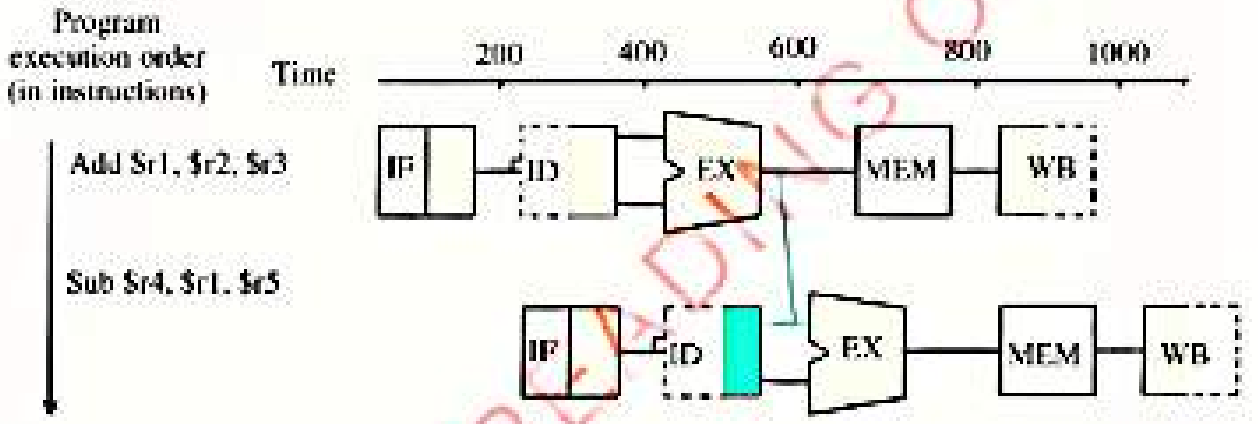


Figure 2.9: *Forwarding to Resolve a Data Hazard.*

In Figure 2.9, the graphical representation of the pipeline has been used. The shading indicates the element is used by the instruction. Hence, MEM has a white background because add and sub do not access the data memory. Shading on the right half of the register file or memory means the element is read in that stage, and shading on the left half means it is written in that stage. Hence, the right half of ID is shaded in the second stage because the register file is read, and the left half of WB is shaded in the fifth stage because the register file is written.

Activity 2.3:

Coin flip challenge for data hazard

Aim:

To simulate a processor pipeline with and without forwarding to understand data hazards and explore the impact on performance.

Materials:

Computer, Simulator or Emulator (such as Ripes, RARS and MARS)

Instructions:

Imagine a program that simulates a coin flip game. It needs to:

(a) Read a user's bet amount (heads or tails).

FOR ONLINE READING ONLY

(b) Randomly generate a coin flip outcome (heads or tails).

(c) Compare the bet and outcome to determine a win or loss.

(d) Update the user's balance based on the result.

A data hazard occurs because instruction 3 (compare) depends on the outcome generated in instruction 2 (random flip). Without proper handling, the pipeline might stall. Complete the following activities.

Part 1: The Penalty of stalling

Step 1: Trace the pipeline (Without forwarding):

(a) List the four instructions involved in the coin flip program.

(b) Create a table with columns for instruction, IF, ID, EX, MEM, WB, and cycle (clock cycle).

(c) Simulate the pipeline flow, assuming a stall if instruction 3 needs data from instruction 2 that is not yet available. Fill in the table, indicating stalls with "Stall" in the corresponding cycle.

Step 2: Observe the impact:

(a) Count the total number of cycles it takes to complete the program with stalling.

(b) Discuss the performance penalty caused by the stall.

Part 2: Forwarding

Step 3: Explain how forwarding could bypass the pipeline stall. Briefly describe the concept of detecting the dependency and sending the result directly from instruction 2 to instruction 3.

Step 4: Trace the pipeline (with forwarding):

(a) Use a new table similar to Part 1, but assume the pipeline forwards the random flip result directly to the compare instruction.

(b) Simulate the pipeline flow with forwarding, avoiding stalls. Fill in the table accordingly.

Step 5: Performance improvement:

(a) Compare the number of cycles taken with forwarding to the number with stalling

(b) Discuss the performance gain achieved by eliminating the stall.

Deliverable: Write the summary of your findings from both parts of the activity and keep in a portfolio.

FOR ONLINE READING ONLY

Exercise 2.3

1. As processor technology advances, new challenges and opportunities arise. Research emerging trends in pipeline hazard mitigation. Are there techniques beyond stalling and forwarding? How do these advancements contribute to the ever-increasing performance demands of modern computing?
2. You are given the following instructions below

   lw $s0, 20($t1)

   sub $t2, $s0, $t3

   (a) Draw the timing diagram to detect hazards. How many stalls are needed to avoid the hazard(s)?

   (b) Now introduce forwarding to resolve the hazard. Has the forwarding helped you to completely avoid stalling the pipeline?

   (c) Based on the answer in 2(b) and based on reputable materials from the library or the internet, state if the following statement is TRUE or FALSE; "You can always avoid stalls in data hazards by forwarding".

Instruction Level Parallelism (ILP)

Concept of Instruction Level Parallelism (ILP)

Scenario 2.2: *Laundry washing machine*

Imagine working at a busy laundromat with high customer demand. Optimising the washing process is crucial for efficiency and customer satisfaction. Traditional washing machines operate step by step (fill, wash, rinse, spin), but Instruction Level Parallelism (ILP) improves efficiency by executing independent tasks concurrently. For example, filling the washer and adding detergent can happen simultaneously, reducing cycle time. ILP also enhances modern machines by processing sensor data (water level, temperature, spin speed) concurrently, adjusting parameters in real-time for optimal washing. Breaking down rinse and spin cycles into smaller, independent instructions further accelerates completion, illustrating ILP's potential for optimising laundry operations.

Questions:

1. Identify other independent tasks within the washing machine that could benefit from ILP. Consider safety measures and potential limitations.

FOR ONLINE READING ONLY

2. How can sensor data be effectively utilized to optimise the washing process using ILP techniques?
3. Explain the concept of ILP to a customer at the laundromat. How does it benefit them from a practical standpoint?

Meaning of Instruction Level Parallelism (ILP)

Parallelism refers to the ability to perform multiple tasks or operations concurrently (at the same time). The concept of Instruction Level Parallelism (ILP) builds upon this foundation, focusing on how processors exploit parallelism within a single program to achieve faster execution. Instruction Level Parallelism is achieved when multiple operations are performed in a single cycle.

Consider a simple MIPS program to add two numbers (A and B) and store the result in register C. For a sequential implementation, as seen in Table 2.1, the processor must load A into a register (such as, register $s1), and then load the value of B into another register (such as, register $s2). Finally, in the third cycle add the values in $s1 and $s2, and store the result in register C (e.g., register $t0).

Table 2.1: *Sequential implementation of adding two numbers*

	Load A	Load B	Add
Cycle 1	A -> $s1		
Cycle 2		B -> $s2	
Cycle 3			C = $s1 + $s2

Modern processors can analyse these instructions and identify opportunities for parallelism. As seen in Table 2.2, the processor can load A into $s1 and B into $s2 simultaneously and then perform the addition using the ALU.

Table 2.2: *ILP implementation of adding two numbers*

	Load A&B	Add
Cycle 1	A -> $s1 B -> $s2	
Cycle 2		C = $s1 + $s2

FOR ONLINE READING ONLY

Approaches of Instruction Level Parallelism (ILP)

(a) Hardware-based approach (Dynamic)

This approach relies on the processor hardware itself to discover and exploit parallelism dynamically during program execution. The processor analyses the instruction stream on the fly, identifying opportunities for parallel execution of independent instructions. Since the compiler does not need complex analysis to identify parallelism, this approach allows for simpler compiler designs. This approach enhances code adaptability since hardware can handle variations in program code without needing code modifications. However, the hardware-based approach presents some disadvantages overhead due to hardware analysis which may lead to lower performance. Also, programmers have less control over how parallelism is exploited. Examples of processors that utilise a hardware-based approach for ILP include Pentium 4 (Intel), AMD Opteron (AMD), and IBM Power (IBM).

These processors employ techniques like:

(i) ***Pipelining*:** Breaking down instruction execution into stages (fetch, decode, execute) allows the processor to work on multiple instructions simultaneously.

(ii) ***Out-of-order execution:*** Instructions can be reordered for execution if no data dependencies exist, potentially improving efficiency.

(iii)***Reservation stations:*** These temporary storage locations hold operands (data) for instructions, enabling faster access during execution.

(b) Software-based pproach (Static)

This approach relies on software tools, specifically compilers, to identify parallelism within a program code during compilation (statically) before the program runs. The compiler analyses the code and rearranges instructions to enable parallel execution. This approach yields higher performance since the compiler can perform a more thorough analysis to uncover parallelism opportunities. Moreover, it gives more control to programmers allowing them to use compiler directives or optimisations to influence how parallelism is exploited. However, this approach requires complex compiler design because compilers need sophisticated algorithms to analyse code for parallelism. It is also presenting less adaptability to changing code. Code modifications might require recompiling to optimise for parallelism. Examples of processors that utilize software-based approach for ILP are Itanium 2 (Intel) processors. Compilers for these processors might use techniques like:

(i) *Instruction scheduling:* Reordering instructions to maximize

FOR ONLINE READING ONLY

parallelism while maintaining program correctness.

(ii) *Loop unrolling:* Duplicating the loop body to expose more potential parallelism within loops.

(iii) *Data dependence analysis:* Identifying dependencies (potential data hazards) between instructions to ensure proper execution order.

Compared to architectures without ILP, ILP processors generally require simpler hardware designs. This is because they rely on efficient pipelining of instructions (breaking down complex operations into smaller, faster steps) to exploit parallelism. Machines without ILP often have more intricate hardware dedicated to handling complex instructions sequentially, making them less efficient in terms of exploiting parallelism. Examples of architectures that exploit ILP are VLIWs and superscalar. The machines without ILP have complex hardware which is hard to implement. Two prominent architectures leverage ILP effectively:

(i) Very Long Instruction Word (VLIW): These processors pack multiple instructions into a single, wider instruction word. This allows them to issue multiple instructions to execution units simultaneously, maximizing parallelism when possible.

(ii) Superscalar architecture: Unlike VLIW, superscalar processors fetch and decode instructions individually. However, they can analyse these instructions for potential parallelism and execute them concurrently on different execution units within the processor. This approach offers flexibility compared to VLIW but might require more complex hardware to identify parallelism opportunities.

Advantages of Instruction-Level Parallelism (ILP)

(a) *Improved performance:* ILP can significantly improve the performance of processors by allowing multiple instructions to be executed simultaneously. This can lead to faster program execution and better system throughput.

(b) *Efficient resource utilisation*: ILP can help to efficiently utilize processor resources by allowing multiple instructions to be executed at the same time. This can help to reduce resource wastage and increase efficiency.

(c) *Reduced instruction dependency*: ILP can help reduce the number of instruction dependencies, which can limit the amount of instruction-level parallelism that can be exploited. This can help to improve performance and reduce bottlenecks.

(d) *Increased throughput:* ILP can help increase the overall throughput of

FOR ONLINE READING ONLY

processors by allowing multiple instructions to be executed simultaneously or out-of-order. This can help to improve the performance of multi-threaded applications and other parallel processing tasks.

Exercise 2.4

1. How identifying data dependencies within a program impact ILP? Explain with an example.
2. Design a simple flow chart illustrating the sequential and ILP execution of a program segment that subtracts two numbers.
3. Discuss the advantages and disadvantages of hardwareand software-based approach to ILP.
4. Discuss the potential disadvantages of ILP in real-world scenarios.

Compilation techniques for ILP

The software-based approach to ILP relies on compilers to identify parallelism within a program code during compilation before the program runs. Compiler techniques used include instruction scheduling, loop unrolling and data dependence analysis.

Instruction scheduling

Another compilation technique for ILP is instruction scheduling. Compilers can analyse the code to identify independent instructions within a sequence. These independent instructions can then be reordered to be executed in parallel on different execution units within the processor. This technique requires careful analysis to ensure proper program behaviour is maintained.

Loop unrolling

This technique duplicates the loop body a certain number of times to expose more potential parallelism within the loop. Consider the following simple C code that adds two numbers 10 times.

```
for (int i = 0; i < 10; i++) {
    result += a + b;
}
```

Without loop unrolling, the processor adds a and b, stores the result, and then repeats the process foreach loop iteration sequentially. Each iteration might involve fetching instructions, reading operands from memory, performing the addition, and storing the result back in memory. Consider a 5-stage pipeline: Fetch (instruction), Decode, Read operands, Execute (addition), and Write result. Without Loop unrolling, the processor goes through the pipeline for each addition: Fetch (1 cycle) -> Decode (1 cycle) -> Read operands (a & b) (1 cycle) -> Execute (add) (1 cycle) -> Write result (1 cycle), resulting to 5 cycles per iteration. As shown in Table 2.3, since there are 10 iterations, the total execution time in a 5-stage pipeline would be 14 cycles.

FOR ONLINE READING ONLY

Table 2.3: *Pipeline without ILP*

	Cycles													
Instruction	1	2	3	4	5	6	7	8	9	10	11	12	13	14
result += a + b	IF	ID	EX	MEM	WB									
result += a + b		IF	ID	EX	MEM	WB								
result += a + b			IF	ID	EX	MEM	WB							
result += a + b				IF	ID	EX	MEM	WB						
result += a + b					IF	ID	EX	MEM						
result += a + b						IF	ID	EX	MEM	WB				
result += a + b							IF	ID	EX	MEM	WB			
result += a + b								IF	ID	EX	MEM	WB		
result += a + b									IF	ID	EX	MEM	WB	
result += a + b										IF	ID	EX	MEM	WB

Loop unrolling can duplicate the loop body, allowing the processor to perform multiple additions concurrently. In our previous example, we unrolled the loop by a factor of 2, creating two independent additions. This means the unrolled loop results in 5 blocks, each with two additions, as shown in Table 2.4.

Table 2.4: *Loop unrolling, a factor of 2*

result += a + b; result += a + b;	Block 1
result += a + b; result += a + b;	Block 2
result += a + b; result += a + b;	Block 3
result += a + b; result += a + b;	Block 4
result += a + b; result += a + b;	Block 5

Modern processors can fetch, decode, and execute 2 of the additions concurrently on different execution units, assuming no data dependencies exist between them. This overlapping of instructions can potentially reduce the total execution time compared to the pipeline without ILP:

Since we unrolled by a factor of 2, two instructions will be issued to the pipeline in each clock cycle as seen in Table 2.5. The total execution time with loop unrolling becomes 9 cycles (assuming no data dependencies and no hazards). This means, that ILP, through the compilation technique of loop unrolling improves our simple program by a factor of: 14/9 = 1.5.

FOR ONLINE READING ONLY

Table 2.5: *Pipeline with Instruction Level Parallelism (ILP)*

	Cycles								
Instruction	1	2	3	4	5	6	7	8	9
result += a + b	IF	ID	EX	MEM	WB				
result += a + b	IF	ID	EX	MEM	WB				
result += a + b		IF	ID	EX	MEM	WB			
result += a + b		IF	ID	EX	MEM	WB			
result += a + b			IF	ID	EX	MEM	WB		
result += a + b			IF	ID	EX	MEM	WB		
result += a + b				IF	ID	EX	MEM	WB	
result += a + b				IF	ID	EX	MEM	WB	
result += a + b					IF	ID	EX	MEM	WB
result += a + b					IF	ID	EX	MEM	WB

Data dependence analysis

It is important to identify data dependencies between instructions. If one instruction relies on the result of another, they cannot be executed simultaneously. Compilers perform data dependence analysis to understand these dependencies and ensure correct execution order even when exploiting parallelism.

Activity 2.4:

Handling data dependencies in ILP

Aim:

To understand and optimise instruction-level parallelism by identifying and mitigating data dependencies, evaluating the impact of forwarding, and comparing performance improvements.

Materials:

Computer with MIPS Simulation software (such as MARS, SPIM, and any other), code segment, Instruction pipeline diagram(such as graph paper, Microsoft visio, Lucidchart, or other)

Instructions:

Step 1: Analyse the code and identify dependencies from the following generated MIPS code, assuming all variables are in memory and are addressable as offsets from $t0.

lw $t1, 0($t0)	//Load b to $t1
lw $t2, 4($t0)	//Load c to $t2
add $t3, $t1,$t2	//$t3 = $t1 + $t2
sw $t3, 16($t0)	//store $t3 to a
lw $t4, 8($t0)	//Load e to $t4
lw $t5, 12($t0)	//Load f to $t5
add $t6, $t4,$t5	//$t6 = $t4 + $t5
sw $t6, 20($t0)	//store $t6 to d

FOR ONLINE READING ONLY

Step 2: Assume no forwarding has been implemented in the pipeline.

(a) Analyse the code and identify dependencies (hint: draw the timing diagram)

(b) Re-order the instructions to facilitate ILP

(c) Assess the performance gain of re-ordering instructions

Step 3: Assume forwarding has been implemented in the pipeline.

(a) Analyse the code and identify dependencies

(b) Re-order the instructions to facilitate ILP

Step 4: Assess the performance gain of forwarding and re-ordering instructions

Deliverables

A comprehensive report comprising of the following:

(a) A concise summary of the key findings from the activity, including the impact of instruction reordering and forwarding on pipeline performance;

(b) Insights and recommendations- Insights gained from the activity and recommendations for handling data dependencies in processor pipelines.

(c) Diagrams and tables: Well-labeled diagrams and tables to illustrate the pipeline stages, timing, and performance comparisons.

(d) Code listings: Clearly presented original and reordered MIPS code segments.

Exercise 2.5

1. Consider the loop unrolling example in the text. Explain why loop unrolling might not always be beneficial. What are some potential drawbacks of loop unrolling?
2. Imagine a loop that iterates 100 times, performing a multiplication (a * b) and storing the result in a variable (c). How much improvement in execution cycles could loop unrolling (factor of 4) achieve, assuming a 5-stage pipeline and perfect pipelining conditions? Explain your calculations.

Scalar versus superscalar pipelining

In scalar pipelining, the processor has a single execution unit for handling instructions offering limited parallelism as only one instruction can be executed at a time in the single execution unit, as seen in Figure 2.10. Superscalar pipelining is a more advanced technique that builds upon scalar pipelining to exploit Instruction Level Parallelism (ILP) within a program. The processor has multiple execution units (typically 2, 4, or even more) that can execute instructions concurrently.

FOR ONLINE READING ONLY

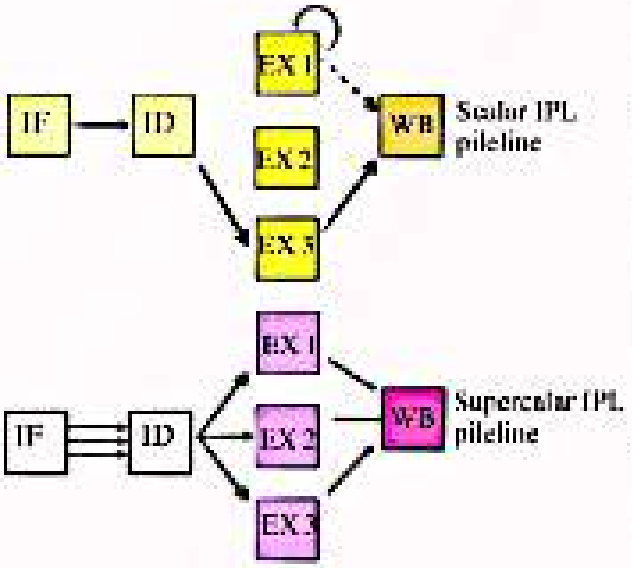


Figure 2.10: *Scalar vs superscalar pipelining*

Imagine a team of chefs preparing a large meal (program execution). ILP focuses on optimising the chefs' workflow (compiler optimisations) and prepping ingredients in advance (loop unrolling) to allow for parallel processing. Additionally, having multiple cutting boards and utensils (hardware features like pipelining and reservation stations) allows them to work on multiple tasks simultaneously, even if the order changes slightly (out-of-order execution). Superscalar execution focuses on having multiple kitchens (execution units) with pre-equipped stations (reservation stations), allowing multiple chefs to work on different dishes concurrently, further enhancing parallel processing.

There are two common categories of superscalars: static superscalars and dynamic superscalars. Static superscalars are also known as static multiple issue processors, and dynamic superscalars are also called dynamic multiple issue processors.

Static multiple issue processors

Traditional processors execute instructions sequentially, one after the other. Multiple issue processors also known as Very Long Instruction Word (VLIW) processors, exploit the concept of ILP to improve performance by exploiting parallelism within the instruction stream. Static multiple issue employ static scheduling which means that the decision of which instructions to execute concurrently is made at compile time by the compiler, not dynamically by the processor during runtime. In a static issue processor, a set of instructions is issued together in one clock cycle. The blocks in Figure 2.7 can be executed in a 2-issue processor as shown in Figure 2.8. The 2-issue processor has two execution units capable of issuing and executing two instructions simultaneously, provided there are no dependencies between them.

Dynamic multiple-issue processors

Dynamic multiple-issue processors are also known as superscalar processors, or simply superscalars. While static multiple issue processors employ static scheduling such that the decision of which instructions to execute concurrently is made at compile time by the compiler, superscalars, employ dynamic scheduling such that the decision of which instructions to execute concurrently is made during execution. In the simplest superscalar processors, instructions issued in order, and the processor decides which instructions can be issued in a

FOR SCHOOL READING ONLY

given clock cycle. Superscalars still require the compiler to try to schedule instructions to move dependencies apart and thereby improve the instruction issue rate. Even with such compiler scheduling, the important difference between superscalars and VLIW processors is that the code, whether scheduled by the compiler or not, is guaranteed by the hardware to execute correctly. In order to facilitate dynamic scheduling in superscalars, the pipeline is divided into three major units: an instruction fetch and issue unit, multiple functional units and a commit unit. Dynamic pipeline scheduling leverages the following key mechanisms;

(a) *In-order issue:* Instructions are fetched and issued to functional units in the order they appear in the program, ensuring program correctness. This is analogous to the chefs receiving instructions sequentially from the head chef.

(b) *Out-of-order execution:* Once issued, instructions can be executed out of order if no data dependencies exist. This allows the processor to exploit parallelism by executing instructions that are ready, even if they arrive later in the stream. Imagine chefs working on independent dishes concurrently, even if they received the recipes at different times.

(c) *In-order commit:* While execution might be out-of-order, the results (written back to memory or registers) are committed in program order to maintain program behavior. This is like ensuring all the dishes are served in the correct sequence, even if they were prepared out of order.

Activity 2.5:

Presentation on the benefits of superscalar execution and ILP in developing imaging applications

You have been selected as part of a team to dekevop an image processing application that involves applying multiple filters to an image. By using reputable materials from the internet and library, prepare a presentation to show how concepts like superscalar execution and ILP will benefit the application's performance.

Exercise 2.6

1. Consider a program with the following instruction sequence:

   Instruction 1: Add A and B, store in C

   Instruction 2: Subtract C from D, store in E

   Instruction 3: Multiply E and F, store in G

   (a) Explain how a scalar pipelined processor would execute this sequence. How many clock cycles would it likely take (assuming no pipeline stalls)?

   (b) How would a 2-issue superscalar processor potentially improve the execution time compared to

FOR ONLINE READING ONLY

the scalar approach? Briefly explain your reasoning.

2. Which of the following statements accurately describes the key difference between a scalar and a superscalar processor?
   (a) Scalar processors can exploit out-of-order execution, while superscalar processors cannot.
   (b) Superscalar processors have multiple execution units, whereas scalar processors have only one.
   (c) Scalar processors are more complex due to dynamic scheduling, unlike superscalar processors.
   (d) Superscalar processors require compiler intervention for parallel execution, while scalar processors do not.

Branch prediction

Branch prediction addresses the challenge of conditional branches, which alter program flow based on conditions. By predicting which path a branch will take, processors enhance performance. For instance, given if (x > 0), goto positive; else, goto negative, the processor must decide which branch to fetch instructions for. Incorrect predictions waste cycles.

There are two primary techniques:

(a) **Static prediction**: The processor consistently predicts a branch as taken or not taken, which is simple but often inaccurate.

(b) **Dynamic prediction**: This method uses the branch's history to predict future behavior, resulting in higher accuracy. Common implementations include Branch Prediction Buffers (BPBs) and Branch History Tables (BHTs).

Branch Prediction Buffers (BPBs) with 1-bit scheme

A Branch Prediction Buffer (BPB) is a small memory indexed by the lower bits of a branch instruction's address, storing a single bit to indicate predicted behavior: 0 for not taken and 1 for taken. For example, a loop branch taken nine times in a row but not taken once would have an 80% prediction accuracy, despite being taken 90% of the time, as shown in Table 2.6.

Table 2.6: *Branch outcomes with a 1-bit prediction scheme*

Iteration	BPB prediction	Actual branch	Outcome update
1	1 (Taken)	Taken	Correct prediction (No update)
2	1 (Taken)	Taken	Correct prediction (No update)
3	1 (Taken)	Taken	Correct prediction (No update)
4	1 (Taken)	Taken	Correct prediction (No update)
5	1 (Taken)	Taken	Correct prediction (No update)

FOR ONLINE READING ONLY

Iteration	BPB prediction	Actual branch	Outcome update
6	1 (Taken)	Taken	Correct Prediction (No update)
7	1 (Taken)	Taken	Correct Prediction (No update)
8	1 (Taken)	Taken	Correct Prediction (No update)
9	1 (Taken)	Not taken	Incorrect Prediction (update BPB to 0 (Not taken))
10	0 (Not taken)	Taken	Incorrect Prediction (update BPB to 1 (Taken))

Prediction accuracy = (Correct predictions / Total predictions) * 100% = 80%. The 1-bit scheme is simpler but less accurate because it struggles with branches that change behaviuor frequently.

Branch History Tables (BHTs) with 2-Bit Scheme

A Branch History Table (BHT) is a small memory indexed by the lower bits of a branch instruction's address, storing multiple bits for more sophisticated and accurate predictions than a BPB. In a 2-bit scheme, as shown in Figure 2.11, the states are: 00 (strongly not taken), 01 (weakly not taken), 10 (weakly taken), and 11 (strongly taken).

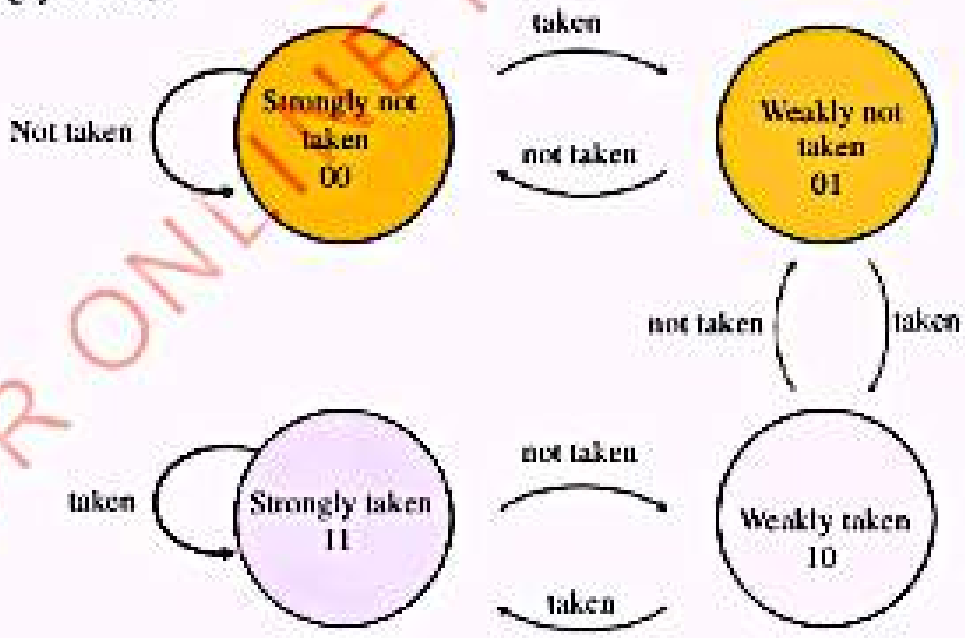


Figure 2.11: *States in a 2-bit prediction scheme*

Consider the previous example with a loop branch that is taken nine times in a row, and then is not taken once. Assume the initial state of the BHT 2-bit predictor is Strongly taken (11). From Table 2.7, we see that a 2-bit scheme improves the prediction accuracy to 90%.

FOR ONLINE READING ONLY

Table 2.7: *Branch outcomes with a 2-bit prediction scheme*

Iteration	BHT prediction	Actual branch	Outcome update
1	11 (Strongly taken)	Taken	Correct prediction (No update)
2	11 (Strongly taken)	Taken	Correct prediction (No update)
3	11 (Strongly taken)	Taken	Correct prediction (No update)
4	11 (Strongly taken)	Taken	Correct prediction (No update)
5	11 (Strongly taken)	Taken	Correct prediction (No update)
6	11 (Strongly taken)	Taken	Correct prediction (No update)
7	11 (Strongly taken)	Taken	Correct prediction (No update)
8	11 (Strongly taken)	Taken	Correct prediction (No update)
9	11 (Strongly taken)	Not taken	Incorrect prediction (Update to 10 (Weakly taken))
10	10 (Weakly taken)	Taken	Correct prediction (Update to 11 (Strongly taken))

Exercise 2.7

1. You are working on a processor design that utilizes dynamic branch prediction. The processor encounters the following loop code:

```
for (int i = 0; i < 10; i++) {
  if (i > 5) {
  // Code for the second half of the loop
  }
}
```

   (a) Analyse the loop behavior and identify how many times the conditional branch ("if (i > 5)") will be taken during a single loop iteration. Explain your answer.

   (b) Which approach would be more suitable for the given loop, and why?

2. 2-Bit BHT Simulation: Simulate on paper the behavior of a 2-bit Branch History Table (BHT) for the provided loop. Assume the initial BHT prediction state is "Strongly Not Taken" (00)

   (a) Create a table with the following columns: Iteration, Actual Branch, BHT Prediction (before the update), and Outcome update (after execution).

   (b) Fill out the table for the first five iterations of the loop.

3. Based on your simulation in question 2, calculate the prediction accuracy of the 2-bit BHT for the first five iterations of the loop.

4. Briefly discuss the impact of branch prediction accuracy on processor performance. How can improved prediction accuracy benefit real-world applications?

FOR ONLINE READING

Register renaming

Modern processors use register renaming to optimises instruction execution by abstracting physical registers from logical registers, increasing instruction-level parallelism (ILP) and improving performance. This technique eliminates bottlenecks caused by multiple instructions depending on the same physical register. By mapping logical registers to a pool of physical registers, processors avoid waiting for register availability. Register renaming eliminates false dependencies, such as:

(a) Write-After-Read (WAR): An instruction reads a register's value, and a subsequent instruction writes to the same register before the value is used, risking outdated data.

(b) Write-After-Write (WAW): Two instructions write to the same register consecutively, where only the second write's value is needed, making the first write redundant.

Traditional processors employ a two-operand instruction format, where the result replaces one of the source operands (e.g., add r1, r2 updates r1 with r1 + r2). Register renaming, however, adopts a three-operand instruction format to support independent source and destination registers. For two-operand instructions, like add r1, r2, a temporary register (e.g., r11) is introduced during renaming, transforming it into a three-operand instruction internally (e.g., r11 ← r1 + r2).

Program Example 2.3:

WAR Without Register Renaming

```
sub r1, r1, r2  // WAR dependency on r1
add r3, r1, r4  // RAW dependency on r1
```

Program Example 2.4:

WAR With Register Renaming

```
temp ← r1 - r2   // Rename r1 to a temporary register (eliminates WAR)
add  r3, temp, r4 // Add r4 to the saved value from r1 and store in r3
```

By using a temporary register (temp) for the destination in the first instruction, the WAR dependency is eliminated. However, as you can see, register renaming does not solve the RAW dependency since 'add' instruction will still have to wait for the correct value of temp to be written before reading and adding it to r4.

FOR ONLINE READING ONLY

Exercise 2.8

1. Imagine you are a computer detective investigating a case of slow program execution. The culprit is inefficient use of registers in the processor! You have identified a segment of instructions that seem to be creating a problem:

   mul r1, r2, r3

   sub r1, r1, 4

   add r4, r1, r5

   (a) Identify the data dependency between these instructions. What type of dependency is it (RAW, WAR, or WAW)?

   (b) Explain why this dependency might create a bottleneck in program execution.

2. How could register renaming be used to eliminate the dependency you identified? Show how the instructions might be rewritten with temporary registers assigned by the renaming process.
3. Explain how register renaming, by eliminating dependencies, helps improve overall program execution speed. Hint: Think about how it allows for more efficient instruction scheduling.
4. Modern processors employ various techniques to optimise performance. Research another technique (e.g., pipelining or out-of-order execution) and explain how it works alongside register renaming for a performance boost.

Thread and data Level parallelism

Scenario 2.3: *Kitchen as a simple processor*

Imagine planning a pizza party with friends involves ordering pizzas, preparing drinks, decorating, and setting up games. It can be overwhelming for one person to manage everything simultaneously. A program acts like a recipe for this party—an instruction set understood by the computer, detailing each step of the planning process. When executed, this becomes a process—a live instance of the program, similar to following a recipe while cooking. Multiple processes can run concurrently, like baking cookies while making pizza dough. Threads within a process represent individual tasks, such as preparing dough and toppings concurrently, as they are lightweight and efficient. Multithreading enables simultaneous handling of multiple tasks in party planning, like decorating while a friend prepares drinks, significantly enhancing efficiency. In contrast, parallelism involves multiple chefs collaborating on different dishes simultaneously, reflecting concurrent execution of various party planning tasks using multithreading.

FOR ONLINE READING ONLY

Questions:

1. Imagine you are the party planner in charge. How would you assign different tasks as separate threads to get things done faster?
2. What potential challenges could arise when using multithreading in party planning? How could you mitigate these challenges?

Pipelining, ILP, superscalars, loop unrolling, and register renaming are used to improve processor performance. However, the problem is that within a single instruction stream we do not find enough independent instructions to execute simultaneously due to many factors such as data dependencies. To overcome this, we can leverage multiple instruction streams by introducing the concept of threads and processes.

Process: An instance of a program in execution, as shown in Figure 2.12. It includes the program code, data structures, and its current execution state (think of it as a recipe being followed while cooking).

Thread: A unit of execution within a process. A process can have multiple threads, each handling a specific task concurrently (imagine multiple cooks working on different parts of the same recipe).

Program: A program is a collection of commands written in a programming language, which a computer executes. These commands outline the actions and tasks the computer needs to carry out.



Figure 2.12: *Processes*

FOR ONLINE READING ONLY

By employing threads and processes, we achieve a form of parallelism where instructions from different streams can execute simultaneously.

Software-based multithreading (Multitasking): The operating system manages multiple processes, allowing them to share CPU resources. It switches rapidly between processes, giving the appearance of simultaneous execution. However, this is technically one instruction stream executed in short bursts across different processes.

Hardware-based multithreading (SMT): This method extends threads to the hardware level. A single CPU core can appear as multiple logical cores to the OS, enabling true parallel execution of instructions from different threads within the same process. If a CPU supports "n" simultaneous threads, the OS views them as "n" processors and distributes workloads accordingly to maximize resource use.

Traditional processors exploit Instruction-Level Parallelism (ILP) by executing multiple instructions from a single thread concurrently. In contrast, Thread-Level Parallelism (TLP) focuses on executing instructions from different threads concurrently. Simultaneous Multithreading (SMT) bridges these approaches, converting TLP into a form of ILP within a single core. Modern processors support SMT with hardware features like out-of-order execution and register renaming, alongside dedicated resources per thread:

(a) Register file per thread: SMT assigns a dedicated register file to each active thread, avoiding conflicts and enabling true parallel execution.

(b) Program Counter (PC) per thread: Each thread maintains its own PC to track its current execution position, facilitating quick thread switching and seamless execution resumption.

Figure 2.13 illustrates three types of hardware multithreading for a single CPU core: coarse-grained, fine-grained, and simultaneous multithreading.

(a) *Coarse-Grained Multithreading (CMT):* Switches between threads only when encountering expensive stalls (such as, waiting for data from memory). This minimizes context switching overhead, the cost of saving the state of one thread and restoring the state of another. However, if a thread encounters a short stall (for example, waiting for a few clock cycles), CMT cannot utilize another thread to keep the processing resources busy. This can lead to wasted cycles in the pipeline.

(b) *Fine-Grained Multithreading (FMT):* Switches between threads after every instruction. This provides the potential for higher parallelism but introduces more context switching overhead.

FOR ONLINE READING ONLY

(c) *Simultaneous Multithreading (SMT):* A form of fine-grained multithreading specifically designed for modern processors. It leverages additional hardware resources within a single core, such as duplicate register files and execution units, to efficiently execute instructions from multiple threads concurrently.

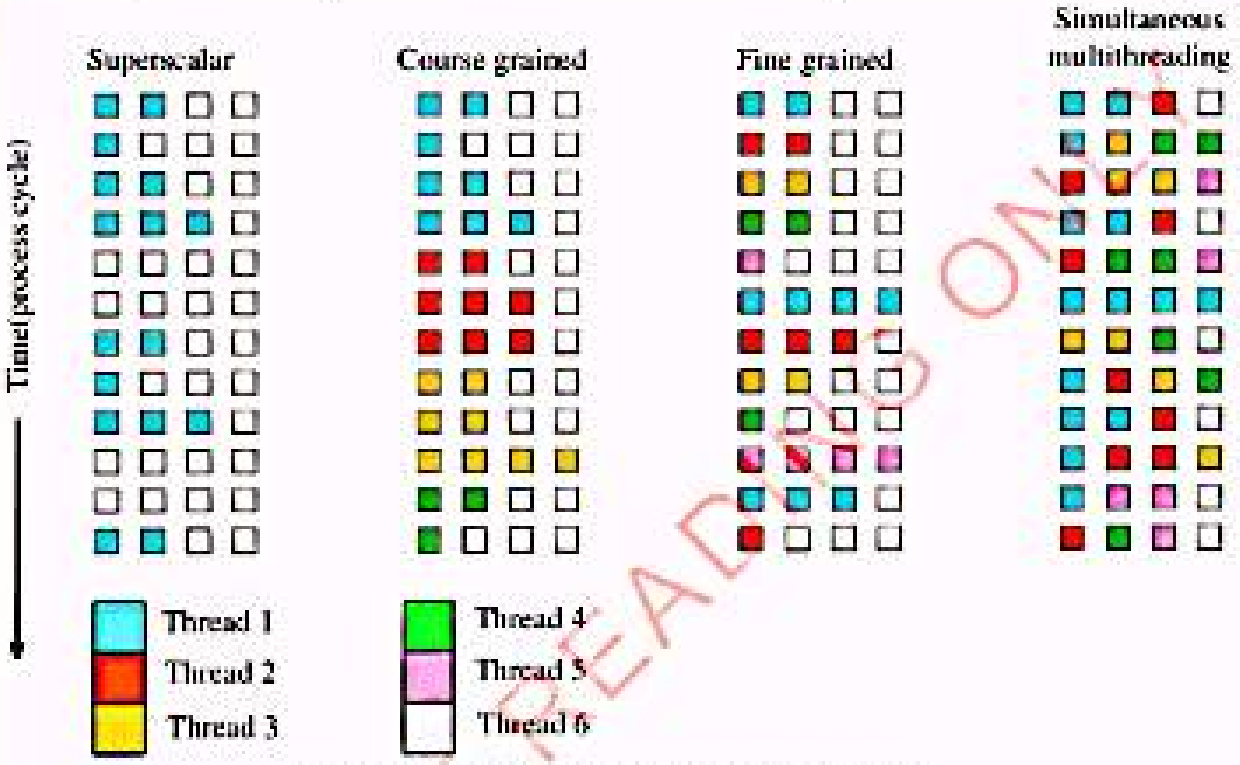


Figure 2.13: *Hardware multithreading categories*

Multithreading offers several advantages. These advantages make multithreading a valuable technique for a wide range of applications, including web servers, multimedia players, game development, and scientific computing.

(a) Multithreading allows multiple threads to execute concurrently hence improving the performance of computer systems.

(b) Multithreaded applications can remain responsive to user interactions even when background tasks are running. This is because threads can be prioritised and scheduled efficiently by the operating system. Imagine downloading a large file while still being able to browse the web smoothly.

(c) Processors often have idle cycles between instructions. Multithreading allows other threads to utilize these idle periods, maximizing the overall utilization of processing resources. This leads to better performance without requiring additional hardware.

(d) Complex applications can be broken down into smaller, more manageable tasks represented by individual threads. This can improve code readability, maintainability, and modularity, making development and debugging easier.

FOR ONLINE READING ONLY

(e) Multithreaded programs can potentially scale well with increasing numbers of processors. By distributing tasks across multiple cores, workloads can be completed faster.

Challenges of multithreading

(a) Register conflicts: Instructions from different threads might use the same register names, requiring techniques like register renaming to avoid data corruption.

(b) Program counter management: Each thread has its own program counter (PC) keeping track of its execution position.

(c) Virtual memory management: The operating system needs to manage memory access for multiple threads within a process and potentially across multiple processes.

Considerations when designing SMT-enabled processors:

(a) SMT works well if threads are well-behaved: There are not too many demanding threads, and they use different processing resources (like doing both math and graphics calculations). Think of it like having a team of specialists who do not all need the same tools at once.

(b) SMT works well if there is harmony in the workload: Threads do not fight over the same resources in the core, allowing for smooth execution. Imagine chefs in a kitchen with separate chopping boards and stoves, minimising bottlenecks.

(c) It is important that in SMT, the OS does not make the mistake of treating all threads equal, even if some come from different cores. This can lead to inefficient workload distribution. Imagine a restaurant manager assigning all tasks to the same waiter, even when others are free.

Examples of SMT architectures

(a) Intel Hyperthreading (HT)

Launched in 2002, Hyperthreading was a revolutionary technology that transformed single-core Pentium 4 processors into logical dual-core processors from the operating system's perspective. Each physical core houses two architectural sets, essentially creating two logical cores for the OS.

These sets have their own:

(i) *General-purpose registers:* Used to store temporary data during program execution.

(ii) *Control registers:* Manage the state of the processor and execution flow.

(iii) *Interrupt control registers:* Handle hardware interrupts and exceptions.

(iv) *Machine state registers:* Track various processor settings and configurations.

(v) *Benefits:* Hyperthreading significantly improved performance for workloads that could benefit from TLP (Thread-Level Parallelism). Applications like web browsing, multitasking, and media processing saw a noticeable boost.

FOR ONLINE READING ONLY

Applications of the Intel Hyperthreading (HT) include;

(a) Web browsers can handle multiple downloads and page rendering concurrently.

(b) Video editing software can encode video while simultaneously previewing the edited content.

(c) Office applications can perform multiple tasks (e.g., spell check, formatting) in parallel.

(b) IBM Power 5 with SMT

Building upon the success of the PowerPC architecture, IBM's Power 5 processor incorporated SMT support in 2004. While offering SMT capabilities, the Power 5 went further with additional improvements:

(a) Increased cache sizes: Larger L1 and L2 cache sizes allowed for more data to be readily available for threads, reducing memory access delays.

(b) Separate instruction prefetch units: Each SMT thread received its own instruction prefetch unit, improving the efficiency of fetching instructions ahead of time.

(c) Increased issue queues: The processor could hold and potentially execute more instructions out of order, further enhancing parallelism.

(d) Larger virtual register File: The number of virtual registers accessible to each thread was expanded, providing more flexibility for complex programs.

The Power 5 with SMT targeted high-performance computing tasks such as scientific simulations, financial modeling, Medical imaging processing and Virtualization environments

While Intel Hyperthreading and IBM Power 5 are prominent examples, SMT is not limited to them. AMD Processors like the Ryzen series also incorporate SMT technology under the name "Simultaneous Multithreading." Many ARM processors for mobile devices and embedded systems utilize SMT for improved performance.

The future of SMT: As processor core counts continue to increase, SMT remains a relevant technology for squeezing additional performance out of individual cores. It offers a balance between adding more physical cores (which can be expensive) and maximizing the utilization of existing resources within a core.

Multithreading vs. Multiprocessing

Multithreading allows a single processor to execute multiple threads concurrently within a single process. Threads share the same memory space but have separate stacks and registers. It is like having multiple cooks (threads) working in the same kitchen (process) using the same ingredients (memory) but with their own tools (registers and stacks).

Multiprocessing utilizes multiple processors to run multiple processes simultaneously. Each process has its own memory space and resources. It's like having multiple kitchens (processes) with separate ingredients (memory) and cooks (processors) working independently.

FOR FREE REFERENCE ONLY

Exercise 2.9

1. Imagine a multithreaded program downloading a large file (Thread A) and performing a complex mathematical calculation (Thread B) concurrently. How might data dependencies between these threads impact their execution? Explain your answer.
2. Imagine you have a program that involves a large number of independent tasks. Discuss the potential advantages and disadvantages of using SMT versus a multi-core processor with separate processes for each task.
3. Modern operating systems employ scheduling algorithms to manage threads and processes. Research different scheduling algorithms used for multithreading and explain how they impact performance and resource utilisation.
4. Imagine a single CPU core with SMT that supports two logical cores. The program has two threads:

```
# Thread 1 (Print Numbers)
    def print_numbers():
     for i in range(1, 6):
       print(i)
# Thread 2 (Calculate Factorial)
        def calculate_factorial(n):
          result = 1
        for i in range(1, n + 1):
          result *= i
        print(f"Factorial of {n} is: {result}")
# Main Program
        start_thread(print_numbers)  # Start thread to print numbers 1-5
        start_thread(calculate_factorial, 5)  # Start thread to calculate facto
        rial of 5

# Wait for threads to finish (pseudocode)
        wait_for_threads()
```

   (a) Explain the expected output of the program. In what order will the numbers be printed (from Thread 1) and the factorial result (from Thread 2) be displayed?
   (b) Imagine running this program with a single thread instead of two. How might the output and execution order differ?
   (c) Considering the scenario above, explain how multithreading can potentially improve the responsiveness of the program, especially for a user interacting with it.
   (d) While this program demonstrates multithreading, can SMT further improve its performance within a single core? Explain your reasoning.

FOR ONLINE READING ONLY

Data Level Parallelism

While TLP focuses on dividing the task itself, DLP focuses on dividing the data. For instance, if the task is sorting candy, DLP would focus on dividing the candy. Each friend could grab a handful of candy and sort them by color independently. In the context of the pizza party, DLP might involve dividing the task of decorating the room further. One friend could hang streamers concurrently while another puts up balloons, effectively utilising multiple "cores" (people) to complete the task faster. Data-Level Parallelism (DLP) is a technique for improving program performance by distributing operations on large datasets across multiple processing units. Instead of focusing on dividing the task itself (like in TLP), DLP emphasises splitting the data into smaller chunks and processing them simultaneously on different cores or processors.

Advantages offered by DLP

(a) *Improved performance:* Dividing the workload amongst multiple processing units can significantly reduce execution time for data-intensive tasks.

(b) *Scalability:* DLP applications can potentially scale well with increasing hardware resources (cores, processors) by further dividing the data for parallel processing.

(c) *Reduced overhead:* Compared to TLP, DLP often incurs less overhead for context switching and maintaining separate thread states.

(d) DLP can contribute to reduced power consumption by potentially enabling faster execution times. By completing tasks quicker, processors can spend less time in active states and more time in low-power modes. However, finding the right balance between parallelism and energy efficiency requires careful optimisation.

Challenges of DLP

(a) *Data Dependencies:* DLP may not be suitable for tasks with high data dependencies, where processing one data element relies on the results of another.

(b) *Granularity:* Finding the optimal data chunk size for efficient processing can be crucial. Too small chunks can lead to increased overhead, while overly large chunks might not utilise all processing units effectively.

(c) *Algorithm design:* Implementing data-level parallelism requires careful algorithm design to ensure proper data division and synchronization between processing units.

Applications of DLP

(a) *Image processing:* Consider an application that needs to apply a filter (Such as, blur and sharpen) to a high-resolution image. DLP allows us to divide the image into smaller tiles. Each core can then independently apply the filter to its assigned tile, significantly accelerating the processing time compared to a single-core approach.

FOR ONLINE READING ONLY

(b) *Scientific simulations:* DLP plays a crucial role in scientific simulations that involve complex calculations on massive datasets. For example, weather simulations can partition weather data for different geographical regions across multiple cores, enabling faster and more accurate modeling.

(c) *Video editing:* Video editing software often utilizes DLP for tasks like encoding or decoding video frames. By distributing the processing of individual frames across cores, the editing process becomes smoother and faster.

(d) *Financial modeling:* Complex financial models involve numerous calculations on extensive financial data. DLP allows us to distribute these calculations across multiple cores, resulting in quicker risk assessments and portfolio optimisations.

The benefits of DLP extend beyond the given examples. Tasks like scientific computing, data mining, and machine learning all rely on efficient data processing, making DLP a cornerstone of high-performance computing. Modern computer systems often utilise parallel memory architectures like DDR (Double Data Rate) memory. These systems allow for reading or writing multiple data elements simultaneously, further enhancing the effectiveness of data-level parallelism.

Multi-core processors

The rise of multicore processors has been a major driver for DLP. With multiple cores available, data can be efficiently distributed and processed concurrently, leading to significant performance gains for compatible applications. Prior to this, processors primarily relied on ILP, which focused on exploiting parallelism within a single instruction stream. However, multicore architectures opened the door for a more granular approach: dividing the data itself.

The multicore revolution was not the only factor driving DLP advancements. The introduction of specialised hardware like Graphics Processing Units (GPUs) opened new avenues for parallel processing. GPUs often contain thousands of cores specifically designed for highly parallel workloads. This has fueled the adoption of DLP techniques including Single Instruction Multiple Data (SIMD), Single Instruction Multiple Thread (SIMT), Data partitioning, Load Balancing and Functional programming.

Data Level Parallelism techniques

(a) SIMD (Single Instruction Multiple Data)

A widely used technique where a single instruction operates on multiple data elements simultaneously. Imagine a group of chefs preparing identical meals for multiple guests. They all follow the same recipe (instruction) but simultaneously operate on individual ingredients (data elements). This is the essence of SIMD.

FOR ONLINE READING ONLY

This parallel execution significantly accelerates computations compared to processing elements one at a time.

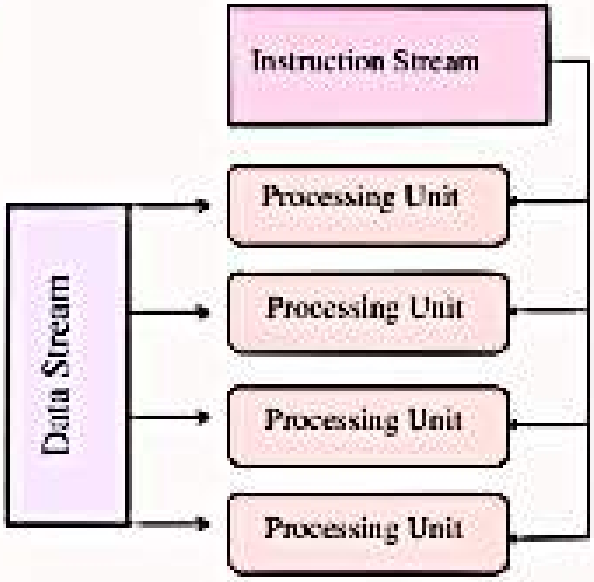


Figure 2. 14: *A 4-lane SIMD architecture*

SIMD supports vectorization

SIMD is mainly dedicated to array processing machines. However, vector processors can also be seen as a part of this group. SIMD provides the hardware capability for vectorisation, which is the process of adapting algorithms to handle data in vectors (arrays) rather than processing single data points one at a time. The data elements reside in special registers called vector registers. SIMD relies on specialised hardware units called vector processing units (VPUs). These VPUs contain multiple processing elements (PEs) that act like mini-processors. When a SIMD instruction is issued, each PE within the VPU executes the same instruction on its corresponding data element in the vector register concurrently. Vector registers can hold an array of data values of the same type, like integers, floating-point numbers, or booleans.

Modern processors often provide special instruction set extensions (ISA Extensions) specifically designed for SIMD operations. These extensions offer a rich set of instructions optimised for vector processing, allowing for efficient manipulation of data elements within a vector register.

SIMD ISA extensions provide instructions for performing arithmetic operations (addition, subtraction, multiplication, etc.), logical operations (AND, OR, NOT, etc.), comparison operations (greater than, less than, equal to, etc.), and data movement operations (loading and storing data from memory to vector registers) on all elements of a vector register simultaneously. Examples of SIMD ISA Extensions:

(a) *SSE (Streaming SIMD Extensions):* A popular SIMD ISA extension for x86 processors offering a wide range of instructions for various data types.

(b) *AVX (Advanced Vector Extensions):* An extension to SSE that provides wider vector registers and additional instructions for improved performance.

(c) *AltiVec:* A SIMD ISA extension for PowerPC processors with similar functionalities to SSE.

FOR ONLINE READING ONLY

(b) SIMT (Single Instruction Multiple Thread)

SIMT (Single Instruction, Multiple Threads) extends SIMD by allowing a single instruction to be applied to different data elements with independent threads of control. SIMT combines SIMD and multithreading.

Imagine chefs preparing identical meals for multiple guests (classic SIMD). Now, each chef leads a small team (thread) in the kitchen. These sub-teams can work on independent tasks (data) while still following the same overall recipe (instruction). This collaborative approach is the essence of SIMT.

Compared to SIMD, SIMT offers greater flexibility by allowing threads to work on independent data, improving the utilization of processing resources, especially for tasks with some data dependencies.

Key characteristics of SIMT

(i) **Single instruction broadcast**: All SIMT threads get the same instruction, like SIMD. However, each thread has its own Stack Pointer, allowing them to work on different data sets even while executing the same instruction.

(ii) **Independent data processing**: Each thread can process its own data from memory using its unique Stack Pointer, providing more flexibility and better handling of data dependencies compared to SIMD.

(iii) **Hardware support**: GPUs (Graphics Processing Units) use SIMT. GPUs have many cores with their own register files, ALUs (both SIMD and scalar), and data caches. Unlike traditional multi-core CPUs, all SIMT cores share a single instruction cache and decoder, making the hardware simpler and more efficient.

SIMT is crucial for General-Purpose Computing on GPUs (GPGPU), that enables GPUs to perform tasks beyond graphics, such as scientific computing and machine learning. Modern supercomputers often combine CPUs with GPUs to take advantage of SIMT for complex simulations and data analysis.

In summary, SIMT enhances SIMD's efficiency by allowing each thread to process its own data, making it a powerful tool for parallel computing and achieving significant performance improvements.

(c) Data partitioning

Data partitioning is a technique in Data-Level Parallelism (DLP) that divides a large dataset into smaller chunks. These chunks can be processed independently by different units for parallel execution. Data partitioning determines what data is processed in parallel by splitting it into independent units for concurrent processing.

Common strategies for data partitioning include:

(i) *Domain decomposition*: Divides data based on logical sections, like splitting customer data by region for parallel processing of regional sales.

FOR ONLINE READING ONLY

(ii) *Spatial decomposition:* Divides data based on spatial properties, like breaking a large image into smaller tiles for parallel image processing.

Data partitioning is used in scientific simulations, big data analytics, and machine learning.

Load balancing

A crucial aspect of data partitioning is ensuring roughly equal work is distributed across processing units. This prevents some cores from staying idle while others are overloaded. Load balancing can adjust task distribution during execution for optimal utilisation of processing units.

(d) Functional programming

While traditional programming languages often focus on changing data over time, functional programming offers a unique approach to DLP. Imagine a team of chefs preparing a banquet. In a traditional approach, each chef might modify the same ingredients (data) repeatedly to create various dishes. Functional programming, however, takes a different perspective of DLP. Functional programming emphasises the creation of immutable data structures. This means data cannot be changed once created, promoting a more predictable and thread-safe environment for parallel processing. Functional programming promotes the use of pure functions. These functions always produce the same output for a given input, regardless of external factors. This characteristic is important for parallel execution, as multiple processing units can independently evaluate pure functions without encountering conflicts. Functional programming can unlock parallelism through recussion and higher-order fuctions.

(a) Functional programming heavily utilizes recursion, where a function calls itself. This can be structured to achieve parallel processing, with each recursive call potentially being handled by a different processing unit.

(b) Higher-order functions are functions that operate on other functions. Higher-order functions offer powerful tools for parallel processing. These functions can be designed to efficiently distribute tasks across multiple cores for processing independent data elements.

Functional programming as a technique to DLP offers several benefits including:

(a) *Simplified reasoning:* Immutability and pure functions lead to simpler program logic, making it easier to reason about potential race conditions or conflicts that can arise during parallel execution.

(b) *Improved concurrency:* The emphasis on immutability and pure functions naturally promotes thread safety, enabling smoother parallel processing without data corruption risks.

(c) *Declarative style:* Functional programming leans towards a declarative style, where the programmer specifies what needs to be done rather than how. This allows

FOR ONLINE READING ONLY

for automatic parallelisation by compilers or runtime systems, leading to efficient utilisation of processing resources.

In conclusion, functional programming offers a unique approach to DLP. By embracing immutability and pure functions, functional programs naturally lend themselves to parallel processing. This style promotes thread safety, simplifies reasoning about concurrent execution, and allows for efficient utilization of multiple processing units in data-intensive applications.

Exercise 2.10

1. Analyse the following applications and explain how SIMD instructions can be beneficial in each scenario:
   (a) Image processing (applying a filter to a high-resolution image)
   (b) Audio processing (encoding/decoding an audio file)
   (c) Scientific computing (performing vectorised calculations on large datasets)
2. Imagine you are working on a program that analyses website traffic data from millions of users. How can data partitioning be advantageous in this scenario? Consider factors such as improved performance, scalability, and efficient resource utilisation.
3. Discuss two common data partitioning strategies, domain decomposition and spatial decomposition, and provide a real-world application example for each.
4. Research and identify two emerging challenges or limitations associated with thread-level parallelism (TLP) in modern processor architectures.
5. Briefly discuss potential solutions or mitigation strategies that computer architects might consider to address emerging TLP challenges.
6. Imagine a program that downloads a large file (Thread A) and performs a complex mathematical calculation (Thread B) concurrently. Analyse two scenarios:
   (a) Scenario 1: No data dependencies exist between Thread A and Thread B.
   (b) Scenario 2: Thread B relies on the completion of Thread A to access specific downloaded data for the calculation.
7. Explain how the presence or absence of data dependencies would impact the execution flow and potential performance benefits of multithreading in these scenarios

Chapter summary

Reflect on what you have learnt from this chapter, summarise and keep it in your portfolio.

FOR ONLINE READING ONLY

Revision Exercise 2

1. For the following code snippet, trace the execution steps (such as fetch, decode, execute, memory access, write back) in both a non-pipelined and a pipelined processor (assuming no hazards). Compare the number of clock cycles it takes to complete the instructions in both scenarios. Explain the speedup achieved through pipelining.

   Code snippet

   lw $t0, 4($s3) # Load value from memory to $t0

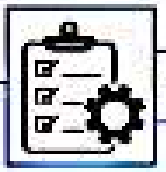
   sub $t1, $t0, $s4 # Subtract $s4 from loaded value to $t1

2. Research the concept of forwarding in pipelined processors. Briefly explain how forwarding can be used to address data hazards and potentially improve performance compared to stalling.
3. Imagine you are a computer architect. How might future processor designs enhance ILP techniques?
4. Research and compare two different architectures (Such as, VLIW and Superscalar) that exploit ILP. Summarise their key differences and advantages in a short presentation.
5. Imagine you are a compiler developer working on optimising a code segment for a modern processor. The code involves multiple loops and calculations. How would you approach this task, considering the concepts of loop unrolling, instruction scheduling, and data dependence analysis? Briefly outline your strategy.
6. Investigate the impact of cache size and memory access patterns on the effectiveness of loop unrolling.
7. Research real-world compiler optimisation techniques related to data dependence analysis (beyond the basic example provided). Explain, in your own words, how data dependence analysis helps compilers exploit ILP effectively.
8. Discuss two potential benefits and two trade-offs associated with using superscalar processors compared to scalar processors.
9. Imagine you are a computer architect tasked with designing the next generation of processors. How could you leverage the concepts of superscalar execution and ILP beyond simply increasing the number of execution units? Briefly describe your innovative approach.
10. Assume the initial BHT prediction is "Strongly taken" (11). Analyse the differences in prediction behavior compared to an initial prediction of "Strongly not taken" scenario.

FOR ONLINE READING ONLY

11. A single CPU core with SMT technology is processing two threads. Thread 1 is performing image editing, which requires frequent memory access. Thread 2 is performing text processing, which is CPU-bound. How might SMT benefit from this scenario?

12. Think about an application you use daily (e.g., web browser, video editor). How might multithreading or SMT contribute to its responsiveness and overall user experience? Explain your reasoning.



Project: *Analyzing website traffic data*

In this project, you will explore how data partitioning can enhance the performance of your program and investigate the challenges and solutions associated with modern processor architectures.

Imagine you are tasked to work on a program that analyses website traffic data from millions of users. As an expert in Computer Science, investigate how data partitioning can enhance the performance of your program and solve this problem. Explore challenges associated with modern processor architectures and suggest the solutions. Document your findings and explanations in a well-organised report and keep it in a portifolio.

FOR ONLINE READING ONLY

Chapter Three

Advanced Computer Networks

Introduction

The modern ways of data communication are highly dependent on the presence of advanced computer networks. Networked computers are able to share or transmit information from one computer to another computer systems located anywhere in the world. This system of networked computers transfers data in the form of signals conveyed through wired media like optical fibres or via wireless transmission such as satellites. In this chapter, you will learn concept of Network communication models, Network models, and Network security principles and protocols in computer networks. Also, you will learn Public switches Telephone Network (PSTN) and Mobile telephone system. The competencies developed will enable you to design and maintain computer networks effectively and productively.

Think:

The complex network of bus routes connecting towns in Tanzania's regions.

Concept of network communication models

Scenario 3.1: *Sending a letter to a friend*

Imagine you want to send a letter to a friend in another region. The entire process involves several steps, each with specific tasks to ensure the letter reaches its destination safely and correctly.

Question:

(a) When you write the letter and choose the language and format, how does this reflect to communication models?
(b) Briefly explain, what do you do before you send the letter to your friends?
(c) Describe the mechanism you undertake in preparing and sending letter to a friend, basing on above scenario.

FOR ONLINE READING ONLY

Open Systems Interconnection Model

The Open Systems Interconnection (OSI) model is a network model that defines the protocols for network communications. The OSI model was developed by the International Organisation for Standardisation (ISO). It was developed to allow systems with different platforms to communicate with each other. The platform could mean hardware, software, or operating system. The model divides similar computer communication protocols into seven layers, each layer providing a specific function to make communication between application processes on various systems easier. Each layer has specific duties to perform and must cooperate with the layers above and below it. Figure 3.1 illustrates the seven layers of the OSI model (top to bottom).

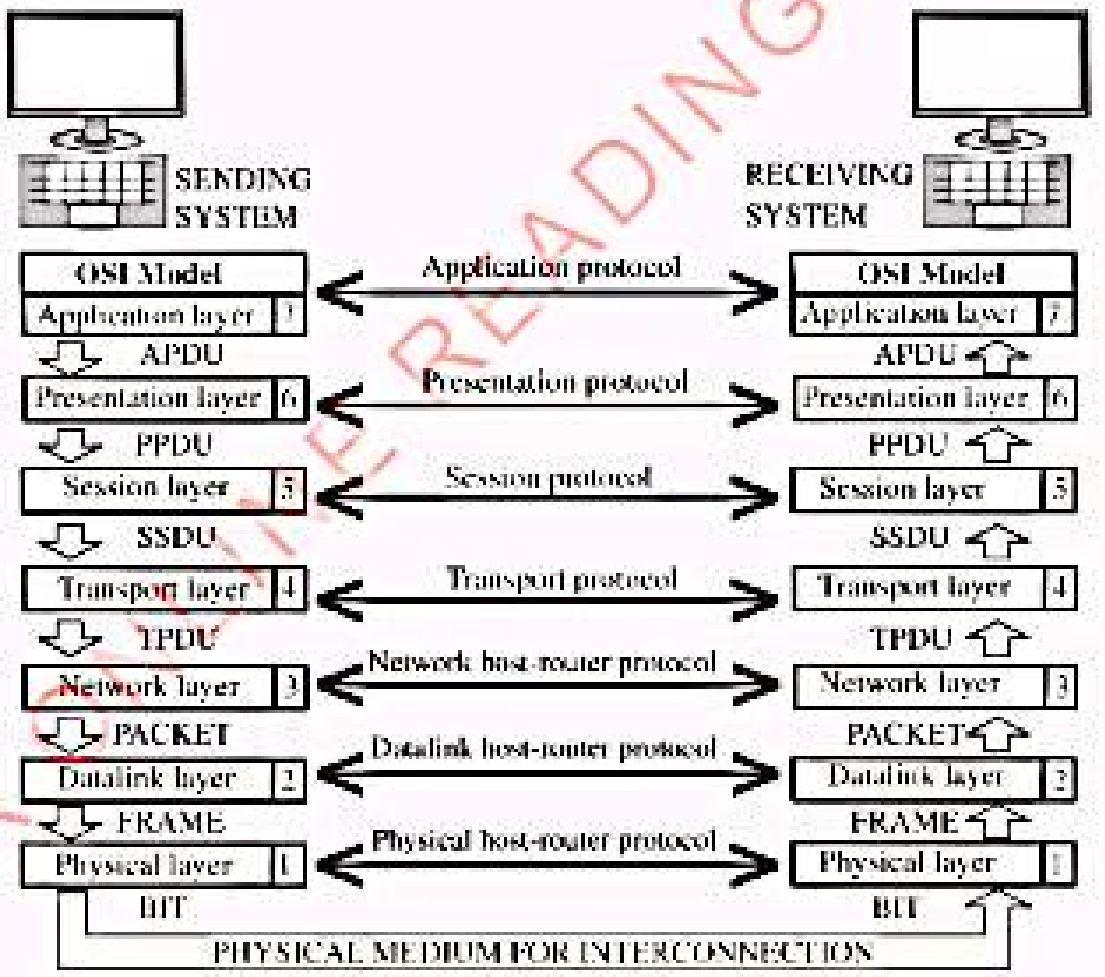


Figure 3.1: *Seven layers of the OSI model*

Advantages of the OSI model

(a) *Conceptual framework:* The OSI model provides a clear and standardised framework for understanding network communication. It breaks down complex network processes into seven distinct layers, making it easier to visualise and troubleshoot network issues.

FOR ONLINE READING ONLY

(b) *Modular design:* Each layer in the OSI model performs specific functions and interacts with adjacent layers through well-defined interfaces. This modular design promotes interoperability between network devices from different vendors as long as they adhere to the same standards at each layer.

(c) *Flexibility:* The layered approach allows for independent development and modification of protocols within each layer without affecting the overall functionality of the model. This flexibility facilitates the introduction of new technologies and protocols without disrupting existing network infrastructure.

(d) *Troubleshooting:* By understanding the functionalities of each layer, network administrators can more effectively isolate and troubleshoot network problems. The layered structure helps pinpoint which layer is likely causing the issue, making troubleshooting more efficient.

Disadvantages of the OSI model

(a) *Theoretical model:* The OSI model is a conceptual framework and does not directly map to any specific real-world network protocol implementations. While it provides a valuable foundation, actual protocols like TCP/IP may have different layer structures or combine the functionalities of multiple OSI layers.

(b) *Overly complex for some applications:* While beneficial for understanding complex networks, the seven layers of the OSI model might appear complicated for simpler network setups.

(c) *Limited scope:* The OSI model primarily focuses on data communication and doesn't encompass broader network functionalities like security, network management, or application services.

Communication between computers in a network in OS model

When information is transmitted from a software application on one computer to an application on another computer, it follows the layers of the OSI model. For example, if computer A's software application needs to send data to computer B's application, the process starts at the application layer (Layer 7) on computer A. The data then goes through the presentation layer (Layer 6), session layer (Layer 5), and so on, until it reaches the physical layer (Layer 1). At this point, the data is transmitted over the physical network medium to computer B.

Upon arrival at computer B, the data moves up from the physical layer (Layer 1) through the data link layer (Layer 2), network layer (Layer 3), and the remaining layers until it reaches the application layer (Layer 7). The application layer on computer B then delivers the information to the intended application program, completing the communication process.

Each layer of the OSI model communicates with its corresponding layer on the other system using control

FOR ONLINE LEARNING ONLY

information. This control information consists of headers and trailers that are attached to the data as it moves through the layers. These headers and trailers contain instructions and requests for the peer layers.

As the data passes through the layers, each layer may add its own header and trailer to the existing data, which already includes headers, trailers, and data from the upper layers. For example, the Transport Header (TH) is only processed by the Transport layer, while lower layers treat it as part of the data, as shown in Figure 3.2. Similarly, at the network layer, an information unit consists of a Layer 3 header (NH) and data. The data link layer treats this as a single unit and adds its own header and trailer.

This process of adding headers and trailers from each layer to the data is known as encapsulation.



Figure 3.2: *Data encapsulation at each layer*

For instance, when computer A needs to send data from a software application to computer B, the data is initially passed to the application layer. In computer A, the application layer adds a header to the data, which contains control information needed by the application layer in computer B. This combined message unit, including the header, data, and possibly a trailer, is then forwarded to the presentation layer. The size of the message unit increases as each layer adds its own header and trailer with control information for its corresponding layer in computer B. When the message unit reaches the physical layer, it is transmitted over the network medium. In computer B, the physical layer receives the message unit and passes it to the data link layer. The data link layer in computer B reads the control information from the header added by the data link layer in computer A. The header and trailer are then removed, and the remaining part of the message unit is forwarded to the network layer.

FOR ONLINE READING ONLY

Each layer follows a similar process: it reads the header and trailer from its peer layer, removes them, and passes the remaining information unit to the next higher layer. After the application layer completes these actions, the data is delivered to the receiving software application in computer B in the same form as it was transmitted by the application in computer A.

Data communication between peer layers

In the OSI model, each layer interacts with other layers to utilise their services. The services provided by adjacent layers assist a particular OSI layer in communicating with its counterpart layer in other computer systems. Typically, a layer in the OSI model communicates with three other layers: the layer directly above it, the layer directly below it, and its peer layer in other networked computer systems. For instance, the data link layer in computer A interacts with the network layer of computer A, the physical layer of computer A, and the data link layer in computer B, as shown in Figure 3.3.

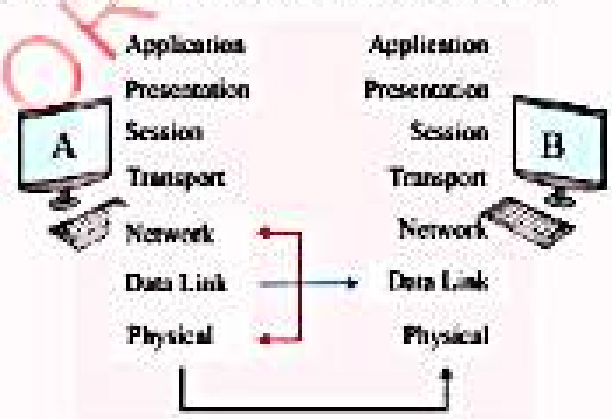


Figure 3.3: *Interactions between network layers*

Transmission Control Protocol/ Internet Protocol model

The Transmission Control Protocol/ Internet Protocol model (TCP/IP) model, often known as the Internet protocol suite, is used on the Internet. It is a set of protocols that allow users to communicate across multiple networks. It always creates an environment in which packets are transferred quickly and efficiently. The OSI model is a broad communication paradigm, whereas the TCP/IP model is the model used by the Internet for all communication. The TCP/IP model provides end-to-end connectivity and specifies how data should be formatted, addressed, transmitted, routed, and received at the destination. These functionalities have been organised into four layers which are known as layers of the TCP/IP model. TCP/IP layers are the Network access, Internet, Transport, and Application layers. Figure 3.4 illustrates the TCP/IP reference model compared to the OSI model.

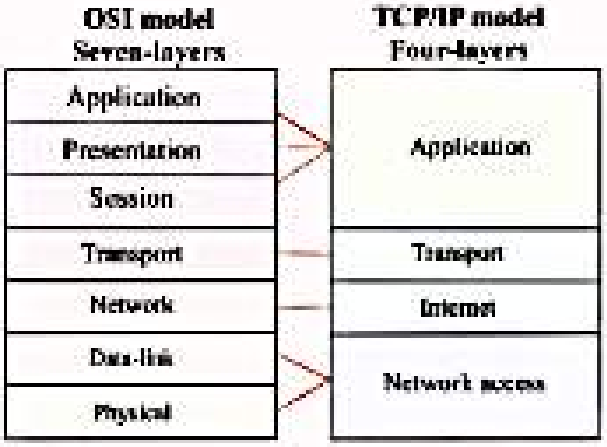


Figure 3.4: *Comparison of TCP/IP model and OSI model*

FOR ONLINE READING ONLY

While the OSI model might appear as the foundational blueprint, the reality is that the *TCP/IP* came first. Developed by the US Department of Defense (DoD), TCP/IP established a simpler approach to network communication. This is evident in the way it combines several layers from the OSI model:

Merged application layer: Layers 5, 6, and 7 of OSI are consolidated into a single *application layer* in TCP/IP.

Network access layer with a twist: Layers 1 and 2 of OSI are combined into a *network access layer* in TCP/IP. However, a key difference exists here, unlike the OSI model, TCP/IP does not handle sequencing and acknowledgment functions within this network access layer. Instead, it delegates these responsibilities to the separate *transport layer*. This revised version highlights the "predating" aspect of TCP/IP and emphasises the key difference in the network access layer's functionalities.

Advantages of the TCP/IP model

(a) *Practical and widely used*: Unlike the OSI model, TCP/IP is a functional model that directly corresponds to real-world network protocol implementations like TCP, UDP, and IP. This makes it a practical tool for network engineers and application developers.

(b) *Simpler structure*: The TCP/IP model uses a four-layer (sometimes five) structure, which is easier to grasp compared to the seven layers of the OSI model. This simplifies learning and understanding network communication fundamentals.

(c) *Flexibility and adaptability*: The TCP/IP model is designed to be flexible and adaptable to accommodate new technologies and protocols. This allows for easier integration of emerging technologies without significant changes to the core model.

(d) Efficient data delivery: The TCP/IP model prioritizes efficient data delivery through protocols like TCP, which offers reliable data transmission with error correction and flow control mechanisms.

Disadvantages of the TCP/IP model

(a) *Limited scope:* Similar to the OSI model, TCP/IP primarily focuses on data communication and doesn't explicitly address aspects like security, network management, or application services.

(b) *Less emphasis on interoperability:* While widely adopted, the TCP/IP model doesn't have the same level of emphasis on strict interoperability compared to the OSI model. This can sometimes lead to compatibility issues between network devices from different vendors.

(c) *Potential for complexity:* Although simpler than the OSI model, the functionalities within some TCP/IP layers, like the Internet layer, can be quite complex, requiring a deeper understanding of advanced network configuration.

FOR ONLINE READING ONLY

(d) *Limited troubleshooting guidance:* Due to its focus on real-world protocols, the TCP/IP model might not provide clear guidance for pinpointing issues within specific layers compared to the OSI model's structured approach.

Functions of each layer in network communication

Activity 3.1:

Exploring OSI Model Layers and Comparing with TCP/IP

Aim:

Understand the functions of OSI model layers and compare them with TCP/IP layers to identify similarities and differences.

Materials:

Internet access or Library resources (optional)

Instructions:

Use an Internet or library search to perform the following:

1. **Explore OSI model layers:**
   (a) Research and understand the distinct functions of each OSI model layer (Physical, Data Link, Network, Transport, Session, Presentation, Application).
   (b) Explore how data is processed and transferred through these layers during communication.
2. **Compare OSI with TCP/IP Layers:**
   (a) Analyse the functionalities of OSI layers and compare them with corresponding layers in the TCP/IP model.
   (b) Identify similarities and differences in terms of protocols, data encapsulation, and network communication handling.

Deliverables:

Write down your findings by outlining the functions of each OSI model layer and Comparative analysis between OSI and TCP/IP layers. Document your work in a portifolio.

Exercise 3.1

1. Give examples of different protocols that devices use on a network?
2. Besides ensuring smooth data exchange, what other advantages do reference models offer in network communication?
3. How do reference models change and incorporate new technologies and network protocols?
4. Provide additional real-world examples where a standardized approach is essential for ensuring communication or compatibility between different systems?

FOR ONLINE READING ONLY

Network models

The concept of network models plays a crucial role in solving the complexities of network interactions. While numerous network models exist, the OSI model is widely recognised for its clear-layered approach. The OSI model offers a structured way to understand data transmission through standardised protocols. The seven OSI models' layers are Application, Presentation, Session, Transport, Network, Datalink and Physical layers. Each layer has specific responsibilities and works together to ensure seamless communication.

Data encapsulation in network communication

When data is sent from one device to another across a network, it goes through a process called encapsulation. This process involves adding protocol information at each layer of the TCP/IP model. Each layer communicates only with its corresponding layer on the receiving device. To facilitate this communication, each layer uses Protocol Data Units (PDUs), which contain control information attached to the data. Typically, the control information is added as a header at the beginning of the data field, but it can also be added as a trailer. At each layer of the TCP/IP model, the PDU is attached to the data through encapsulation, and it has a specific name based on the header information. When the receiving device receives the data, the corresponding layer reads the PDU information. Once read, the control information is removed, and the data is passed up to the next layer.

Figure 3.5 shows how PDUs attach control information at each layer for both the TCP/IP and OSI models. These figures demonstrate how user data from upper layers is prepared for transmission across the network. The data is first passed down to the Transport layer, which establishes a virtual circuit to the receiving device by sending a synchronisation packet. Next, the data stream is divided into smaller segments, and a Transport layer header (PDU) is added to each segment. These segments are numbered to ensure proper reassembly at the receiving device. Each segment is then forwarded to the Network layer, which handles the addressing and routing of the data through the internetwork.

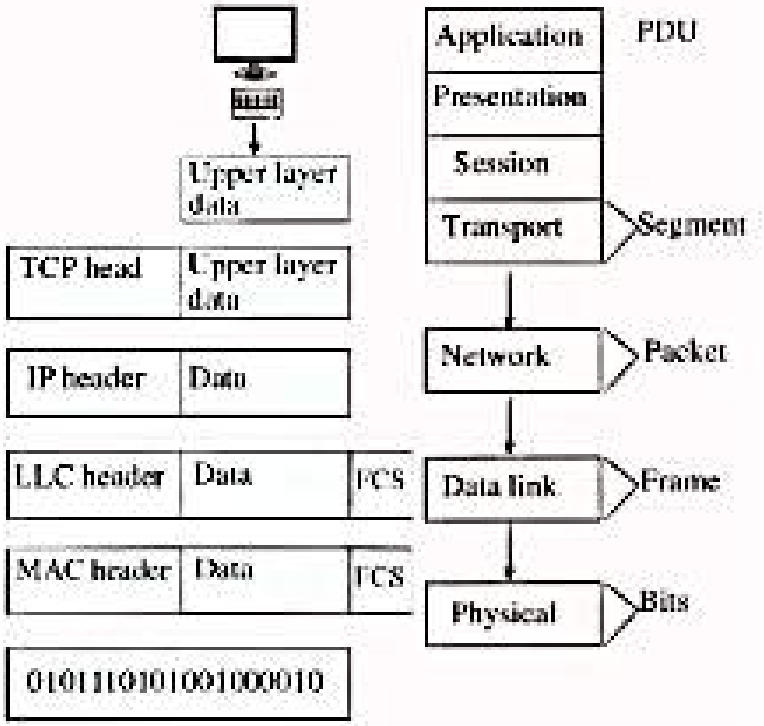


Figure 3.5: PDU attaching control information for OSI and *TCP/IP*

FOR ONLINE REMINDER ONLY

Logical addressing, such as IP addressing, is used to route each segment to the correct network. The Network layer protocol adds a control header to the segment from the Transport layer, turning it into an IP packet or datagram. While the Transport and Network layers work together to reassemble a data stream on the receiving host, they do not handle placing their PDUs on a local network segment for delivery to a router or host. This task falls to the Data Link layer, which receives packets from the Network layer and transmits them onto the network medium, whether cable or wireless. The Data Link layer encapsulates each packet within a frame, whose header includes the hardware addresses of the source and destination hosts. If the destination is on a different network, the frame is sent to a router for further routing. Once on the destination network, a new frame carries the packet to the target host.

To transmit this frame, it must be converted into a digital signal. Since a frame is a logical group of 1s and 0s, the Physical layer encodes these bits into a digital signal interpretable by devices on the same local network. The receiving devices then synchronise with the digital signal and decode the embedded 1s and 0s. At this point, devices reconstruct the frames, perform a Cyclic Redundancy Check (CRC), and compare the result with the frame check sequence (FCS) field. If they match, the packet is extracted from the frame, and the remainder of the frame is discarded in a process called de-encapsulation. The packet is then forwarded to the Network layer, where its address is verified. If the address matches, the segment is extracted from the packet, and the rest is discarded. Figure 3.6 shows PDU and Layer addressing.

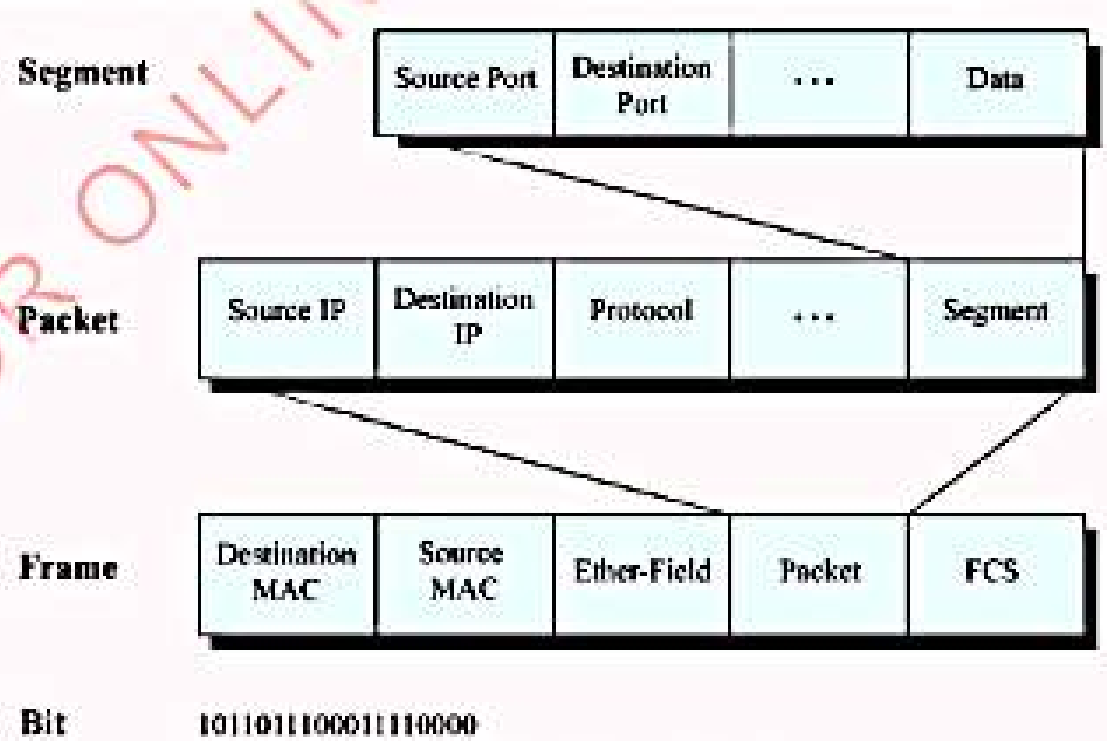


Figure 3.6: *PDU and Layer addressing*

FOR ONLINE READING ONLY

(a) Data encapsulation process

At the transport layer, data is divided into smaller pieces. Each piece receives a header containing information such as the source port, destination port, sequence number, and other relevant details. This process transforms the original data into a new format.

The encapsulated data at the transport layer is known as segments or datagrams. When TCP is utilised, this encapsulated data is specifically referred to as segments. When UDP is used for transmission, this is referred to as Datagrams.

The data then moves to the network layer, where additional information such as the source IP address and destination IP address is added from the layer 3 header. This combination of data and additional information is referred to as packets.

When the packets reach the Data Link layer, additional information, known as a trailer is added. This trailer includes details like MAC address and destination MAC address, and it is also utilised for error-checking purposes. At the Data Link layer, the encapsulated data is termed as frames.

(b) Data de-encapsulation process

The Physical layer receives the bits, de-encapsulates them into frames, and forwards them to the upper layer. At the Data Link layer, frames are received and checked for MAC address matches and errors. If everything is correct, the layer 2 header and trailers are removed, and the data is de-encapsulated before being sent to the upper layer.

The Network layer receives the packet from the Data Link layer, verifies IP addresses, removes the layer 3 header, and delivers the de-encapsulated data packet to the Transport layer.

Finally, the Transport layer removes the layer 4 header before passing the data segments/datagrams to the upper layers (Session, Presentation, and Application) for delivery to the recipient.

This process illustrates the steps involved in data encapsulation and de-encapsulation. Figure 3.7 shows encapsulation and decapsulation in OSI model.

FOR ONLINE READING ONLY

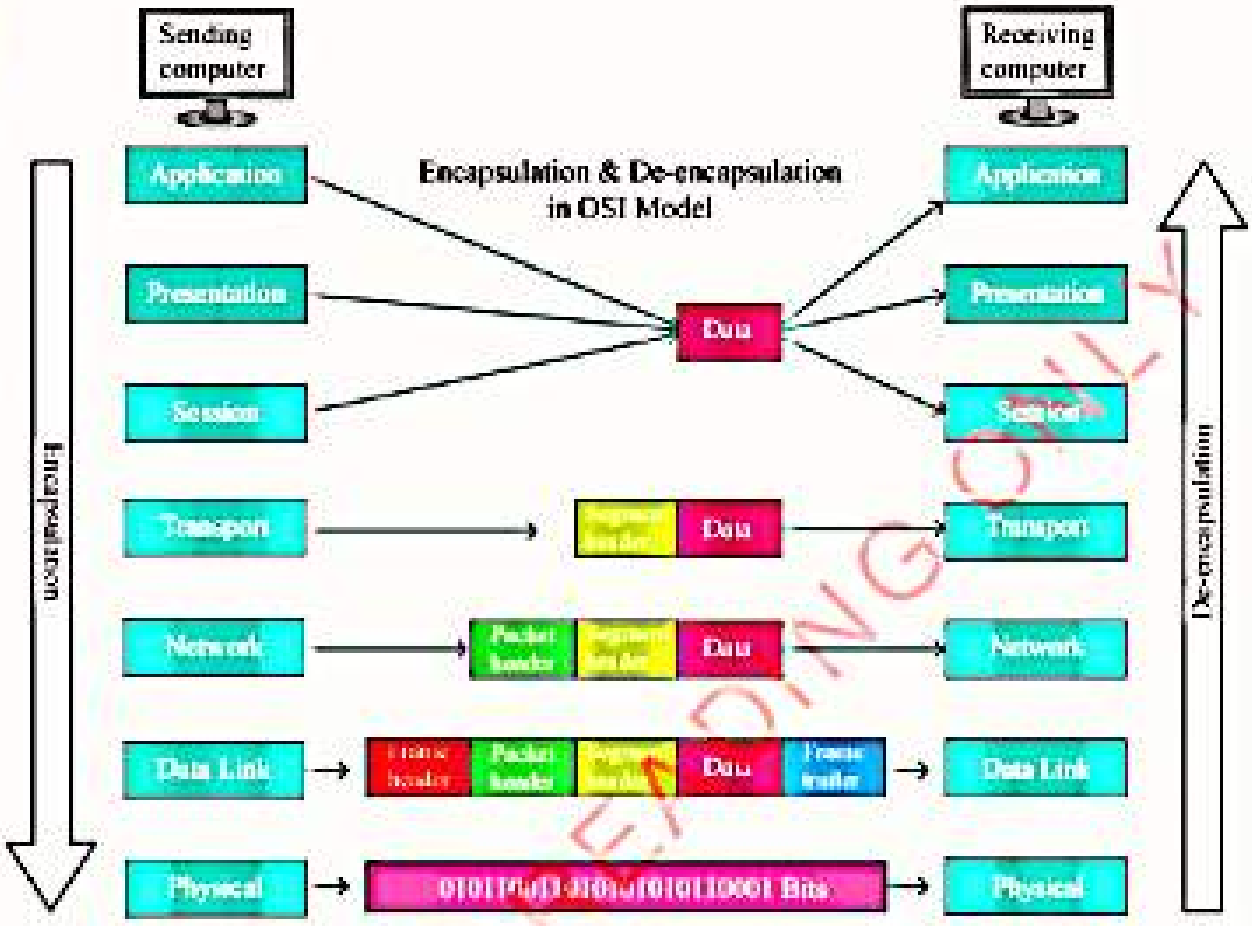


Figure 3.7: *Encapsulation and decapsulation in OSI model*

FOR ONLINE READING ONLY

Exercise 3.2

1. What happens when application data moves from one layer to another in the OSI model as far as OSI models is concerned.
2. Evaluate what is added to the user data when it passes from one layer to another in the OSI model. Give the resulting name for user data at each layer.
3. The OSI model has been around for decades. Despite its enduring value, can you envision any limitations of the model in the context of future network technologies? If so, propose potential adaptations or alternative models that might be necessary.
4. Differentiate between physical communications and virtual communications between OSI layers during data transfer.

Physical layer

The Physical layer of the OSI model is the lowest layer in the hierarchy and is responsible for the transmission and reception of raw data bits. It serves as the interface between the network's logical data and the actual physical medium

through which the data is transmitted. At its core, the Physical layer is concerned with converting the binary data (1s and 0s) into signals that can be transmitted over different types of communication media. These signals can take various forms depending on the medium used, such as electrical voltages, light pulses, or radio waves. The Physical layer ensures that these signals are properly generated, encoded, and transmitted to the receiving device.

The representation of bits in the Physical layer is such that, each bit is assigned a numerical value of either 1 or 0. These binary values form the basic building blocks of digital communication. The Physical layer works with these bits, organising them into frames or packets that can be transmitted over the network.

The Physical layer communicates directly with different types of communication media, including copper wires, fibre optic cables, and wireless channels. Each type of medium has its own unique characteristics, such as bandwidth, data rate, and transmission range. The Physical layer interacts with these media by converting the binary data into appropriate signals that can be transmitted over the specific medium.

To ensure proper communication over different media, specific protocols are needed. These protocols define the appropriate bit patterns, encoding methods, and interface characteristics for each type of medium. For example, Ethernet is a commonly used protocol for wired communication, while Wi-Fi is used for wireless communication. These protocols specify the electrical, optical, or radio signals that represent the binary data and define how devices should interpret and process these signals.

Characteristics of the physical layer

The physical layer encompasses four crucial aspects for establishing, maintaining, and terminating a physical connection between end systems:

1. *Mechanical:* Pertains to the physical attributes of the interface to a transmission medium, usually described as a pluggable connector that connects one or more signal conductors, known as circuits.
2. *Electrical:* Concerns the representation of bits, such as voltage levels, and the bit transmission rate.
3. *Functional:* Defines the functions carried out by individual circuits of the physical interface between a system and the transmission medium.
4. *Procedural:* Details the sequence of events for exchanging bit streams across the physical medium.

The physical layer defines network devices, interfaces between network medium and devices, and defines optical, electrical, and mechanical characteristics.

FOR ONLINE READING ONLY

Activity 3.2:

Exploring of the Physical Layer

Aim:
Understand the physical layer of the OSI model and its role in network communication.

Materials:
1. Computer with Internet access
2. Access to online resources or library

Instructions:
Use Internet search, perform the following:
1. Research the Physical Layer: Use the Internet to explore and analyse the physical layer of the OSI model.
2. Identify Transmission Medium: Determine the type of transmission medium used to connect your computer to the network.
3. Classify Transmission Medium: Identify the broad category of this transmission medium (such as, wired, wireless) and list its advantages and disadvantages.
4. Analyse data transmission: Discuss how this transmission medium sends and receives bits for website viewing on your computer.

Deliverables:
A written report detailing your analysis of the physical layer, the type of transmission medium used, its category, advantages and disadvantages, and an explanation of how it facilitates data transmission.

Finally, document your findings in a portifolio.

Exercise 3.3

1. Discuss the factors affecting signal strength at physical layer
2. What are the main devices used at the physical layer
3. Discuss how the physical layer's design and implementation influence the effieiency and reliability of a network.

Data link layer

While the physical layer only provides a basic service of transmitting raw bits, the data link layer aims to enhance the reliability of the physical link while enabling the activation, maintenance, and deactivation of the link. The data link layer is responsible for the physical transmission of data and performs various tasks such as error notification (error detection) through ACK/NACK, managing network topology (logical topology), and handling flow control to prevent buffer overflow issues. With a fully functional data link layer protocol, the higher layer can assume that transmission over the link is error-free. The data link layer ensures that messages are delivered to the correct device on a local area network (LAN) using hardware addresses. It also translates messages from the network layer into bits for transmission by the physical layer. The data link layer breaks down the message into smaller units called data frames and adds a customized header that includes the hardware destination and source addresses. The data link layer consists

FOR ONLINE READING ONLY

of two sublayers: the Logical Link Control (LLC) and the Media Access Control (MAC) sublayers as shown in Figure 3.8.

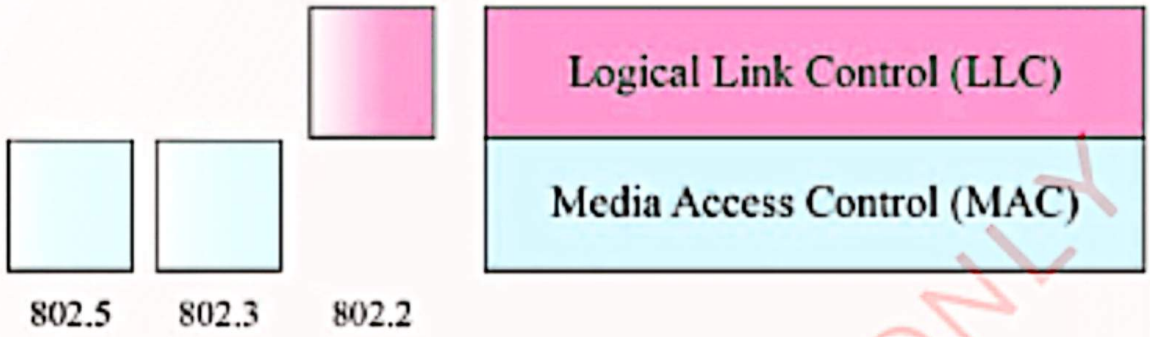


Figure 3.8: *Sub layers of data link layer*

The data link layer consists of two sublayers, which are as follows:

(i) Media Access Control (MAC): This sublayer determines how packets are placed on the media. In contention media access, all devices share the same bandwidth, and access is based on a "first come, first served" basis. The MAC sublayer defines physical addressing and logical topologies. Logical topology refers to the signal path through a physical topology. Additionally, the MAC sublayer can handle line discipline, error notification (not correction), ordered delivery of frames, and optional flow control.

(ii) Logical Link Control (LLC): The LLC sublayer is responsible for identifying Network layer protocols and encapsulating them. An LLC header provides instructions to the data link layer regarding the handling of a packet once a frame is received. For example, the LLC header may indicate that the packet is destined for the IP protocol at the Network layer. The LLC sublayer can also offer flow control and sequencing of control bits.

Switched LANs key concepts

Switched LANs refer to local area networks (LANs) that use switches to connect devices within the network. These switches operate at the data link layer (Layer 2) of the OSI model, and they significantly enhance network performance, efficiency, and manageability compared to traditional LANs that use hubs.

(a) Link-Layer Addressing and ARP

Link-Layer Addressing:

Each device on a local area network (LAN) has a unique MAC (Media Access Control) address, which is used for communication within the same network. This MAC address is a 48-bit identifier usually represented in hexadecimal format (such as, 00:1A:2B:3C:4D:5E).

ARP (Address Resolution Protocol):

ARP is used to map a known IP address to a MAC address. When a device wants to communicate with another device

FOR ONLINE READING ONLY

on the same network, it uses ARP to find out the MAC address associated with the target IP address. The device broadcasts an ARP request packet on the network, and the device with the matching IP address responds with its MAC address.

(b) Ethernet

Ethernet is a widely used technology for wired LANs. It defines wiring and signaling standards for the physical layer and a data frame format and protocols for the data link layer. Ethernet frames consist of a preamble, destination and source MAC addresses, an EtherType field, payload, and frame check sequence (FCS).

The standards that ethernet follows include common Ethernet standards such as 10BASE-T (10 Mbps), Fast Ethernet (100 Mbps), Gigabit Ethernet (1 Gbps), and 10 Gigabit Ethernet (10 Gbps).

(c) Link-Layer Switches

The Link-Layer switches sometimes are known as Ethernet switches. These devices operate at the data link layer (Layer 2) of the OSI model. Switches receive incoming frames on one port and forward them to the appropriate output port based on the destination MAC address. They use MAC address tables to make forwarding decisions, which they build by learning the source addresses of incoming frames. Switches reduce collisions and improve network performance by creating separate collision domains for each connected device. Figure 3.9 shows switch incoming forwarding frame.

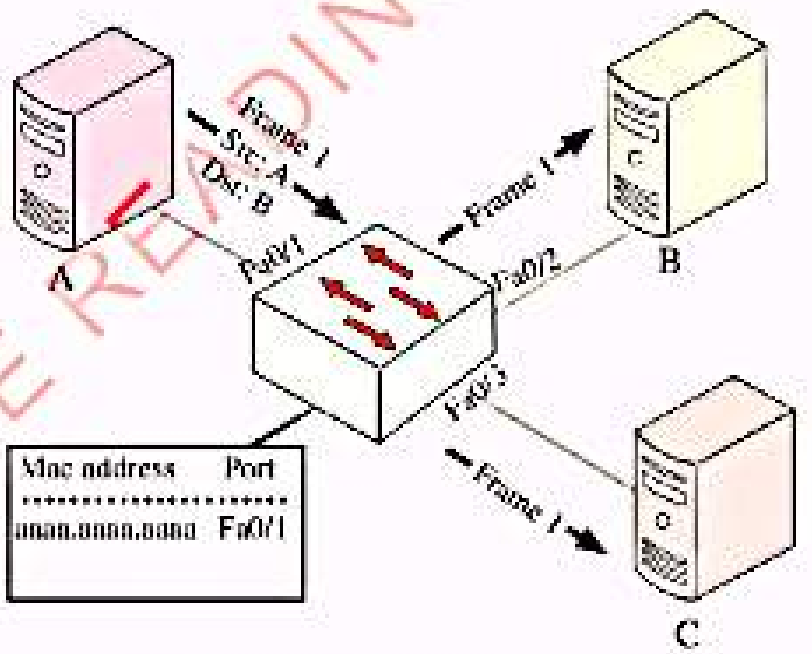


Figure 3.9: *Switch's incoming forwarding frame*

(d) VLANs (Virtual Local Area Networks)

VLANs allow network administrators to segment a physical network into multiple logical networks. Devices on different VLANs cannot directly communicate with each other without a router or Layer 3 switch. VLANs provide improved security, reduce broadcast traffic, and allow for better management of network resources.

FOR ONLINE READING ONLY

Configuring VLANs

VLANs are configured on network switches. Each port on a switch can be assigned to a specific VLAN. VLAN tagging (using the IEEE 802.1Q standard) adds a VLAN identifier to Ethernet frames to indicate the VLAN membership.

Benefits of VLANS

The key benefits of implementing VLANs are as follows:

(i) VLANs enable network administrators to enhance security for network communication.

(ii) They simplify the process of expanding or relocating a network or network devices.

(iii) VLANs provide flexibility by allowing administrators to configure them in a centralised environment, even if the devices are located in different geographical locations.

(iv) By reducing latency and network traffic load, VLANs improve overall network performance.

Disadvantages and limitations of VLANS

(i) VLANs carry the risk of virus propagation, as a single infected system can potentially spread a virus throughout the entire logical network.

(ii) In very large networks, there may be equipment limitations, requiring the addition of routers to manage the increased workload.

(iii) While VLANs can help control latency better than a wide area network (WAN), they are generally less efficient in terms of latency control compared to a local area network (LAN).

(e) Key Characteristics of switched LANs

Switching mechanism

(i) Layer 2 switching: Switches forward Ethernet frames based on MAC addresses. They maintain a MAC address table that maps each MAC address to the corresponding switch port. This allows them to deliver frames only to the intended recipient, reducing unnecessary traffic.

(ii) Layer 3 switching: Advanced switches can also perform routing functions, handle IP addresses, and enable inter-VLAN communication.

Reduced collisions

Unlike hubs that broadcast incoming data to all ports, switches send data only to the port where the destination device is connected. This segmentation of the network into smaller collision domains reduces collisions and increases efficiency.

Full-duplex communication

Switches support full-duplex communication, allowing devices to send and receive data simultaneously, effectively doubling the network's bandwidth.

FOR ONLINE READING ONLY

MAC address table

Switches build a MAC address table by learning the source MAC addresses of incoming frames and associating them with the corresponding ports. This dynamic learning process enables efficient frame forwarding.

(f) Components of a switched LAN:

Switches: Switches are the central devices in a switched LAN, connecting end devices (such as computers, printers, and servers) as well as other network devices (like routers and firewalls).

End devices: End devices refer to computers, servers, printers, and other devices that connect to the network through switches.

Cabling: Cabling involves the use of Ethernet cables, such as Cat5e or Cat6, to establish connections between devices and switches.

Network Interface Cards (NICs): NICs are hardware components present in each device, providing the physical interface for connecting to the network.

Router (for inter-VLAN communication): Routers are devices that enable communication between multiple VLANs by routing traffic between them. In some cases, routers are integrated into Layer 3 switches.

Real environment of the use of a switched LAN is when the office have several departments such as HR, Sales, Accounts, Procurement, and IT. Each department may have its own set of devices and requires its own logical network for security and traffic management purposes. Using switched LANs with VLANs, the network administrators can perform the following:

(i) *Create separate VLANs:* for example VLAN 5 for Sales, VLAN 10 for Accounting, and VLAN 15 for IT. Also they can assign switch ports to the appropriate VLANs based on the devices connected.

(ii) *Isolate traffic*: for example traffic from accounting devices stays within VLAN 10, sales traffic stays within VLAN 5, and IT traffic stays within VLAN 15. The purpose of this is reduce unnecessary broadcast traffic and improves security by isolating departments.

(iii) *Enable inter-VLAN communication:* Enabling inter-VLAN communication through the use of a Layer 3 switch or router allows for the necessary communication between departments while still preserving the segmentation of VLANs.

Activity 3.3:

Exploring of the Data Link Layer

Aim:

Understand how the data link layer facilitates hop-to-hop delivery of frames and its role in network communication.

FOR ONLINE READING ONLY

Materials:

Computer with Internet access; Access to online resources; and Network simulation software (Such as Cisco Packet Tracer, GNS3, or Wireshark)

Instructions:

Step 1: Explore hop-to-hop delivery

(a) Research how the data link layer enables hop-to-hop delivery of frames that encapsulate your request for website viewing. Trace the path from your computer (first hop) to the layer 2 or 3 switch (second hop) and finally to the router (third hop), ready to exit your school's network infrastructure.

(b) Use network simulation software to create a simple network topology. Simulate sending a request from your computer to a web server. Observe how the data is encapsulated into frames and travels through each hop.

Step 2: Analyse incoming packages

(a) Investigate how incoming packages (web page responses) arrive at the router, pass through the switch, and eventually reach your computer for website viewing.

(b) Use network simulation software to simulate the return path of the web page response. Observe the de-encapsulation process at each hop.

Step 3: Examine Data Link layer protocols

(a) Study how protocols at the data link layer cooperate to facilitate the delivery of web content to your computer from your local area network.

(b) Use tools like Wireshark to capture and analyse real network traffic. Identify data link layer protocols such as Ethernet and examine the frames for source and destination MAC addresses, and other protocol-specific details.

Step 4: Document your findings

Write down your findings and document them in a portfolio. Include screenshots from the simulations and analysis from Wireshark to support your observations.

Deliverables:

(a) A detailed report documenting your exploration of the data link layer's role in hop-to-hop delivery, the path of web page responses from the router to your computer, and the cooperation of data link layer protocols in facilitating content delivery.

(b) Screenshots and descriptions of the network simulations and traffic analysis using Wireshark.

Exercise 3.4

1. How does the Data Link layer handle errors that occur during data transmission, and what are the advantages and disadvantages of these error detection and correction methods?

FOR ONLINE READING ONLY

2. Evaluate the impact of different Data Link layer protocols on network performance. How do protocols like Ethernet and Wi-Fi manage access to the shared communication medium, and which is more efficient in various scenarios?
3. Explain why is it necessary for organisation to use VLANs?

Network layer

The Network layer, also known as layer 3 in the OSI model, handles device addressing, tracks device locations on the network, and determines the most efficient way to transport data. It is responsible for facilitating communication between devices that are not directly connected. In cases where there is a direct point-to-point link between stations, the Network layer may not be necessary as the data link layer can handle the link management. Routers, which are layer 3 devices, are specifically designed for the Network layer and provide routing services within the internetwork. The Network layer divides messages into data packets and adds a customized header containing the IP destination and source addresses. There are two types of packets used at the Network layer: data packets and route update packets. Data packets are used to transport user data across the internetwork. The protocols that support data traffic are known as routed protocols, with examples including IP and IPX. On the other hand, route update packets are used to inform neighbouring routers about the networks connected to all routers within the internetwork. The protocols that send route update packets are called routing protocols, such as RIP, EIGRP, and OSPF. Route update packets assist in building and maintaining routing tables on each router.

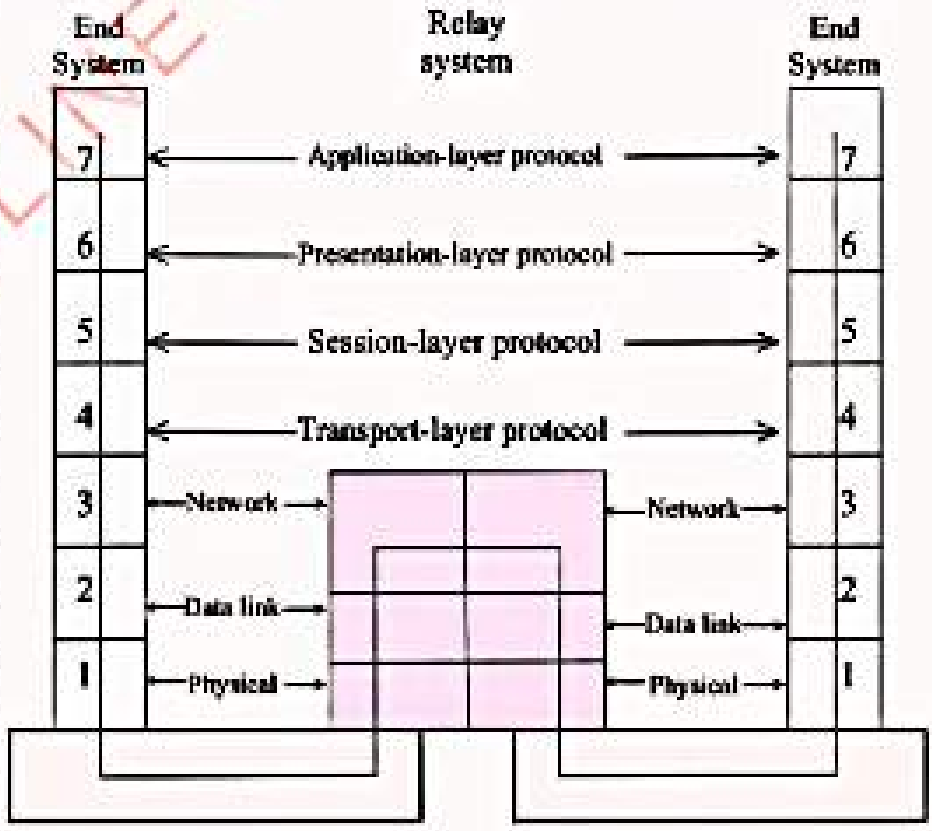


Figure 3.10: *The network layer (to be redrawn)*

FOR ONLINE READING ONLY

The bottom three layers of the OSI model focus on connecting to and communicating with the network, as seen in Figure 3.10 When an end system generates packets, these packets traverse through network nodes, which serve as relays between the two end systems. These network nodes implement layers 1-3 of the OSI architecture. The network layer is responsible for managing the transfer of data between network devices. It determines the routing of packets based on the unique network device addresses. Additionally, it provides flow and congestion control mechanisms to prevent depletion of network resources.

Activity 3.4:

Exploring the Network layer

Aim:

Understand the role and functionality of the network layer in computer networking.

Materials:

Computer with Internet access or Network simulation software (Such as Cisco Packet Tracer, GNS3, NetSim, or any other preferred tool)

Instructions:

1. *Designing a network for a new office*
   (a) Imagine you need to design a network for a new office building.
   (b) Plan how you will set up the network layer to support both wired and wireless devices.
   (c) Ensure your network can grow in the future (scalability).
   (d) Think about how to keep the network secure.
   (e) Consider your budget and how to keep costs low.
   (f) Document your network plan in detail.
2. *Handling a router failure*
   (a) Imagine a router in your network stops working, causing communication problems.
   (b) Think of different ways to reroute the network traffic to fix the issue quickly.
   (c) Make sure your solutions keep the network secure and data safe.
   (d) Write down your ideas for alternative routing solutions.
3. *Document your findings:* Write a report or create a portfolio that includes your network design plan, and your solutions for handling a router failure.

Deliverables:

A written report. Document your findings and solutions in a portfolio.

FOR ONLINE READING ONLY

Network layer design issues

Network layer involves several design issues that need to be addressed to ensure efficient and reliable data transmission across networks. Table 3.1 describe the key consideration for the network layer design.

Table 3.1: *Key design consideration for the network layer*

S/n	Key issues to consider	Action	Description
1.	Routing	Finding the Best path	The network layer needs to determine the optimal route for data packets to travel from source to destination. This involves choosing the path with the least congestion, delay, or cost. Routing protocols like Open Shortest Path First (OSPF) and Border Gateway Protocol (BGP) are used to achieve this.
2.	Addressing	Unique identification	Every device on a network needs a unique identifier, like an IP address, to allow for proper routing of data packets. The network layer ensures that these addresses are assigned and managed correctly.
3.	Packet switching	Dividing and conquering	The network layer often breaks down large data files into smaller packets for efficient transmission. It's responsible for adding header information to these packets, including source and destination addresses, and for reassembling them at the receiving end.
4.	Error handling	Ensuring data integrity	Errors can occur during data transmission. The network layer might implement error detection and correction mechanisms to ensure data arrives intact. This might involve adding checksums to packets and requesting retransmission of corrupted data.
5.	Congestion control	Avoiding gridlock	Networks can become congested when too much data tries to flow through them. The network layer might employ techniques like congestion avoidance algorithms to regulate traffic flow and prevent network overload.

FOR ONLINE PERSONAL USE ONLY

S/n	Key issues to consider	Action	Description
6.	Security	Protecting data	Security threats like eavesdropping and data manipulation exist on networks. The network layer might be involved in implementing basic security measures to protect data in transit, such as access control lists (ACLs) or encryption (although higher-level layers typically handle more robust security).
7.	Interoperability	Speaking the same language	Networks often consist of devices from different vendors using various protocols. The network layer plays a role in ensuring these devices can communicate seamlessly by adhering to standardised protocols like TCP/IP

Network layer routing

The network layer plays a crucial role in getting data packets from their source to their intended destination across a network. This layer relies on a special tool called a *routing algorithm* to determine the optimal path for packet transmission. Imagine sending a package across town. You would not just pick any random route; you would likely use a map or GPS to find the fastest or most efficient path. Similarly, routing algorithms act as a guide for network devices, helping them choose the best path (least-cost path) to forward data packets to their destination. The network layer can offer two main service types:

Datagram service: Packets are treated independently, like sending individual letters. Each packet finds its way to the destination using the routing algorithm.

Virtual circuit service: This establishes a pre-defined connection between source and destination, similar to a phone call. Packets follow this established path for more reliable delivery.

Power of routing protocols

Routing algorithms are implemented through software programs called *routing protocols*. These protocols constantly exchange information about network conditions (traffic, congestion) with other devices. This allows them to dynamically calculate the best paths for data packets, ensuring efficient and reliable data delivery.

Categories of routing algorithms

There are three main categories of routing algorithms namely, adaptive routing algorithms, non-adaptive routing algorithms, and hybrid routing algorithms.

FOR ONLINE READING ONLY

Adaptive routing algorithms

These algorithms are like intelligent navigators. They can dynamically adjust to changing network conditions (traffic congestion, link failures) by continuously collecting information and updating routing tables. This allows them to find the most efficient paths for data packets in real-time. They continuously analyse network conditions like topology (network layout) and traffic congestion to make informed routing decisions. This allows them to find the most efficient paths for data packets in real-time, adapting to changes in the network.

Factors for decision-making

Adaptive algorithms consider several factors when choosing a path

(a) *Hop count:* The number of "hops" (network devices) a packet needs to traverse to reach its destination. A lower hop count often indicates a faster path.

(b) *Distance:* The physical distance between network devices, which can be relevant for some routing protocols.

(c) *Estimated transit time*: An estimate of how long it might take for a packet to travel along a particular path.

Types of routing algorithm

There are three main categories of adaptive routing algorithms (see Figure 3.11).

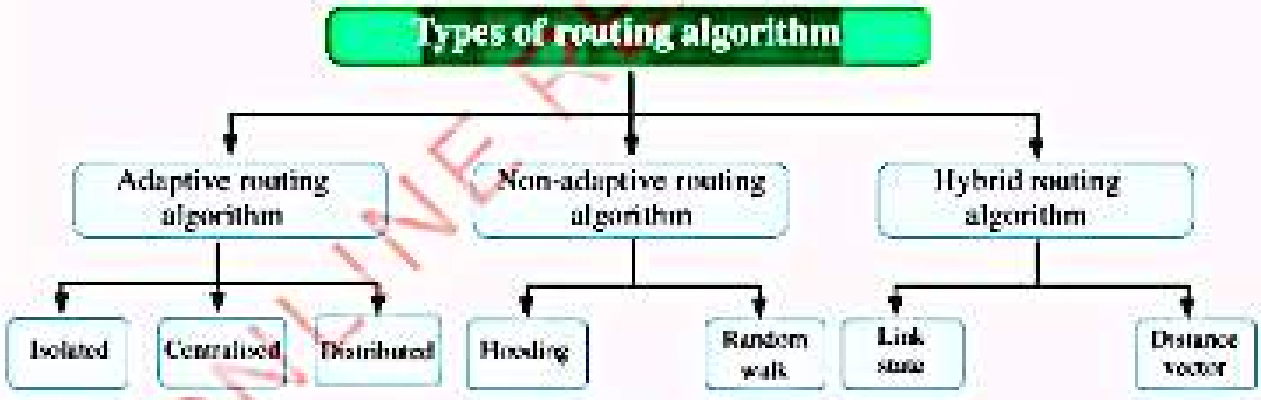


Figure 3.11: *Types routing algorithms*

Adaptive routing algorithm

Centralised routing

In a real real-world scenario, imagine a central traffic control center with a comprehensive view of the entire road network. This approach is similar: A central entity has complete knowledge of the network topology and link costs such as congestion levels. Using this global information, it calculates the optimal path for each data packet.

Isolated routing

This is like navigating using only local road signs without consulting a map or GPS. In this approach, each network device makes routing decisions based on information about its directly connected neighbors, without relying on updates from other devices.

FOR ONLINE READING ONLY

Distributed routing

Imagine multiple traffic control centers communicating with each other to share local traffic information. Here, network devices exchange information about their neighbors and the costs of reaching different destinations. Each device then iteratively calculates the best path based on the collected information, building a network-wide picture without a central authority.

Non-adaptive routing algorithms

These algorithms rely on pre-defined routing tables that remain static. They are simpler to implement but may not be as efficient in dynamic network environments. Imagine a fixed travel route you plan, without considering real-time traffic conditions. They rely on a static routing table programmed into routers during network setup. This table defines the path each packet should take to reach its destination, regardless of current network conditions. Non-adaptive algorithms are relatively simple to configure and manage. However, they cannot adapt to changes in network traffic or topology (layout changes, link failures). This can lead to inefficient routing if optimal paths become congested or unavailable.

Types of non-adaptive routing

There are two main types of non-adaptive routing algorithms:

(a) *Flooding:* Imagine sending a lost person a copy of a map with every street highlighted. Flooding sends every incoming data packet to all connected network devices except the one from which it arrived. While this ensures the packet reaches its destination eventually, it can be inefficient and create unnecessary network traffic. Multiple copies of the same packet can circulate, potentially overwhelming network resources.

(b) *Random walks:* Imagine taking random turns at each intersection until you reach your destination. In random walks, a packet is sent to one of its neighboring devices randomly. While this approach can explore alternative routes, it lacks any direction and may not be the most efficient way to reach the target.

Differences between adaptive and non-adaptive routing algorithms

When it comes to moving data across a network, routing algorithms play a crucial role in choosing the best path for data packets. Use Activity 3.5 to explore the differences between adaptive and non-adaptive routing algorithms.

Activity 3.5:

Exploring adaptive and non-adaptive routing algorithms

Aim:

To understand the differences between adaptive and non-adaptive routing algorithms through research and practical analysis.

Materials:

1. Computer with Internet access or Library access
2. Network simulation software (Such

FOR PREVIEW ONLY

as Cisco Packet Tracer, GNS3, or similar)

3. Text editor or word processor for documenting findings

Instructions

Step 1: Research adaptive and non-adaptive routing algorithms

(a) Use the Internet or library resources to explore the concepts of adaptive and non-adaptive routing algorithms.

(b) Focus on understanding how each type functions, their advantages, and disadvantages.

(c) Identify real-world examples of each type of algorithm and their typical use cases.

Step 2: Compare adaptive and non-adaptive routing algorithms

(a) Create a table or chart that compares the key features of adaptive and non-adaptive routing algorithms.

(b) Include factors such as response to network changes, complexity, efficiency, and scalability.

Step 3: Simulate routing algorithms

(a) Use network simulation software to set up a simple network topology.

(b) Implement and observe the behavior of an adaptive routing algorithm (e.g., OSPF or EIGRP) and a non-adaptive routing algorithm (e.g., static routing).

(c) Document the network setup process and the outcomes of each simulation.

Step 4: Analyse and document findings

(a) Write a detailed report comparing the performance and behavior of adaptive and non-adaptive routing algorithms based on your research and simulation results.

(b) Discuss scenarios where one type of routing algorithm might be more beneficial than the other.

(c) Provide examples of network environments where adaptive or non-adaptive routing algorithms are commonly used.

Deliverables:

(a) Comparative table/chart of the key features of both types of algorithms

(b) Network simulation setup and results

(c) A detailed report analyzing the differences, performance, and use cases of adaptive and non-adaptive routing algorithms

Hybrid routing algorithms

Hybrid routing algorithms blend aspects of both adaptive and non-adaptive routing methods to achieve a balance between efficiency and reliability.

Characteristics

(a) *Balanced approach*: They combine the stability of fixed routes with the flexibility of dynamic routes, adapting to network changes

FOR ONLINE READING ONLY

while maintaining a degree of predictability.

(b) *Efficient performance:* These algorithms aim for a sweet spot between fast convergence (adapting quickly to changes) and optimal resource utilization (minimizing bandwidth and processing overhead).

(c) *Scalability:* Hybrid routing algorithms are often well-suited for larger, more complex networks due to their ability to balance adaptability with stability.

Example

Enhanced Interior Gateway Routing Protocol (EIGRP): A prime example of a hybrid routing algorithm. EIGRP combines elements of distance-vector and link-state protocols. This allows for relatively fast convergence times like distance-vector protocols while offering some of the advantages of link-state protocols, making it relatively easy to use.

Advantages

(a) *Stability and adaptability:* Hybrid algorithms provide a good balance between these two desirable characteristics, adapting to changes without being overly sensitive to network fluctuations.

(b) *Enhanced performance and efficiency:* By combining the strengths of different routing approaches, hybrid algorithms can achieve better overall routing performance and more efficient use of network resources.

Disadvantages

(a) *Complexity:* Hybrid algorithms can be more complex to configure and maintain compared to purely adaptive or non-adaptive methods.

(b) *Resource requirements:* The increased complexity may demand additional resources for processing and management.

Routing Protocols

Imagine a large company with offices in different cities. Each office has its own network, but employees need to share information across locations. Routing protocols act like GPS for data packets, as shown in Figure 3.12. This includes:

(a) ***Interior Gateway Protocols (IGP):*** Used within a single organization's network, like local maps. Examples are RIP and OSPF.

(b) ***Exterior Gateway Protocol (EGP):*** Used for communication between different organizations, like connecting a company's network to the Internet.

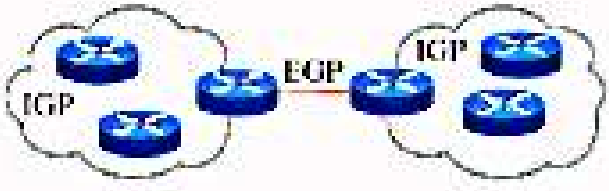


Figure 3.12: *Gateway protocols*

Address Resolution Protocol (ARP)

Sending a message to a friend at a party requires knowing their location. Similarly, ARP translates IP addresses to MAC addresses for direct network communication, as shown in Figure 3.13. For example:

(a) Device A with IP address 2.2.2.2 wants to send data to device B with IP address 4.4.4.4.

(b) Device A sends an ARP request for device B's MAC address.

(c) Device B responds with its MAC address.

(d) Device A sends data directly to device B.

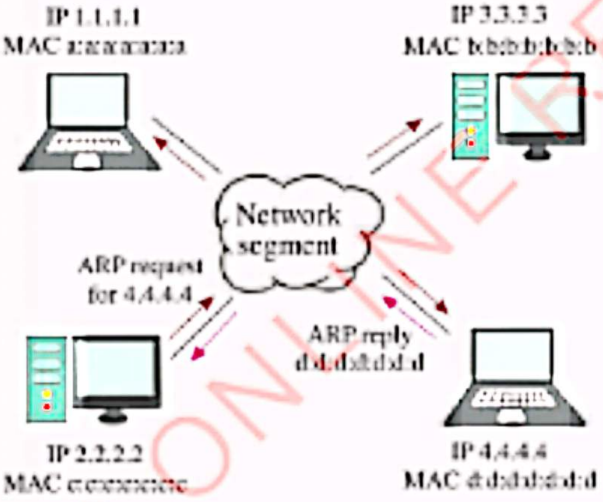


Figure 3.13: *Network segment*

Internet Control Message Protocol (ICMP)

ICMP monitors network issues and reports them back. It uses IP packets for transportation and includes tools like:

ICMP Mechanisms

ICMP messages are encapsulated within IP packets. Some of the most common ICMP, include:

(a) ICMP echo request (Ping)

Purpose: To check if a destination device is reachable.

Process: A source device sends an ICMP Echo Request message to the target device. If the target device is reachable and operational, it responds with an ICMP Echo Reply message.

(b) ICMP echo reply (Pong)

Purpose: To confirm that the target device is reachable and responsive.

Process: Upon receiving an ICMP Echo Request, the target device sends an ICMP echo reply back to the source device, as shown in Figure 3.14.

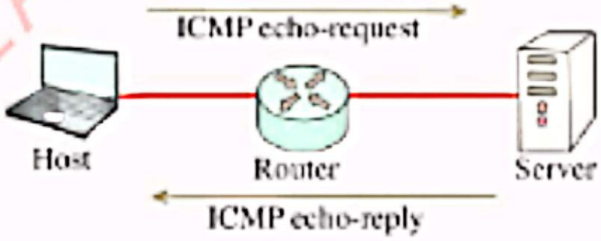


Figure 3.14: *ICMP implementation*

(c) Destination unreachable

Purpose: To notify the source device that the destination cannot be reached.

Process: If a router or gateway cannot deliver a packet to the destination, it sends an ICMP Destination Unreachable message back to the source device.

(d) Time exceeded

Purpose: To indicate that a packet has been discarded because it has exceeded the maximum time to live (TTL) value.

Process: If a packet's TTL value reaches zero before it reaches its destination, the

FOR ONLINE READING ONLY

router discards the packet and sends an ICMP Time Exceeded message to the source device.

(e) Redirect

Purpose: To inform the source device of a better route for sending packets.
Process: If a router receives a packet and determines there is a more efficient route for future packets, it sends an ICMP Redirect message to the source device.

Congestion control

Imagine a highway packed with cars, all struggling for the same lane. This traffic jam is analogous to what happens in a network when there is too much data flowing at once. This phenomenon is called *congestion*. It slows everything down and disrupts smooth communication.

(a) Effect of congestion

As congestion increases, messages experience delays, just like cars stuck in traffic. This means it takes longer for information to travel across the network, impacting performance and user experience. These delays can trigger a congestion effect. When messages arrive late or not at all, they might need to be re-sent, further adding to the traffic and worsening the congestion.

(b) Congestion control algorithms

These algorithms act like traffic cops, regulating the flow of data packets entering the network. Their primary goal is to ensure efficient use of the shared network infrastructure and prevent a complete breakdown, known as congestive collapse. There are two congestion control algorithms namely; Leaky Bucket Algorithms (LAB) and Token Bucket Algorithms (TBA).

(i) *Leaky Bucket Algorithm (LAB):* The LAB controls the rate at which data packets enter the network, smoothing out burst traffic and preventing network overload. If data arrives faster than the network can handle, the excess is discarded to ensure efficient resource utilisation. Data packets entering the network are like water flowing into the bucket (see Figure 3.15).

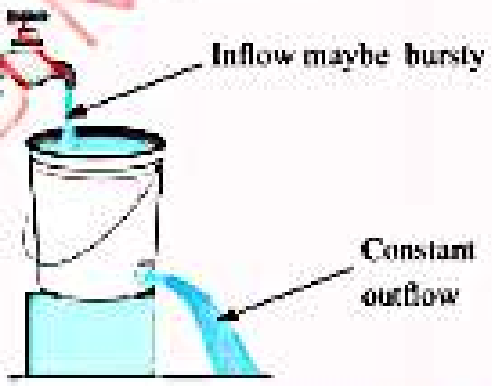


Figure 3.15: *Water flowing into the bucket*

(ii) Token Bucket Algorithm (TBA): The TBA uses tokens to represent permission to send data packets. The bucket refills with tokens at a fixed rate, and as long as there are enough tokens, data can be transmitted at its peak burst rate. If the bucket is empty, the data flow needs to wait until more tokens are generated.

Network Quality of Service (QoS)

In today's interconnected world, ensuring efficient data transmission across networks is crucial. Quality of Service (QoS) is a key concept in network management that prioritises and regulates data traffic to optimise performance and user experience.

ONLINE READING COPY

This discussion explores the importance of QoS, its components, implementation methods, and real-world applications.

(a) Importance of QoS

QoS is essential for reliable and predictable network performance. It manages bandwidth, reduces latency, and mitigates packet loss, benefiting critical applications like voice and video conferencing, online gaming, and real-time data analytics. QoS also ensures fair resource allocation, preventing congestion and bottlenecks during peak usage.

(b) Key components of QoS

Bandwidth management: QoS allocates available bandwidth based on policies and traffic characteristics. It prioritises critical applications while providing sufficient resources for less time-sensitive traffic.

(i) *Traffic prioritisation:* Differentiated Services (DiffServ) and Integrated Services (IntServ) classify and prioritize traffic based on application requirements, user profiles, or service level agreements (SLAs).

(ii) *Traffic shaping and policing:* Traffic shaping controls data transmission rates, smoothing traffic flows to prevent congestion. Traffic policing enforces rate limits to ensure compliance with QoS policies.

(iii) ***Packet scheduling:*** Algorithms like Weighted Fair Queuing (WFQ), Class-Based Queuing (CBQ), and Hierarchical Fair Service Curve (HFSC) determine the order of packet transmission, giving priority to higher-priority traffic classes.

Exercise 3.5

1. The Internet has revolutionised communication and collaboration on a global scale. Explain how a deep understanding of the network layer empowers individuals and businesses to leverage the power of the Internet effectively.
2. Explain the concept of routing protocols and how they help routers choose the best paths for data to travel. How does this choice of routing algorithm affect the performance and fault tolerance of different network topologies (e.g., mesh, star, ring)? Provide a detailed analysis with examples.
3. Imagine you are a network administrator troubleshooting a slow network connection. How would understanding the network layer's functionalities, particularly routing, help you diagnose the issue?
4. How does Quality of Service (QoS) impact network performance and user experience in modern digital environments?
5. What are the key components and implementation methods of Quality of Service (QoS) mechanisms, and how do they contribute to optimising network traffic?

Internet Protocol (IP) addressing

Internet Protocol version 4 (IPv4)

Each device in TCP/IP networks is assigned a logical IP address for identification. Unlike the Data-Link layer

FOR CONTENT READING ONLY

address (such as a MAC address), the IP address functions at the network layer and is independent of the Data-Link layer address. Every device and network component that uses TCP/IP requires a unique IP address.

An IP address is akin to a street address, designating a system's location within the network. Just as a street address must be unique to a particular house, an IP address must be globally unique and conform to a standardised format. Each IP address comprises a network ID and a host ID.

The network ID, also referred to as the network address, identifies systems that are part of the same physical network defined by IP routers. All systems within this network must have the same network ID, and this network ID must be unique across the entire internetwork.

The host ID, or host address as shown in Figure 3.6, identifies individual workstations, servers, routers, or other TCP/IP hosts within a network. Each host's address must be unique within its respective network ID. The concept of network ID applies to any type of IP network, regardless of whether it is class-based, a subnet, or a supernet.

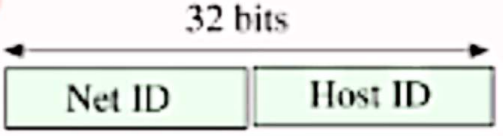


Figure 3.16: IPv4 address format

An IP address is made up of 32 bits. To simplify working with these bits, the 32-bit IP address is commonly divided into four 8-bit sections called octets, as shown in Figure 3.17. Each octet is then converted into a decimal number ranging from 0 to 255, and the sections are separated by periods. This format, known as dotted decimal notation, is used to represent IP addresses. For instance, the binary notation 11000000 10101000 00000011 00011000 corresponds to the dotted decimal notation 192.168.3.24.



Figure 3.17: Example of IPv4 address format

IP Address classes

Initially, the Internet community introduced five address classes (A, B, C, D, and E) to cater to networks of varying sizes. Each address class specifies the allocation of bits for the network ID and host ID, as well as the permissible number of networks and hosts. TCP/IP supports class A, B, and C addresses for assigning hosts, among the five classes established.

Class A

Class A addresses are designated for networks needing a large number of hosts. In a Class A address, the highest-order bit is set to zero. The next seven bits in the first octet form the network ID, while the remaining 24 bits (the last three octets) represent the host ID. This structure permits 128 possible network IDs (0 to 127), but only 126 are usable (excluding 0 and 127), and up to 16,777,214 valid hosts per network. Class A addresses range from 1.0.0.0 to 126.0.0.0 in decimal notation, with 127.0.0.0 reserved for loopback and

FOR ONLINE READING ONLY

inter-process communication, as shown in Table 3.2.

Class B

Class B addresses are assigned to networks ranging from medium-sized to large-sized. In a Class B address, the two highest-order bits are always set to binary 10. The following 14 bits (completing the first two octets) form the network ID, while the remaining 16 bits (last two octets) represent the host ID. This allocation allows for 16,384 networks and up to 65,534 valid hosts per network. Class B addresses span from 128.0.0.0 to 191.255.255.255 in decimal notation, encompassing the range from 128 to 191 for the first octet, as shown in Table 3.2.

Class C

Class C addresses are assigned to small-sized networks. In a Class C address, the three highest-order bits are always set to binary 110. The following 21 bits (completing the first three octets) form the network ID, while the remaining 8 bits (last octet) represent the host ID. This allocation allows for 2,097,152 networks and up to 254 valid hosts per network. Class C addresses range from 192.0.0.0 to 223.255.255.255 in decimal notation, encompassing the range from 192 to 223 for the first octet, as shown in Table 3.2.

Class D

Class D addresses are specifically reserved for IP multicast addresses. In a Class D address, the four highest-order bits are always set to binary 1110. The remaining bits are used for the multicast group address that interested hosts recognise. Class D addresses are supported by applications to multicast data to hosts capable of receiving multicast on an internetwork. Class D addresses range from 224.0.0.0 to 239.255.255.255 in decimal notation, covering the first octet values from 224 to 239, as shown in Table 3.2.

Class E

Class E is an address class designated for experimental purposes and reserved for future use. In a Class E address, the four highest-order bits are set to binary 1111, as shown in Table 3.2.

Table 3.2: *Five IP address classes*

Class	First octet decimal (Range)	First octet binary (Range)	IP range	Subnet mask	Hosts per Network ID	Number of networks
A	0 - 127	0xxxxxxx	0.0.0.0 - 127.255.255.255	255.0.0.0	$2^{24} - 2$	2^7 (128)
B	128 - 191	10xxxxxx	128.0.0.0 - 191.255.255.255	255.255.0.0	$2^{16} - 2$	2^{14} (16,384)
C	192 - 223	110xxxxx	192.0.0.0 - 223.255.255.255	255.255.255.0	$2^8 - 2$	2^{21} (2,097,152)
D	224 - 239	1110xxxx	224.0.0.0 - 239.255.255.255	Not applicable	Not applicable	Not applicable
E	240 - 255	1111xxxx	240.0.0.0 - 255.255.255.255	Not applicable	Not applicable	Not applicable

FOR ONLINE READING ONLY

Network ID guidelines

Follow these guidelines when assigning a network ID:

(i) The network ID must be unique within the IP internetwork. If you plan to connect to the public Internet, it must be unique on the Internet. If you are not connecting to the public Internet, the local network ID must be unique within your private internetwork.

(ii) The network ID cannot start with the number 127, as this number is reserved for internal loopback functions in a class A address.

(iii) The network ID cannot have all bits set to 1, as this is reserved for the IP broadcast address.

(iv) The network ID cannot have all bits set to 0, as this is used to indicate a specific host on the local network and is not routed.

Class ranges of network IDs

Table 3.3: *Class ranges of network ID*

Address Class	First Network ID	Last Network ID
Class A	1.0.0.0	126.0.0.0
Class B	128.0.0.0	191.255.0.0
Class C	192.0.0.0	223.255.255.0

Host ID guidelines

The host ID serves to identify a TCP/IP host within a network, and when combined with the IP network ID, it forms an IP address. Here are some guidelines to follow when assigning a host ID:

(i) The host ID must be unique within the network ID.

(ii) It is not allowed to have all bits set to 1 within the host ID, as this is reserved as a broadcast address used to send packets to all hosts on a network.

(iii) It is not permitted to have all bits set to 0 within the host ID, as this is reserved to indicate the IP network ID.

Class ranges of host IDs

Table 3.4: *Class ranges of host IDs*

Address class	First host ID	Last host ID
Class A	*N*.0.0.1	*N*.255.255.254
Class B	*N.N*.0.1	*N.N*.255.254
Class C	*N.N.N*.1	*N.N.N*.254

FOR ONLINE READING ONLY

Exercise 3.6

1. (a) How would you determine the class for a given IP address in dotted decimal notation?
   (b) Identify class for the following IP addresses in dotted decimal notation:
   (i) 100.31.21.11
   (ii) 165.10.11.1
   (iii) 242.12.13.14
   (iv) 200.200.200.200
   (v) 230.13.56.100
2. How would you determine the class for a given IP address in binary notation? Identify class for the following IP addresses in binary notation:
   (a) 11110001 11111000
   10110011 00011001
   (b) 11100111 00001111
   01110000 11110001
   (c) 11000111 00111110
   00110011 00000000
   (d) 10110011 10111110
   00110011 11111111
   (e) 01000111 00110110
   10110011 00011111
3. (a) How would you determine Network ID and Host ID parts for a given IP address?
   (b) Identify Network ID and Host ID parts for the following IP addresses:
   (i) 125.100.90.10
   (ii) 201.198.168.72
   (iii) 165.19.20.6
4. Calculate the number of Network IDs and Host IDs for each IP address class. Then, pick one network ID from each class and list all possible IP addresses that can be assigned within that network.

Network masks

A network mask, also known as an default mask, is a 32-bit value used to differentiate the network ID from the host ID in any given IP address. The network mask is defined as follows:

(a) All bits that correspond to the network ID are set to 1.

(b) All bits that correspond to the host ID are set to 0.

Network masks are typically shown in binary or dotted decimal notation, as seen in Table 3.5.

Table 3.5: *Binary and dotted decimal representation of default network masks*

Address class	Bits for network mask	Network mask
Class A	11111111 00000000 00000000 00000000	255.0.0.0
Class B	11111111 11111111 00000000 00000000	255.255.0.0
Class C	11111111 11111111 11111111 00000000	255.255.255.0

FOR ONLINE READING ONLY

Network prefix length representation of network masks

Network prefix length representation of default network masks are shown in Table 3.6.

Table 3.6: *Network prefix Length representation of network masks*

Address class	Bits for network mask	Network prefix
Class A	11111111 00000000 00000000 00000000	/8
Class B	11111111 11111111 00000000 00000000	/16
Class C	11111111 11111111 11111111 00000000	/24

For instance, the class B network ID 138.96.0.0 with a default network mask of 255.255.0.0 would be represented in network prefix notation as 138.96.0.0/16. This method of representation is also referred to as "Classless Interdomain Routing (CIDR)" notation.

Public and Private IP addresses

When a local network is not connected to the Internet, any IP addressing scheme can be used. For Internet connectivity, two types of addresses are used: public and private addresses.

Public addresses: Assigned by InterNIC, these are globally unique and routed across the Internet. For example, a CIDR block like 203.0.113.0/24 is allocated, ensuring proper routing.

Private addresses: Reserved for internal use, they are not routable on the Internet. Defined in RFC 1918, they include: **10.0.0.0/8** for class A ; **172.16.0.0/12** for class B ; and **192.168.0.0/16** for class C. These addresses are often converted to public addresses using Network Address Translation (NAT) or Port Address Translation (PAT).

Dynamic vs. Static IP addresses:

(a) Dynamic IP (DHCP)

These addresses are automatically assigned by a DHCP server. They can change over time or when the device reconnects to the network. They are ideal for most personal devices and general use.

(b) Static IP

These addresses are manually assigned and does not change. They are essential for servers, printers, and other devices that need a constant IP address. They are useful for devices that require consistent access and easy network configuration.

Routing and the default gateway

When the source and destination are on the same network, IP directly delivers packets via the local area network (LAN). If the destination is on a different network, IP sends packets to the default gateway, a router on the same network.

FOR OFFLINE READING ONLY

on, the route from the Internet to any subnet of a given IP address remains the same.

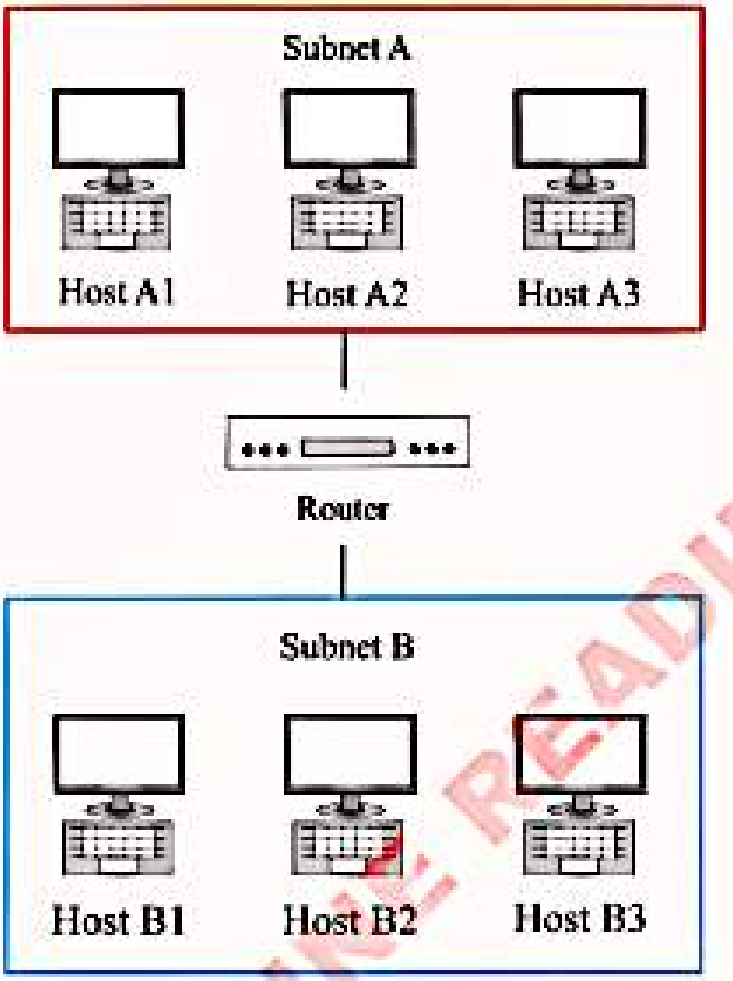


Figure 3.18: *Network subnets*

Subnetting is a method used to divide a single IP network into smaller, more manageable subnetworks or subnets. This process helps in optimising network performance, enhancing security, and making efficient use of IP addresses. Subnetting involves breaking down a large network into smaller segments, each with its own unique network identifier.

Key concepts in Subnetting

(a) IP address and subnet mask

(i) **IP Address:** An IP address is a 32-bit number (IPv4) represented in dotted-decimal notation (e.g., 192.168.1.1).

(ii) **Subnet mask:** A subnet mask is also a 32-bit number that separates the network portion of the IP address from the host portion. Common subnet masks include 255.255.255.0 (for a /24 network).

(b) Network and host portions

(i) **Network portion:** The part of the IP address that identifies the specific network as determined by the subnet mask.

(ii) **Host portion:** The part of the IP address that identifies a specific device (host) within that network.

(c) Subnetting process

(i) *Determine the network requirements*: Identify the number of subnets needed and the number of hosts per subnet.

(ii) *Select an appropriate subnet mask:* Choose a subnet mask that supports the required number of subnets and hosts.

(iii) *Calculate subnet addresses:* Use the subnet mask to determine the range of IP addresses within each subnet.

FOR ONLINE READING ONLY

(d) Examples of subnetting

Example 3.1:

Imagine a ZCB group of companies Ltd have an IP address of 192.168.1.0/24 according to information from its IT department. The company needs to create four (4) subnets. As an IT Manager, perform this task.

Step 1: *Determine the subnet mask*

(i) The original default network mask is 255.255.255.0 (/24), which allows for 256 IP addresses (2^8).

(ii) To create four subnets, you need to borrow 2 bits from the Host ID part.

(iii) To calculate the number of subnets, use the formula, number of subnets=2^n , where n is the borrowed bits. In this case, n=2, therefore $2^2 = 4$ subnets.

(iv) Change the borrowed 2 bits from bit 0 to bit 1 in the host portion (add them to the network ID portion of the mask). This is to say, the default mask for class C 11111111 111111 11111111 00000000 will now be come 11111111 111111 11111111 11000000

(v) The resulting subnet mask is now 255.255.255.192 (/26).

(vi) From the remaining zeros of the subnet mask, calculate the number of addresses per each subnet as allowed (2^n, where n is the number of zeros in the host ID part of the subnet mask).

(vii) Taking the remaining 6 zeros in the host ID portion of the mask (2^6), which allows for 64 IP addresses per subnet.

Step 2: *Calculate subnet addresses*

Each subnet has 64 address ranges, the granted address will be the first address then add 63 to find the last address in each case. Remember the first address is the subnet address and the last address is the broadcast address.

(i) Subnet 1: 192.168.1.0/26 (Range: 192.168.1.0 - 192.168.1.63)

(ii) Subnet 2: 192.168.1.64/26 (Range: 192.168.1.64 - 192.168.1.127)

(iii) Subnet 3: 192.168.1.128/26 (Range: 192.168.1.128 - 192.168.1.191)

(iv) Subnet 4: 192.168.1.192/26 (Range: 192.168.1.192 - 192.168.1.255)

Step 3: *Assign addresses*

Each subnet can accommodate up to 62 hosts (64 total addresses minus 2 for network and broadcast addresses).

FOR ONLINE READING ONLY

(e) Benefits of subnetting the network

(i) *Efficient IP address management*

(ii) Reduces the wastage of IP addresses and allows better utilization of the available address space.

(iii) *Improved network performance*

(iv) Limits the size of broadcast domains, reducing network traffic and collisions.

(v) *Enhanced security*

(vi) Segments the network into smaller subnets, making it easier to manage and secure each segment individually.

(vii) *Simplified troubleshooting*

Easier to identify and isolate network issues within smaller subnetworks.

Internet protocol version 6 (IPV6)

One of the primary motivations behind the development of IPv6 was to increase the IP address pool. IPv6 uses a 128-bit address, which allows for a maximum of 2^{128} addresses, or 340,282,366,920,938,463,463,374,607,431,768,211,456 addresses—enough to assign multiple IP addresses to every grain of sand on Earth. Consequently, the traditional 32-bit IPv4 dotted-quads are no longer sufficient. IPv6 addresses are formatted into eight 16-bit hexadecimal blocks, separated by colons. These addresses are not only longer but also include both numbers and letters. At first glance, IPv6 addresses, such as 2001:0db8:3c4d:0015:0000:0000:abcd:ef12, may look like an impenetrable secret code.

(a) IPv6 address types

IPv6 does not utilise classes. Instead, it supports the following three types of IP addresses: Unicast, multicast, and anycast.

In IPv6, unicast and multicast messaging function in the same way as they do in IPv4. However, IPv6 does not support broadcast; instead, it achieves similar results using its multicast mechanism. Multicast addresses in IPv6 begin with 'FF' (255), similar to IPv4 addresses. Anycast in IPv6 is a variation of multicast: while multicast sends messages to all nodes within the multicast group, anycast sends messages to any one node in the group. Anycast is an advanced networking technique designed to meet the application's failover and load balancing requirements.

(b) IPv6 header format

In addition to the expanded address space, IPv6 introduces several enhancements, including changes to the header format as shown in Figure 3.19.

FOR ONLINE READING ONLY

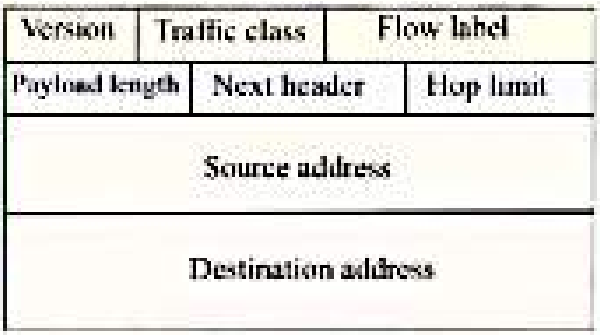


Figure 3.19: *IPv6 header format*

The IPv6 header format includes various fields, each serving a specific purpose:

(i) **Version:** Specifies the version of the IP protocol being used, which is IPv6 in this case.

(ii) **Traffic class:** Similar to the Type of Service (ToS) field in IPv4, the Traffic Class field in IPv6 allows for the classification and prioritization of packets.

(iii) **Flow label:** This field is used to identify packets belonging to the same flow, facilitating quality of service (QoS) and flow-based routing.

(iv) **Payload length:** Indicates the length of the data payload in the packet, excluding the header.

(v) **Next header:** Specifies the type of the next header in the packet, allowing for the chaining of multiple headers.

(vi) **Hop limit:** Similar to the Time to Live (TTL) field in IPv4, the hop limit field in IPv6 specifies the maximum number of hops or routers that a packet can traverse before being discarded.

(vii) **Source address:** Represents the IPv6 address of the packet's source.

(viii) **Destination address:** Represents the IPv6 address of the packet's destination.

Overall, the IPv6 header format is more streamlined and efficient compared to IPv4, providing improved support for modern networking requirements such as security, mobility, and quality of service.

Activity 3.7:

Exploring the network layer with simulation

Aim:
To deepen understanding of the network layer through simulation, focusing on end-to-end packet delivery, network layer protocols, and the operational role of routers.

Materials:
Internet access; Computer or laptop; Simulation software (Such as Cisco Packet Tracer, GNS3); Web browser; Text editor or word processing software; Diagramming tools (Such as Lucidchart, Microsoft Visio)

Instructions:
Use the Internet and simulation tools to complete the following tasks:

1. Simulate end-to-end delivery
   (a) Use network simulation software (Such as Cisco Packet Tracer, GNS3) to simulate how the network layer enables end-to-end delivery of packets

FOR ONLINE READING ONLY

between your computer and a simulated web server.

(b) Explore how simulated network configurations affect packet routing and delivery.

2. Examine network layer protocols
   (a) Research how protocols at the network layer (such as IPv4 and IPv6) facilitate the delivery of web content in various simulated network environments.
   (b) Compare and contrast the functionalities of different network layer protocols using simulation scenarios.
3. Study the role of routers
   (a) Utilize simulation to observe the role of Layer 3 devices (routers) in managing simulated outgoing and incoming packets.
   (b) Simulate scenarios where routers handle packets exiting your simulated network and packets entering your network from external sources.
4. Document your findings
   (a) Record your observations and insights from the simulations.
   (b) Compile your findings, including screenshots or diagrams from the simulation, into a portfolio for future reference and learning.

Deliverables:

Prepare a simulation report, include screenshots or diagrams to illustrate key simulation scenarios.

Exercise 3.7

1. Tanzania Institute of Education has been allocated the network address 211.77.20.0 and needs to create 6 subnets. The largest subnet must accommodate up to 25 hosts, and future subnet sizes should not exceed 30 hosts. Identify the valid subnets and the host IP addresses within each subnet.
2. A school is granted site address (IP address) 11.0.0.0/24 for its use. Design the subnets for the school considering the following number of hosts:
   (a) Subnet A with 50 hosts
   (b) Subnet B with 25 hosts
   (c) Subnet C with 10 hosts, and
   (d) Subnet D with 5 hosts
3. NECTA head office is granted the site address 135.65.20.31 by the regional ISP (Internet Service Provider) for its use.
   (a) Create 16 subnets for NECTA
   (b) Calculate the number of hosts per subnet
   (c) Determine the subnet ranges

Transport layer

The Transport layer is responsible for segmenting and reassembling data into a coherent data stream. It facilitates data exchange between end systems, segmenting and reassembling data from upper-layer applications into a unified

FOR ONLINE READING ONLY

data stream. This layer provides end-to-end data transport services, establishing a logical connection between the sending and receiving hosts across an Internet. Additionally, the Transport layer manages the multiplexing of upper-layer applications, session establishment, and the termination of virtual circuits. By offering transparent data transfer, it conceals network-dependent details from the higher layers. The Transport layer can operate in a connectionless-mode using User Datagram Protocol (UDP) or in a connection-oriented mode using Transmission Control Protocol (TCP).

Connection-oriented transport

The connection-oriented transport service guarantees that data is delivered without errors, in the correct order, and without any losses or duplications. The Transport layer also focuses on optimising the use of network services and delivering the requested quality of service to session entities. For instance, a session entity might specify acceptable error rates, maximum delay, priority, and security levels. Consequently, the transport layer must adapt to meet these requirements while maintaining efficient data transmission.

A service is classified as connection-oriented when it exhibits certain characteristics. These characteristics include:

(a) *Virtual circuit setup:* The service establishes a virtual circuit between the communicating entities. This setup often involves a process like a three-way handshake, where synchronization and connection parameters are exchanged.

(b) *Sequencing:* The service ensures that data is transmitted and received in a specific order. This sequencing guarantees that data is delivered to the recipient exactly as it was sent.

(c) *Acknowledgments:* The service utilises acknowledgments to confirm the successful receipt of data segments. These acknowledgments provide feedback to the sender, ensuring reliable data transfer.

(d) *Flow control:* Connection-oriented services implement flow control mechanisms to regulate the rate of data transmission. This helps to prevent overwhelming the receiving entity and avoids congestion.

Flow control

Flow control is essential for managing the rate at which data is transmitted between a sender and a receiver, ensuring that the receiver can handle the incoming data without becoming overwhelmed. At the Transport layer, data integrity is maintained through the management of flow control and the provision of reliable data transport between systems. Flow control prevents the sending host from overwhelming the receiving host's buffers, which can result in data loss. Reliable data transport involves establishing a connection-oriented communication session, ensuring the following:

1. Received segments are acknowledged back to the sender.
2. Unacknowledged segments are retransmitted.

FOR ONLINE READING ONLY

3. Segments are correctly reordered upon arrival.
4. Data flow is regulated to prevent congestion and data loss.

Virtual circuit setup: In the context of reliable transport operations, a device initiates a connection-oriented communication with a remote device by establishing a session. This process typically involves a call setup or a three-way handshake. Once the connection is established, data is transmitted between the devices. After the transmission is complete, the connection is terminated, effectively dismantling the virtual circuit. Maintaining data integrity through flow control and reliable data transport ensures the smooth and error-free exchange of information between systems. Figure 3.19.

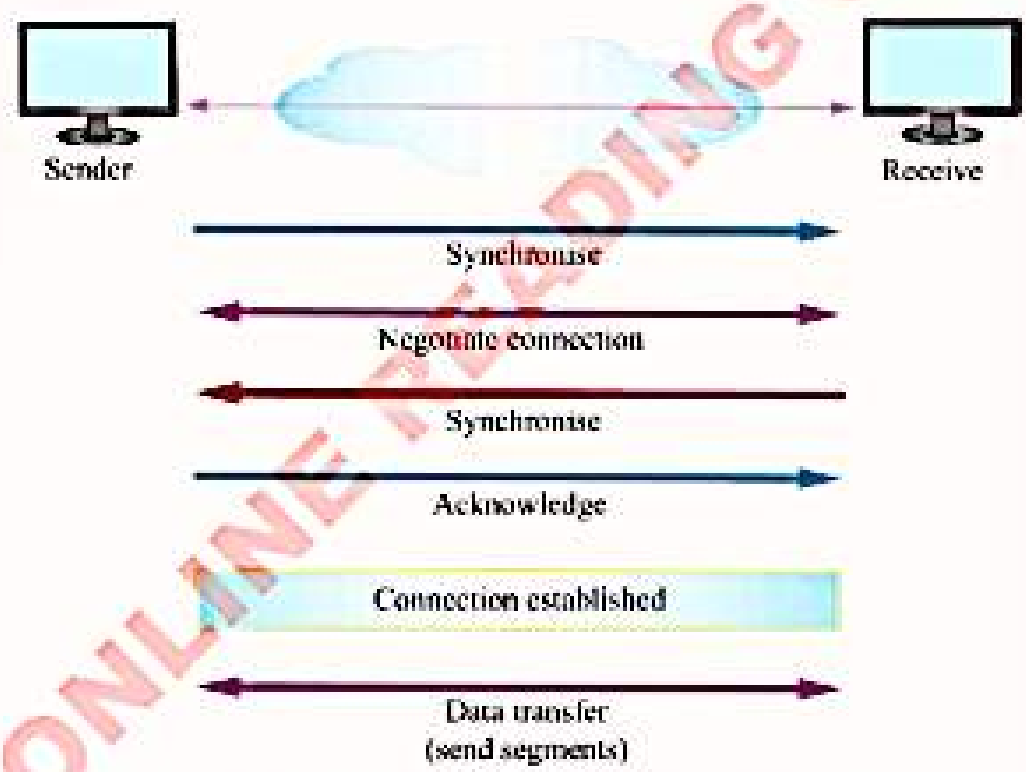


Figure 3.20: *Establishing a connection-oriented session*

In a connection-oriented session, communication is initiated through a process known as the three-way handshake. This handshake involves the following steps:

1. The first segment, called the "connection agreement," is sent to request synchronisation.
2. The second and third segments establish acknowledgment and connection parameters between the hosts.
3. The synchronisation of the receiver's sequencing is also requested to establish a bidirectional connection.
4. The final segment serves as an acknowledgment, confirming the acceptance of the connection agreement and the establishment of the actual connection.

FOR ONLINE READING ONLY

Once the connection is established, data transfer can begin, and the session continues until it is eventually terminated. However, issues can arise during the transfer process. For example, congestion may occur when high-speed data generation exceeds the capacity of the network. Additionally, when multiple computers send datagrams through a single gateway or destination, congestion can also occur. This congestion can be likened to a bottleneck on a freeway, where the sheer volume of computers overwhelms the network's capacity, rather than individual sources. When a machine receives an excessive flood of datagrams, it buffers them in memory. However, if the influx of datagrams continues without relief, the device's memory may become overwhelmed, resulting in the discarding of additional datagrams as shown in Figure 3.21. In reliable, connection-oriented data transfer, it is crucial that the datagrams are delivered to the receiving host in the exact sequence in which they were transmitted. Any deviation from the correct sequence will result in a transmission failure. To address this, the receiving host acknowledges the receipt of each data segment, ensuring reliable data transfer.

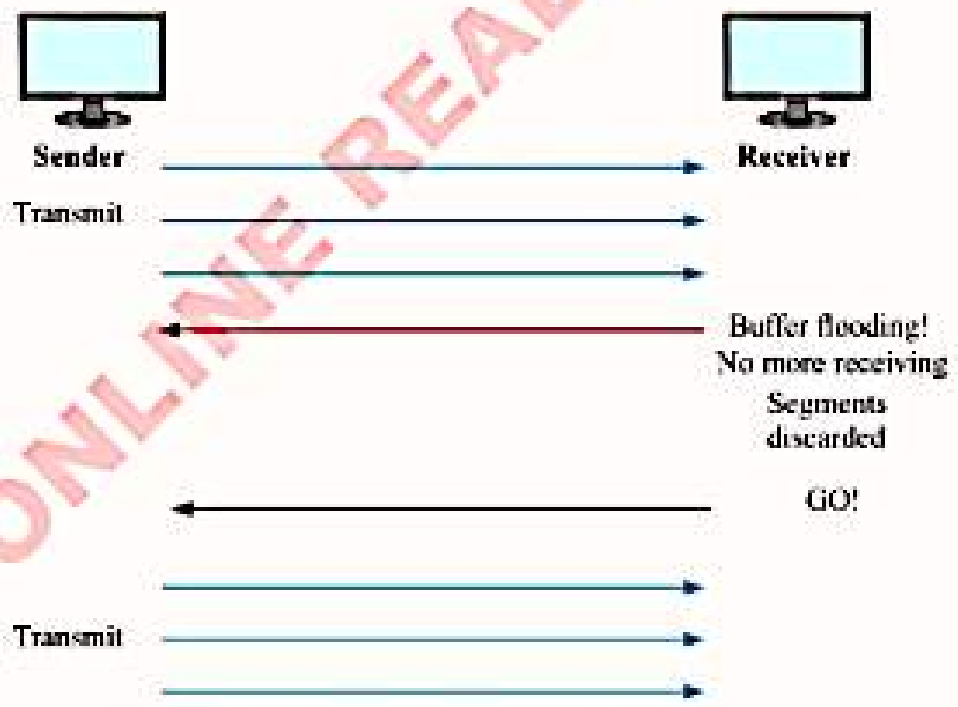


Figure 3.21: *Transmitting segments with flow control*

There are different types of flow control techniques employed in connection-oriented services, including:

(a) Buffering: Excessive data is buffered in memory to prevent overflow and ensure smooth data transfer.

(b) Windowing: The amount of unacknowledged data that can be sent at a time is controlled using windows. This helps to optimise data transmission and prevent overwhelming the receiver.

FOR ONLINE READING

(c) Congestion avoidance: To manage network congestion caused by high data traffic, congestion avoidance techniques are employed. These techniques aim to regulate data flow and prevent network congestion from occurring.

Buffering

Buffering involves temporarily storing excessive data in memory (buffers) to prevent overflow and ensure smooth data transfer. This technique is crucial for maintaining efficient and reliable communication between devices with different processing speeds or capabilities.

Buffering mechanisms

1. ***Data storage:*** When a sender transmits data faster than the receiver can process, excess data is stored in a buffer, a temporary holding area in the receiver's memory.
2. ***Smooth data transfer:*** The buffer allows the receiver to process data at its own pace, preventing data loss or overflow.
3. ***Buffer management:*** Efficient management ensures the buffer does not overflow; if full, the sender is signaled to slow down or pause transmission.
4. ***Clearing buffers:*** Once processed, buffered data is cleared, making room for new data, maintaining a steady data flow.

Figure 3.22 shows how the buffering works.

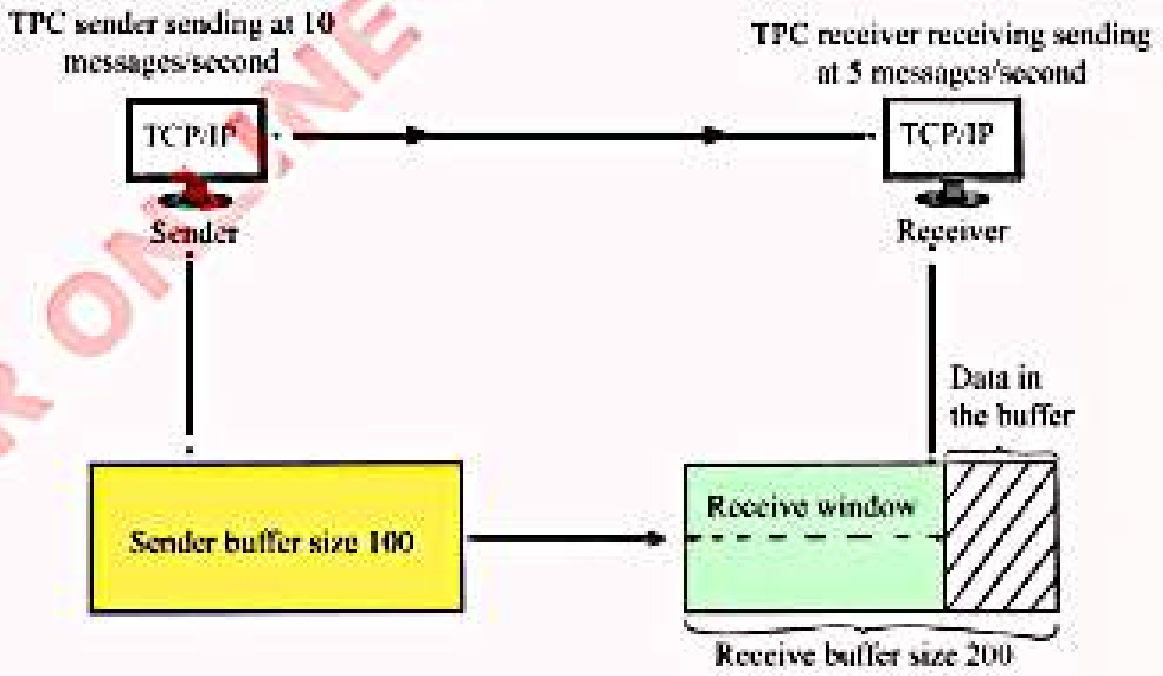


Figure 3.22: *How buffering works*

Benefits of buffering

1. *Prevents data loss:* Temporarily stores excess data until the receiver is ready to process it.

FOR ONLINE READING ONLY

2. *Ensures smooth Communication:* Manages data flow between sender and receiver, even at different speeds.
3. *Handles burst data:* Manages large amounts of data sent in short periods.

Practical examples of buffering

1. *Video streaming:* Ensures continuous playback by buffering video data to handle network speed variations.
2. *File transfer protocols:* Smooths data transfer between systems with different processing speeds.
3. *Network routers and switches:* Uses buffering to handle varying traffic loads, ensuring smooth packet transmission.

Challenges of buffering

1. *Buffer overflow:* Occurs if the buffer becomes full, leading to potential data loss.
2. *Memory usage:* Buffers require memory, which can be constrained on systems with limited resources.
3. *Latency:* Excessive buffering can introduce delays in data transmission and reception.

Windowing

To optimise data throughput, the sender transmits multiple data segments (measured in bytes) without waiting for acknowledgments, known as windowing. Windows control the flow of data by managing the amount of unacknowledged data segments. For example, if a TCP session starts with a window size of 2 bytes and increases to 3 bytes, the sender can send 3 bytes before waiting for an acknowledgment, improving transmission efficiency. Figure 3.22 shows the windowing technique to control congestion.

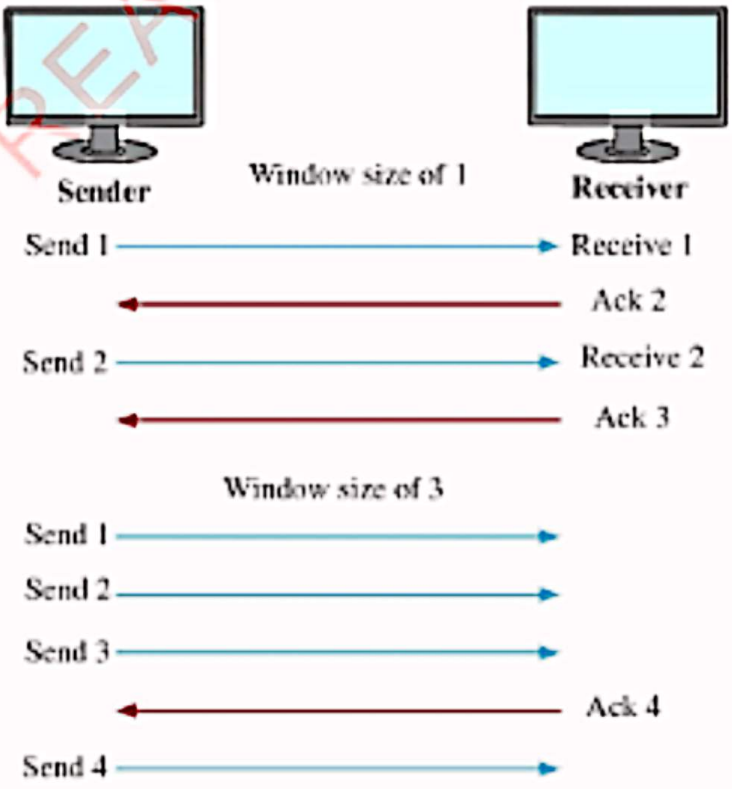


Figure 3.23: *Windowing flow control technique*

Congestion avoidance

Congestion avoidance prevents network congestion, ensuring optimal performance by regulating data flow. Congestion occurs due to heavy data

FOR ONLINE READING ONLY

traffic, bandwidth limitations, network bottlenecks, and numerous simultaneous connections.

Techniques

1. *Traffic shaping*: Regulates data transmission rates using algorithms like token and leaky bucket.
2. *Congestion control algorithms:* Adjusts transmission rates based on network conditions using TCP Tahoe, Reno, and New Reno.
3. *Random early detection (RED):* Proactively drops packets when traffic exceeds thresholds to prevent congestion.
4. *Explicit congestion notification (ECN):* Marks packets instead of dropping them to indicate congestion, prompting senders to reduce rates.
5. *Window-based flow control*: Uses sliding window mechanisms to regulate outstanding packets, adjusting window size based on conditions.

Acknowledgments

Reliable data delivery guarantees the integrity of a data stream during transmission between machines, preventing duplication or loss. This is achieved by employing a fully functional data link and utilising a technique known as positive acknowledgment with retransmission.

In this method, the receiving machine sends an acknowledgment message back to the sender upon successful receipt of data. The sender keeps track of each segment it sends and waits for this acknowledgment before proceeding with the transmission of the next segment. To ensure timely delivery, the transmitting machine starts a timer when a segment is sent. If the timer expires before receiving an acknowledgment from the receiving machine, the sender will retransmit the segment. By implementing positive acknowledgment with retransmission, reliable data delivery is achieved, ensuring the preservation of data integrity and minimising the chances of duplication or loss.

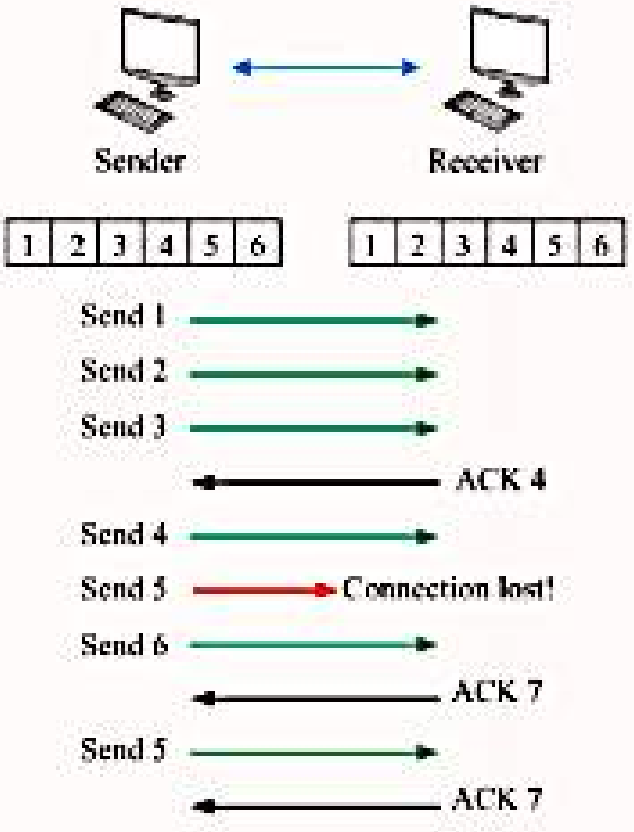


Figure 3.24: *Transport layer reliable delivery*

FOR ONLINE READING ONLY

In Figure 3.24, the sending machine sends segments 1, 2, and 3 to the receiving node. The receiving node confirms successful receipt by requesting segment 4. After getting this acknowledgment, the sender transmits segments 4, 5, and 6. If segment 5 does not arrive, the receiving node requests a retransmission of that segment. The sender resends the missing segment and waits for an acknowledgment before sending segment 7.

In essence, this process ensures the reliable, sequential delivery of messages in a network. It uses error recovery and flow control mechanisms to manage end-to-end message delivery and supports both connection-oriented and connectionless packet delivery.

Sequencing

Sequencing is crucial in connection-oriented services as it ensures data is transmitted and received in a specific order, preserving the exact sequence sent by the sender. This guarantees the recipient receives data accurately as intended, which is essential for time-sensitive or sequence-dependent information.

To achieve sequencing, each data segment is assigned a unique sequence number before transmission. This number helps the recipient identify the correct order of segments. Upon receiving the data, the recipient examines the sequence numbers to reassemble the original message accurately. If segments arrive out of order, the recipient holds them until all necessary segments are received, then arranges them based on their sequence numbers.

Sequencing is vital in applications like video streaming, voice communication, or file transfer, where maintaining data integrity and order is critical. By ensuring data is transmitted and received in the correct sequence, connection-oriented services provide accurate and reliable information delivery.

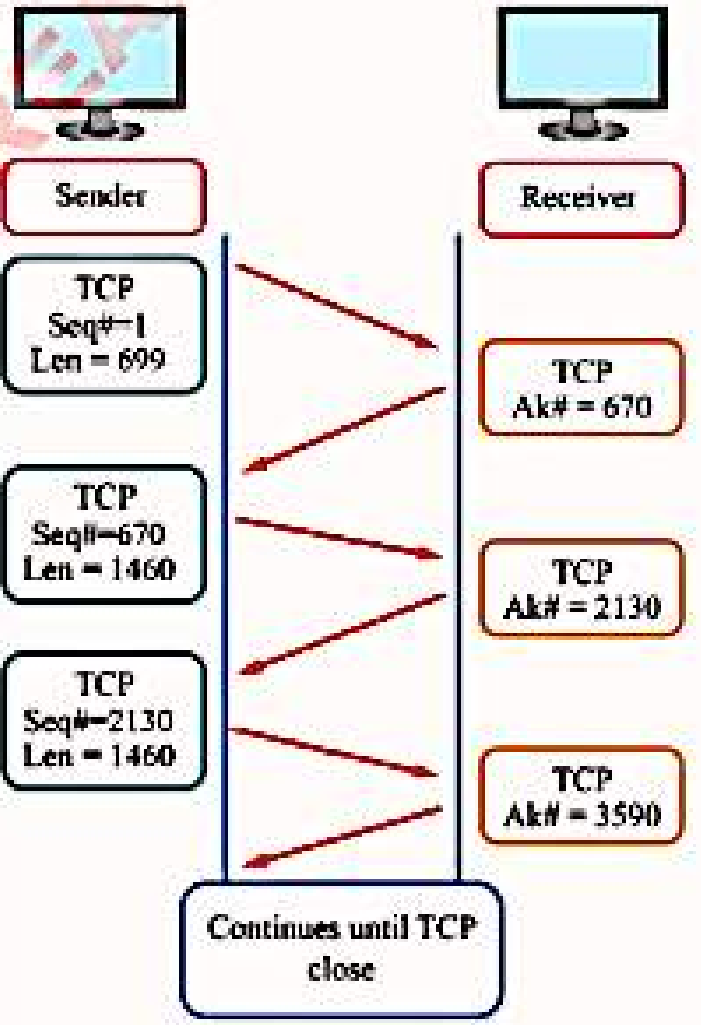


Figure 3.25: *Sequencing in network*

Connectionless-oriented transport

Just like TCP, the User Datagram Protocol Protocol (UDP) is a core communication protocol of the Internet protocol suite. It is used for transmitting and receiving messages (datagrams). However, unlike TCP, UDP operates as a connectionless protocol, negating the need to establish a connection prior to sending the message. Consequently, UDP does not ensure the delivery of data packets to the recipient. It is often referred to as the "fire-and-forget" protocol due to its nature of sending data without concern for its reception. Table 3.7 illustrates the differences between TCP and UDP.

Table 3.7: *Difference between TCP and UDP*

No.	Feature	TCP	UDP
1.	Connection	Connection-oriented	Connectionless
2.	Delivery	Guaranteed delivery of packets	No guarantee of delivery of packets
3.	Reliability	Reliable data transmission	Unreliable data transmission
4.	Order	Preserves order of packets	Order of packets not preserved
5.	Overhead	Higher overhead due to connection setup	Lower overhead without connection setup
6.	Speed	Generally slower due to connection setup	Generally faster due to no connection setup
7.	Header size	Larger header size	Smaller header size
8.	Acknowledgement	Uses acknowledgements for data delivery	No acknowledgements for data delivery
9.	Flow control	Implements flow control mechanism	No built-in flow control mechanism
10.	Error checking	Provides error checking using checksums	No built-in error-checking mechanism
11.	Application	Suitable for applications requiring reliable and ordered data transmission	Suitable for applications requiring fast and efficient data transmission

Activity 3.8:

Exploring the Transport layer

Aim:

To understand the functions of the Transport layer, including the roles of TCP and UDP, and mechanisms for flow control and congestion avoidance.

Materials Required:

Internet access; Computer or laptop; Web browser; Text editor or word processing software; Diagramming tools (Such as Lucidchart, Microsoft Visio)

FOR ONLINE LEARNING ONLY

Instructions

Use the Internet to complete the following tasks:

1. Explore the Transport layer

   Investigate how the Transport layer facilitates end-to-end delivery of content between your computer and a web server with each click on a website.
2. Examine the roles of TCP and UDP
   (a) Study the functions of Transmission Control Protocol (TCP) and User Datagram Protocol (UDP) in delivering web content to your computer.
   (b) Highlight the differences between TCP and UDP.
3. Flow control and congestion avoidance
   (a) Flow control: Explore mechanisms that manage the rate of data transmission between your computer and a web server to prevent overloading.
   (b) Congestion avoidance: Investigate techniques that regulate data flow to prevent network congestion and ensure optimal performance.

Deliverables:

Written and detailed report including explanations, examples, and any relevant diagrams. Use visuals.

Exercise 3.8

1. Imagine you are designing a video-conferencing application. Would you choose TCP or UDP for data transmission? Justify your answer considering the characteristics of each protocol and the requirements of a video conference
2. Imagine you are developing a reliable file transfer application. Discuss how understanding error detection mechanisms in the Transport layer can be used to improve data integrity during file transfers.
3. Explain the concept of flow control in the Transport layer. How does it prevent data from overwhelming the receiver?
4. Imagine you are downloading a large file from the internet. How might flow control mechanisms implemented in the Transport layer affect the download speed you experience?

Application layer

The topmost layer in both the OSI and TCP/IP models of network communication is known as the Application layer. This layer acts as a bridge between the end-user and the network, offering a range of services and protocols that facilitate communication between applications over a network.

FOR ONLINE READING ONLY

Key functions of the application Layer

(a) ***End-user services:*** The application layer directly offers network services to both end-users and applications. It plays a crucial role in facilitating communication and the exchange of data between software applications and other resources within the network.

(b) ***Protocol support:*** The Application layer supports a range of protocols that establish rules and standards for the exchange of data between network applications. These protocols, such as HTTP, FTP, SMTP, DNS, and DHCP, are widely used for various network communication purposes.

(c) ***Data encoding and presentation:*** While the Presentation layer in the OSI model usually handles tasks like data formatting, encoding, and translation, the application layer in the TCP/IP model often takes on these responsibilities. Its role is to ensure that the data is properly formatted and encoded, making it usable for the receiving application.

(d) ***Network transparency:*** The Application layer simplifies the complexities of the underlying network, providing an easy-to-use interface that lets applications communicate over the network. This abstraction eliminates the need for applications to handle the complex details of network operations. Common Application layer's protocols and their functions are shown in Figure 3.26

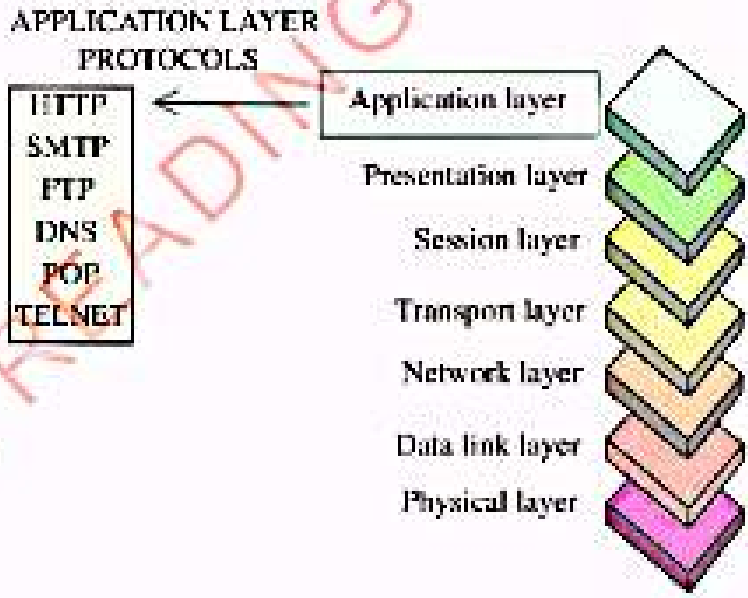


Figure 3.26: ***Common Application layer protocols***

The Application layer protocols are HTTP, SMTP, FTP, DNS, POP, TELNET. Their functions are explained below.

(i) HTTP (Hypertext Transfer Protocol)

This is a protocol that is used for transferring web pages on the Internet. It enables browsers to retrieve and display web content from web servers.

(ii) FTP (File Transfer Protocol)

FTP allows transferring files between a client and a server on a network. It supports both uploading and downloading of files.

FOR ONLINE READING ONLY

Connection: The client connects to the server using FTP and authenticates with a username and password.

Uploading: The client selects a file to send to the server, which stores it in a specified location.

Downloading: The client requests a file from the server, which sends it back for the client to save locally.

FTP ensures secure and reliable file transfers between the client and server.

(iii) SMTP (Simple Mail Transfer Protocol)

This protocol is utilised for sending emails and works together with other protocols like IMAP and POP3 to manage the delivery and retrieval of emails. Figure 3.27 demonstrates how SMTP works.

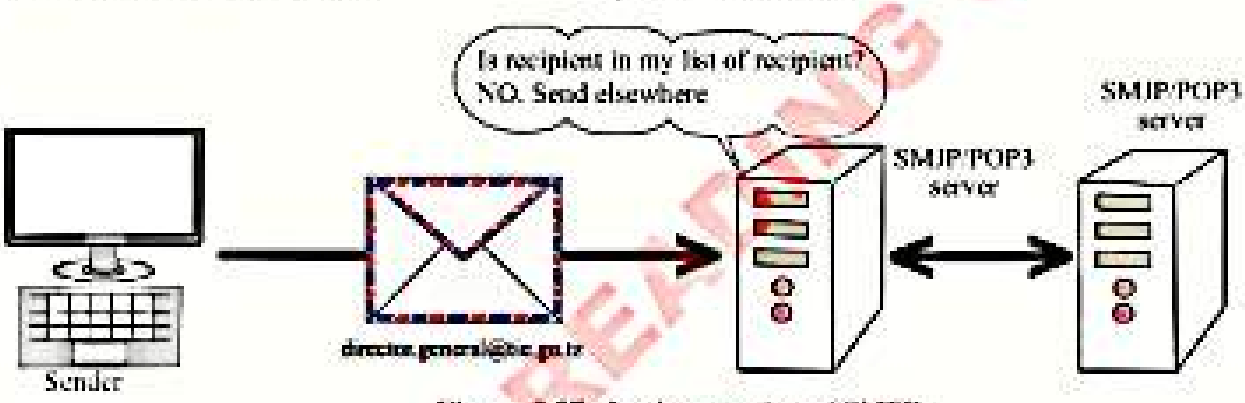


Figure 3.27: *Implementation of SMTP*

(i) *DNS (Domain Name System)*

This protocol translates human-readable domain names (such as, www.tic.go.tz) into IP addresses that computers use to identify each other on the network.

(ii) *DHCP (Dynamic Host Configuration Protocol)*

Automatically assigns IP addresses and other network configuration parameters to devices on a network, enabling them to communicate effectively.

Activity 3.9:

Accessing web pages to explore the Application layer

Aim: To understand how the application layer facilitates web browsing and data exchange between client computers and web servers.

Materials: Computer lab access or a personal computer/laptop; Internet access; and Web browser (such as Chrome, Firefox, Safari)

Instructions

Step 1: Open a web browser on a networked computer.

FOR ONLINE READING ONLY

Step 2: Enter a web address, such as www.necta.go.tz. This sends an HTTP GET request from your computer to NECTA's web server.

Step 3: Observe the server's response. Note the content that loads on the web page. Explain what you see, focusing on how the server's response appears on your screen.

Step 4: Understand that the web page from NECTA's server is rendered on your computer through HTTP at the application layer.

Step 5: Interact with the web page by clicking on any links, filling forms, opening images, or accessing Form VI results. Note how your interactions generate additional HTTP requests.

Step 6: Open embedded links, images, or videos that are hosted on different servers. Observe how these requests are processed and the content is displayed.

Step 7: Close the browser to terminate your session at the application layer.

Deliverables:
Written Report that Summarize your observations and explain how the Application layer handles web browsing. Include descriptions of the server responses and the interaction process, with relevant screenshots and diagrams to illustrate your findings.

Application layer security

(a) *Authentication:* Ensures that users and devices are who they claim to be. Protocols like Kerberos and *mechanisms like password verification are commonly used.*

(b) *Encryption:* Protects data by converting it into a secure format that can only be read by authorised parties. SSL/TLS (Secure Sockets Layer/Transport Layer Security) are widely used for securing web communications.

(c) *Data integrity:* Ensures that the data sent is the same as the data received. Techniques like checksums and hash functions help verify data integrity.

(d) *Access control:* Regulates who can access what resources on a network. It involves permissions, user roles, and policies that restrict access based on predefined rules.

Application layer in different network models

(a) *OSI model:* The application layer is the seventh layer and interacts with the Presentation layer (Layer 6) and the Session layer (Layer 5) to provide comprehensive end-user services.

(b) *TCP/IP model:* The application layer is a broader layer that combines the functions of the OSI model's Application, Presentation, and Session layers, focusing on the protocols that applications use to communicate.

FOR ONLINE READING ONLY

Application layer in daily uses

(a) *Web browsing:* Web browsers use HTTP/HTTPS to access web pages, retrieve data, and display content to users. Figure 3.38 shows a user can enter the Internet through the World Wide Web and request information from the server.

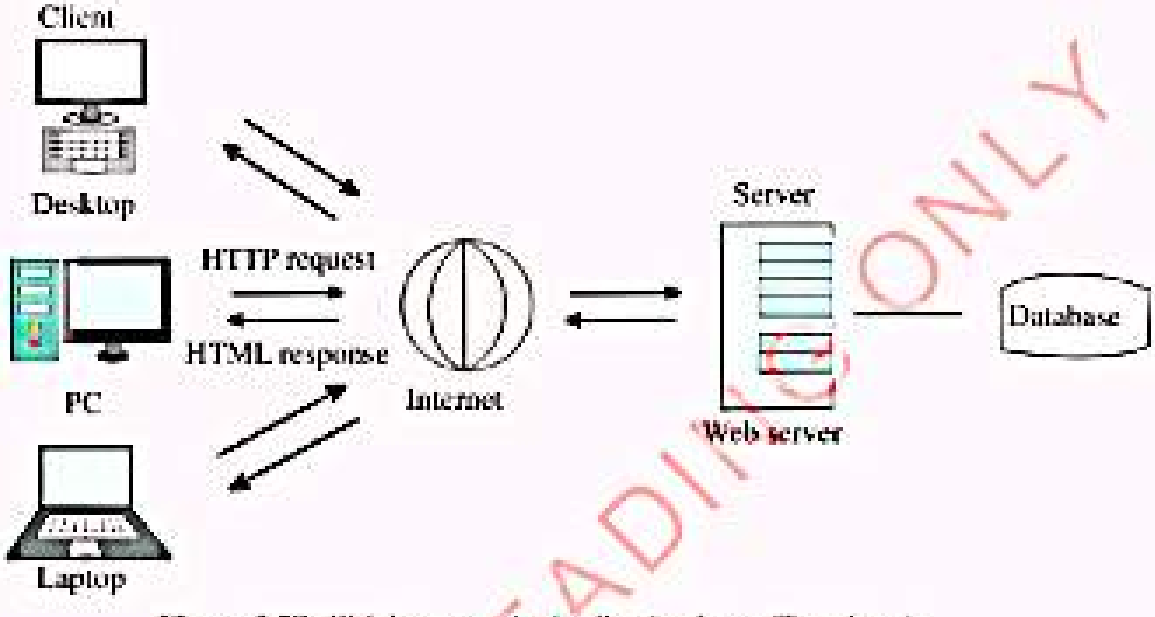


Figure 3.28: *Web browsing in Application layer (To redraw)*

(b) *Email Services:* Email clients use SMTP, IMAP, and POP3 to send, receive, and organise emails.

(c) *File Sharing:* Services like Google Drive and Dropbox use FTP/SFTP protocols for file transfers between clients and servers. This is file sharing as shown in Figure 2.29.

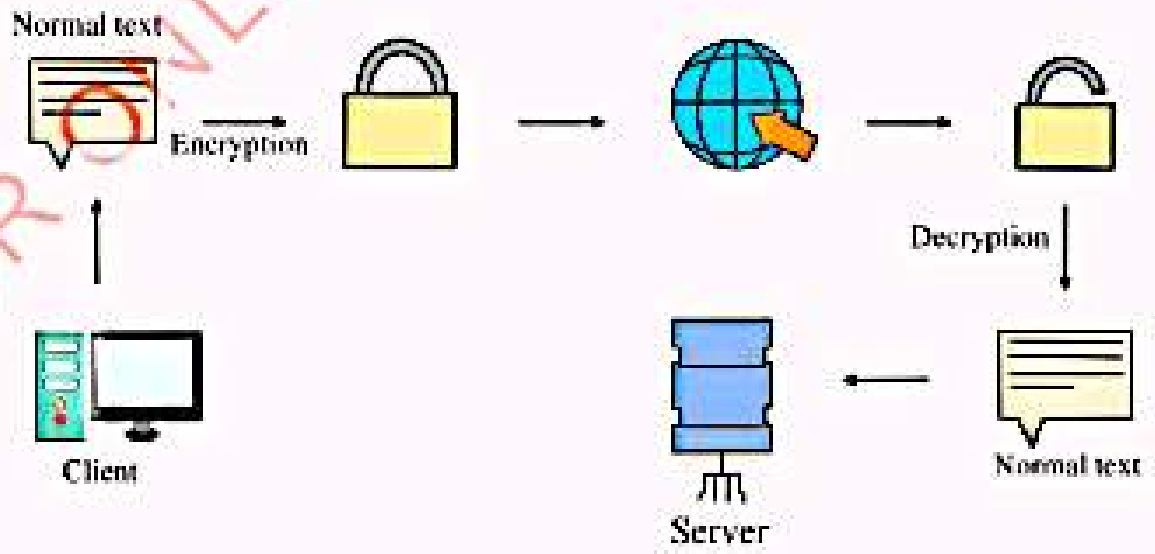


Figure 3.29: *File sharing in Application layer*

(d) *Remote access:* Protocols like SSH (Secure Shell) provide secure remote access to servers and devices over a network.

FOR ONLINE READING ONLY

Activity 3.10:

Exploring the Application Layer

Aim:

To understand the functions and real-world applications of common application layer protocols.

Materials Required:

Internet access; Computer or laptop; Text editor or word processing software; or access to library resources (optional)

Instructions

Step 1: Research protocols

(a) Use reliable Internet sources or library materials to research HTTP, HTTPS, FTP, SMTP, and DNS.

(b) Take notes on each protocol's primary function and typical use cases.

Step 2: Analyse features

(a) Look into the detailed features of each protocol.

(b) Compare the advantages and disadvantages of each.

Step 3: Explore applications

(a) Identify real-world applications that use these protocols.

(b) Document how these applications utilize the protocols in their operations.

Deliverables:

Written Report summarizing your research on each protocol, including their primary functions and use cases. Report should include analysis of the key features, advantages, and disadvantages of each protocol.

Document your findings in a portfolio.

Exercise 3.9

1. Use a network analysis tool such as Wireshark to capture traffic from a web browsing session or simulate by using packet tracer. Identify and describe the key components of an HTTP request and response. How do these components differ when browsing a secure HTTPS site?
2. Use any network analyser (such as wireshark) to set up an FTP server and client on your local network or simulate by using packet tracer. Transfer a file using FTP and then using SFTP. Compare the transfer process and the security measures in place. What differences do you observe?
3. Configure an email client with SMTP, IMAP, and POP3 settings for a given email service. Send an email and verify its receipt using IMAP and POP3. What are the key differences in how emails are handled by IMAP versus POP3?
4. How can you explore a Session layer and Presentation layer using network simulation software (such as GNS3 or Packet tracer). Explain their role in OSI model, primary functions and real world examples.

FOR ONLINE READING ONLY

Transmission media

The transmission medium is the pathway that can send data and information from the source to the destination. For example, the transmission medium for the conversation between you as a student and teacher in class is air. Also, when you have a written message such as a letter, the transmission medium can be a mail carrier, truck, or aeroplane. Figure 3.30 illustrates the transmission of data from the sender to the destination.

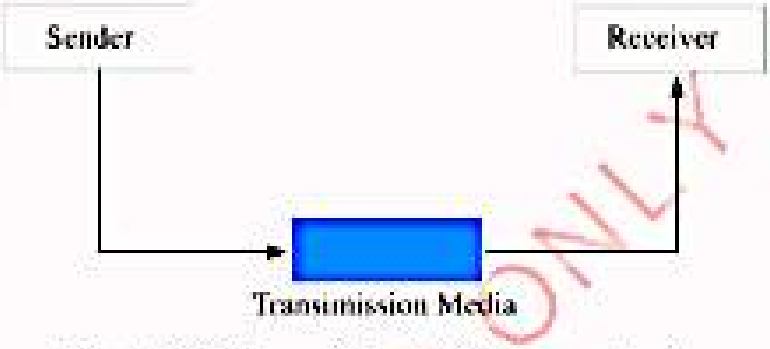


Figure 3.30: *Transmission of data from sender to receiver through a medium*

In data communication, transmission media are the links that carry messages between two or more communicating devices. Computers and other telecommunication devices use signals to represent data and information. These signals are transmitted in the form of electromagnetic energy from one device to another. The transmission medium in data communication can therefore be more specific and is usually free space, metallic cable, or fibre optic cable. Transmission media can be categorised as guided or unguided media. Figure 3.31 shows the classification of transmission media.

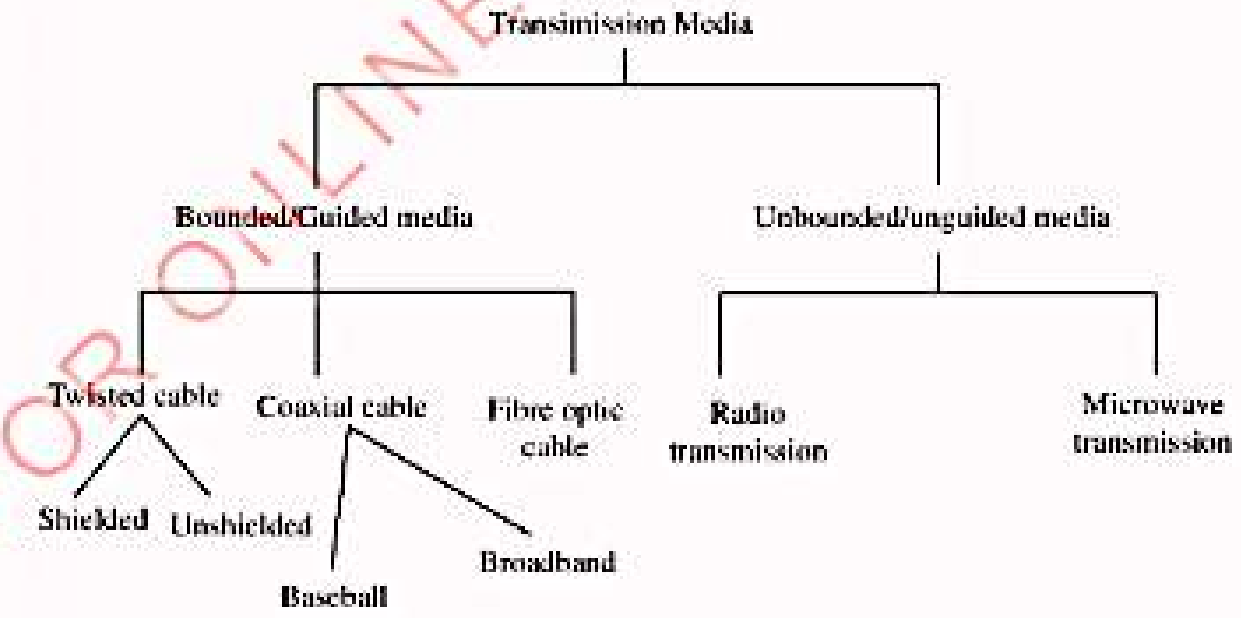


Figure 3.31: *Classification of the transmission medium*

Guided transmission media

In guided transmission media, there is a physical link made of wire/cable through which data in terms of signals are propagated between the source and destination.

FOR ONLINE READING ONLY

Another name for guided transmission media is wired media. Examples of guided media are metallic cable, fibre-optic cable, and others. Guided transmission media use a cabling system that guides signals to a specific path from the source to the destination. Examples of guided transmission are twisted pair, coaxial cable, and fibre optic cables. A twisted pair and coaxial cable transmit data in the form of electric signals, whereas optical fibre cables carry data in the form of light signals.

Twisted pair cable

Twisted pair cables, also known as Ethernet cables, are insulated cables consisting of two copper wires twisted together. Each twisted pair has a single communication link. It is twisted to minimise the effect of electrical interference from nearly similar pairs, which cancel each other. These cables are less expensive and are mostly used in telephone lines and LANs. There are two types of twisted-pair cables: Unshielded Twisted Pair (UTP) and Shielded Twisted Pair (STP). Figure 3.32 illustrates a twisted pair of cables.

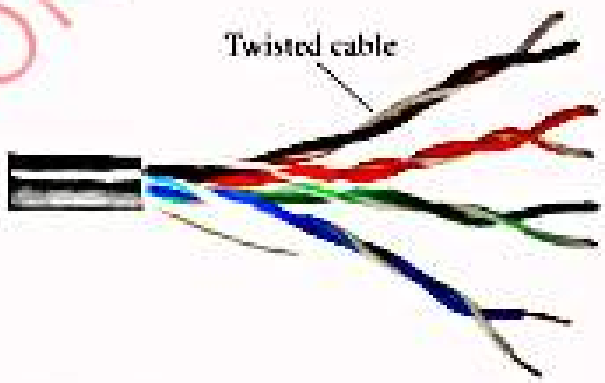


Figure 3.32: *An example of a twisted pair cable*

Unshielded twisted pair cable

Unshielded Twisted Pairs (UTP) cable comprises only wires and insulated cables. Unshielded Twisted pair cable (UTP) is the most common cable used in telecommunication compared with the Shielded Twisted Pair Cable, which consists of two conductors, usually copper, each with its colour plastic insulator. The UTP cabling is commonly found in workplaces, schools, and homes. The reason for the popularity of twisted pairs is their low cost. This cable is easy to install and has the lowest cost per meter of any cable. Figure 3.33 shows an example of an unshielded twisted pair cable.

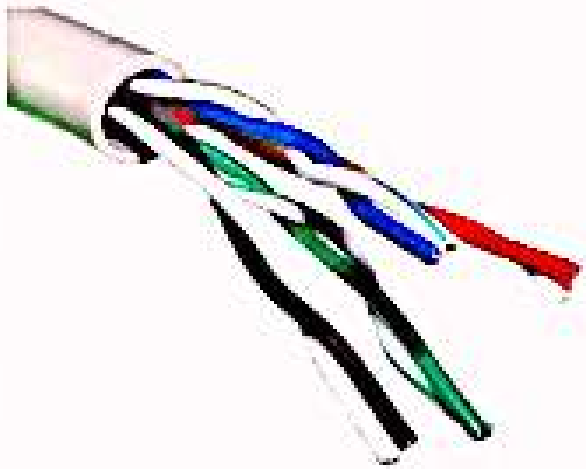


Figure 3.33: *Example of unshielded twisted*

UTP cables consist of two insulating copper wires (1mm thick). Electronic Industries Association (EIA) has developed standards for UTP cables based on their performance. UTP cables are divided into various categories based on their capability to carry higher bandwidth. Examples of UTP cables are Category 5 (Cat5), Enhanced Category 5

FOR ONLINE READING ONLY

(Cat5e) and Category 6 (Cat6) as shown in Table 3.8. Higher categories are designed to carry higher bandwidth. The choice of UTP cables depends on the intended application. The standard connector for UTP cabling is a Registered Jack 45 (RJ-45) connector.

Table 3.8: *Categories of UTP cables*

Type	Use
Category 1	Voice Only (Telephone Wire)
Category 2	Data to 4 Mbps (Local Talk)
Category 3	Data to 10 Mbps (Ethernet)
Category 4	Data to 20 Mbps (16 Mbps Token Ring)
Category 5	Data to 100 Mbps (Fast Ethernet)
Category 6	Data to gigabit connections

Advantages of Unshielded Twisted Pairs

(a) It is easy to install and maintain.

(b) It is flexible.

(c) It is cheap compared to coaxial and fibre optic cables.

(d) It has a high-speed capacity of up to 100Mbps and with category 6 it also handles gigabit connections.

Disadvantages of Unshielded Twisted Pairs

(a) It has low bandwidth when compared with coaxial cable.

(b) It has poor protection against interference/noise.

(c) It has limitations in distance.

Shielded Twisted Pair cable

Shielded Twisted Pairs (STP) cable is a cable braided with wire mesh that encases each insulated wire pair. STP cables are less affected by Electromagnet Induction (EMI) compared to UTP cables because of the metal casing. It has the same attenuation (decrease of signal strength due to length/distance of medium) as unshielded twisted pair. Shielded Twisted Pair cables are faster and more expensive than unshielded and coaxial cables. Figure 3.34 shows an example of a shielded twisted pair cable.



Figure 3.34: *Example of shielded twisted pair cable*

Advantages of Shielded Twisted Pair

(a) It is easy to install.

(b) It has adequate performance.

FOR ONLINE READING ONLY

(c) It can be used for analogue or digital transmission.

(d) It increases the signalling rate.

(e) It has a higher capacity than an unshielded twisted pair.

(f) It eliminates crosstalk.

Disadvantages of shielded Twisted Pairs

(a) It is difficult to manufacture.

(b) It is heavy.

(c) It is expensive.

Coaxial cable

Coaxial cables are commonly used by cable TV, telephone companies, and computer networks for transmitting data. It is better shielded and has more bandwidth than a twisted pair. It has a copper wire as the core, which is surrounded by insulating material as see in Figure 3.35. The coaxial cable contains two conductors that are parallel to one another. Copper is used as a centre conductor, a solid wire, or a standard one.

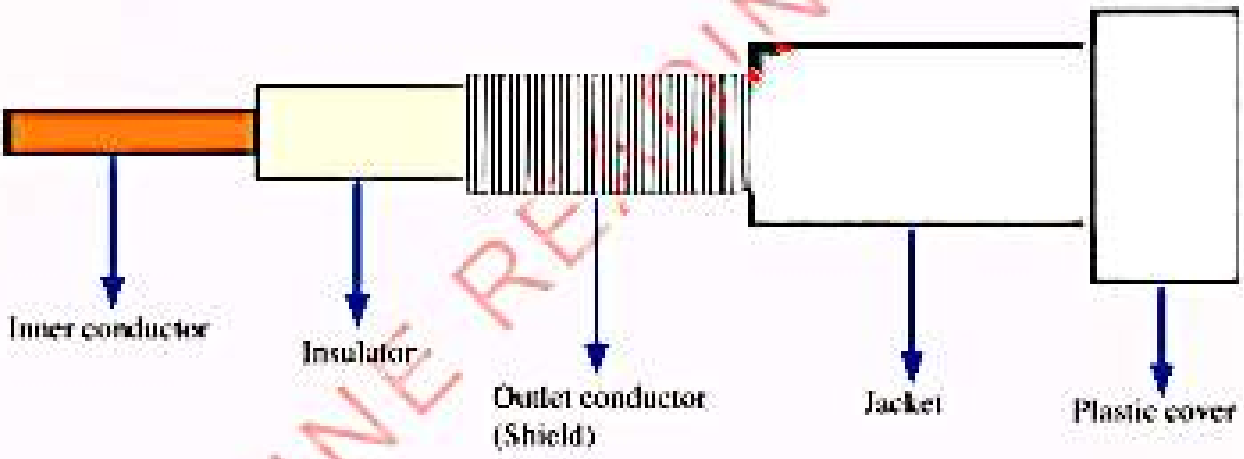


Figure 3.35: *A coaxial cable*

It is surrounded by Polyvinyl Chloride (PVC) installation encased in an outer conductor of metal foil, braid, or both. Outer metallic wrapping is used as a shield against noise and as the second conductor that completes the circuit. The outer conductor is also encased in an insulating cover; the outer part is the plastic cover which protects the whole cable. The most common coaxial standards are:

(a) 50-Ohm RG-7 or RG-11 is used with a thick ethernet.

(b) 50-Ohm RG-58 is used with athin ethernet.

(c) 75-Ohm RG-59 is used with cable television.

The key to the success of coaxial cable is its shielded design that allows the cable's copper core to transmit data quickly, without the interference of environmental factors. These types of cables are used to carry signals of higher frequencies to a longer distance

FOR ONLINE READING ONLY

Advantages of coaxial cable

(a) It is not easily affected by attenuation and electromagnetic interference.

(b) It can transmit data over longer distances than a twisted pair cable.

(c) It supports the transfer of high bandwidth signal compared to twisted pair.

(d) It supports the transmission of digital signals at a very high rate of 10Mbps.

(e) They can span a longer distance at higher speeds as they have better shielding when compared to twisted pair cables.

Disadvantages of coaxial cable

(a) It is difficult to install and more expensive than a twisted pair of cable.

(b) It is not safe because it can be taped by breaking the cable and inserting the T-joint.

(c) Single cable failure can fail the entire network.

(d) If the shield is imperfect, it can lead to a grounded loop.

Fibre Optical Cables (FOC)

Fibre optic cables are made up of very thin strands of glass or plastic. One cable may contain two strands or several hundreds of strands (see Figure 3.36). Fibre optic cable transmits data signals as light. Optic fibre uses refraction to direct the light through the media. A thin transparent strand of glass at the centre is covered with a layer of less dense glass called cladding. This arrangement is covered with an outer jacket made of PVC or Teflon. Such types of cables are usually used in backbone networks. These cables are lightweight and have higher bandwidth which means a higher data transfer rate. Signals can travel longer distances, and electromagnetic noise cannot affect the cable. However, fibre optic cables are expensive and unidirectional. Thus, two cables are required for full-duplex communication.

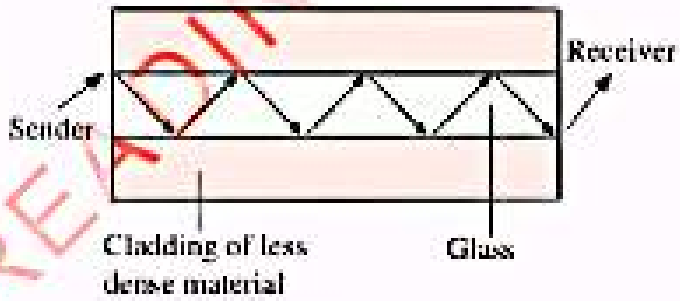


Figure 3.36: *Fibre optic cable*

Advantages of fibre optical cable

(a) It is the faster transmission media.

(b) It supports higher bandwidth.

(c) It is not affected by attenuation and electromagnetic interference.

(d) It is used in transmitting signals over longer distances.

(e) It has high security; therefore, difficult to tap the signal.

Disadvantages of fibre optical cable

(a) It is the most expensive cable because it requires costly equipment.

(b) It is very difficult to install.

(c) It is fragile because it is easier to break if bent out.

FOR ONLINE READING ONLY

Unguided/wireless transmission media

In unguided transmission, data travels in the air in the form of electromagnetic waves enabled by antennas. The antennas are used to transmit and receive signals between the source and destination. Unguided media are also known as wireless media. Examples of unguided media are wireless transmission such, as radio, microwave, infrared, and sound. Unguided media, also called wireless communication, transmits electromagnetic waves without using a physical medium. Unguided media use an antenna for transmitting through air, vacuum, or water. Unguided media information travels in the form of electromagnetic signals through the air. The electromagnetic spectrum of frequency ranges from 3 kHz to 900 THz; this range is available for wireless communication as shown in Figure 3.37.

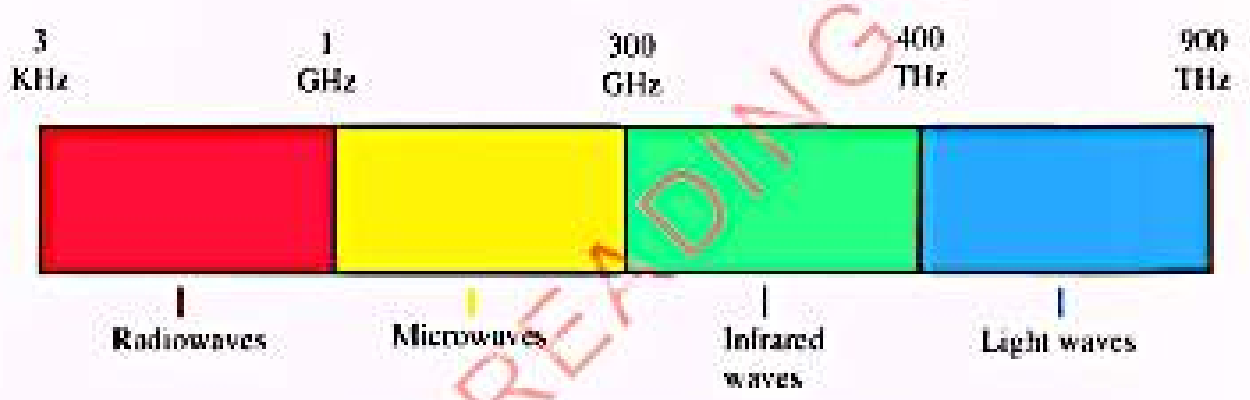


Figure 3.37: *Electromagnetic waves spectrum*

Wireless technologies allow communication between two or more devices from short to long distances without any physical media. There are many wireless communication technologies, such as Bluetooth, Wi-Fi, and WiMAX. The electromagnetic spectrum ranges from 3 kHz to 900 THz. It can be divided into four categories according to their frequency ranges, as shown in Figure 3.37. These are Radio waves, Microwaves, Infrared waves, and Light waves.

Radio waves: Radio waves can be broadcast in all directions from the source, that is, omnidirectional so that the transmitter and receiver do not have to be carefully aligned physically.

Microwave transmission: In microwave transmission, the waves travel in straight lines. It gives a much higher signal-noise ratio, but the transmitting and receiving antennas must be accurately aligned with each other.

Infrared waves: Infrared waves are widely used for short-range communication. The remote controls used on television, Video Cassette Recordings (VCRs) and stereos use infrared communication.

Communication satellite: In satellite communication, signal transferring

FOR ONLINE READING ONLY

between the sender and receiver is done with the help of a satellite. The signal, which is a beam of modulated microwaves, is sent toward the satellite. Then the satellite amplifies the signal and sends it back to the receiver's antenna present on the earth's surface. So, all the signal transferring is happening in space. Thus, this type of communication is known as space communication. Some of the properties of each wave are listed in Table 3.9.

Table 3.9: *Classification of transmission waves and their properties*

Transmission waves	**Properties**
Radio waves	(i) Waves of frequency range from 3 kHz to 1GHz. (ii) Radio waves are multi-directional, can move in all directions. (iii) Radio waves of frequency 300 kHz-30 MHz can travel long distances. (iv) They are susceptible to interference. (v) Radio waves of frequencies from 3 kHz to 300 kHz can penetrate walls. (vi) These waves are used in AM and FM radio, television, and cordless phones.
Microwaves	(i) Electromagnetic waves of frequency range from 1 GHz to 300 GHz; Unidirectional, it can move in only one direction; They cannot penetrate solid objects such as walls, hills, or mountains; They need line-of-sight propagation; that is, both communicating antennae must be in the direction of each other; They are used in point-to-point or unicast communications such as radar and satellite; Provide very large information-carrying capacity.
Infrared waves	(i) Electromagnetic waves of frequency range from 300 GHz to 400 THz; Very high-frequency waves; They cannot penetrate solid objects such as walls, hills, or mountains; Used for short-distance point-to-point communication such as mobile-to-mobile, mobile-to-printer, remote-control-to-TV, and Bluetooth-enabled devices to other devices.

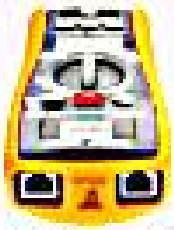
FOR ONLINE READING ONLY

Transmission waves	Properties
Communication satellite or light waves	(i) In satellite communication, signal transferring between the sender and receiver is done with the help of a satellite. (ii) The signal, which is a beam of modulated microwaves, is sent toward the satellite. Then the satellite amplifies the signal and sends it back to the receiver's antenna present on the earth's surface. So, all the signal transferring is happening in space. Thus, this type of communication is known as space communication.

Cable testing tools

Every cable run must receive a minimum level of testing. The most important test is simply determining whether the cable pairs are connected properly or not. To achieve testing, you can use cable testers which are included in Table 3.10.

Table 3.10: *Cable testers*

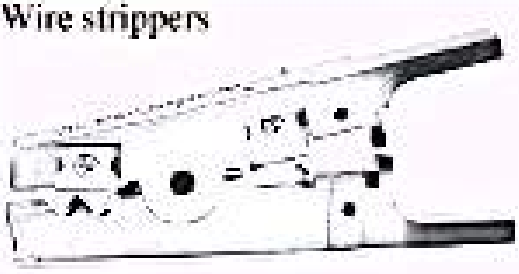
No	Tool	Description
(a)	Tone generators and amplifier probes  **Figure 3.38:** *Tone generator*	A wire-map tester that examines wire pairs to ensure they are correctly connected and checks the continuity of each wire. Figure 3.38 shows a simple tester that performs both wire-map and continuity testing.
(b)	***Continuity testers***  **Figure 3.39:** *Signal testing tool*	This tool tests if the signal reaches the far end of the cable. Simple continuity testers confirm signal reception as shown in Figure 3.39 but do not test for attenuation or crosstalk.
(c)	Wire-map testers  **Figure 3.40:** *A simple cable testing tool*	A wire-map tester checks if wire pairs are correctly connected and verifies wire continuity. Figure 3.40 shows a tester that performs both wire-map and continuity testing.

FOR ONLINE READING ONLY

Common Cabling Tools

The following tools are common to most cabling tool kits: wire strippers, wire cutters, cable crimpers, punch-down tools, fish tape, and toning tools. These tools are essential for installing cabling systems. Table 3.11 shows tools and descriptions of cabling tools.

Table 3.11: *Common cabling tools*

No	Tool	Description
(a)	Wire strippers  **Figure 3.41:** *Wire stripper*	Strippers for UTP, ScTP, and STP cables remove the outer jacket while accommodating UTP cable variations. Ensure the insulation on the conductors is not damaged. Figure 3.41 shows a wire stripper. Coaxial and fiber optic cable use different wire stripper tools
(b)	Wire cutters  **Figure 3.42:** *Wire cutter*	Use a wire cutter (Figure 3.42) to snip coaxial and twisted-pair cables without damaging them. Special scissors are available for cutting fiber-optic cables
(c)	Cable crimpers  **Figure 3.43:** *Crimping tool*	Crimpers, like the one in Figure 3.43, attach RJ-45 and coaxial connectors to cable ends without damage, ensuring strong contact and avoiding wasted time and materials
(d)	Punch-down tools  **Figure 3.44:** *Punching tool with adjustable impact settings*	The punch-down tool in Figure 3.44 inserts and trims cable ends into 66-blocks, 110-blocks, or patch panels

FOR ONLINE READING ONLY

No	Tool	Description
(e)	Fish Tapes  Figure 3.45: *Fish tape*	A fish tape (Figure 3.45) is a flexible wire, steel tape, or fibreglass rod used to clear obstacles in inaccessible areas like walls or conduits, allowing easy cable installation. In fibre networks, a fiberglass pushrod is typically used instead.

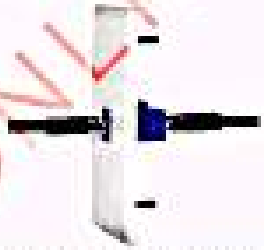
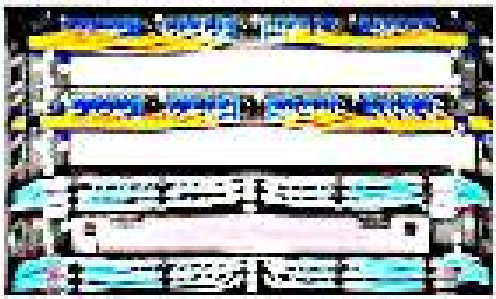
Cable-toning tool

A cable toner is a device that determines if the fundamental cable installation has been done properly. The toner is a simple continuity tester that confirms that what is connected at one end is electrically continuous to the other end. An electrical signal, or tone, is injected on the circuit being tested and is either received and verified on the other end or looped back for verification on the sending end.

Cable termination tools

Termination of the cable is done on patch panels, cross-connections by using wall plates as shown in Table 3.12.

Table 3.12: *Cable termination tools*

No	Tool	Description
(a)	Wall plate  Figure 3.46: *A plate to mount on a wall*	A wall plate show in Figure 3.46 is a flat plastic or metal plate mounted on a wall or surface, containing one or more jacks. A jack in the wall plate allows a workstation to connect to the network cabling system
(b)	Patch panel  Figure 3.47: *Patch panel to connect wires*	A patch panel as shown in Figure 3.47 connects individual wires to network ports on the front, which can then link to network equipment or other ports with a patch cable. Patch panels are cost-effective and offer great flexibility for connecting workstations.

FOR ONLINE READING ONLY

No	Tool	Description
(c)	Patch cords  Figure 3.48: *Patch cord to connect cables*	Patch cords connect field-terminated horizontal cables to network devices like switches and hubs, and link telecommunications outlets to devices like computers and printers.

Cabling pathways (Conduit)

Conduit can be metallic or nonmetallic pipe, rigid or flexible (as permitted by the applicable electrical code), and it runs from a work area to a telecommunications room and a telecommunications room to an equipment room.

(a) Cable trays

As an alternative to conduit, *cable trays* can be installed to route your cable. Cable trays are typically wire racks specially designed to support the weight of a cable infrastructure. They provide an ideal way to manage a large number of horizontal runs.

(b) Raceways

Raceways are conduits for surface-mounting horizontal cables, pieced together modularly with connectors that maintain the minimum bend radius. They are mounted on walls where internal installation is difficult, such as brick or concrete walls without telecommunications conduit.

Cable installation and termination guidelines

The following are some of the guidelines for installing cables:

(a) Allow extra cable: Use more cable than you need and leave plenty of slack.

(b) Test each segment: Test every part of the network as you install it. New cables can still have problems that are difficult to isolate later.

(c) Use proper tools: Use the recommended crimping and cutting tools to avoid damaging the cables.

(d) Avoid interference: Keep cables at least 3 feet away from fluorescent light fixtures and other sources of electrical interference.

(e) Secure cables: Ensure all trailing wires or cables are securely fixed to avoid disturbing users.

(f) Protect floor cables: If cables must run across the floor, cover them with cable protectors.

(g) Label cables: Label both ends of each cable for easy identification.

FOR ONLINE READING ONLY

(h) Group cables properly: Use cable ties, not tape, to group cables in the same location together.

Activity 3.11:

Straight-through cable construction

Aim:

To build and test a functional Ethernet patch cable, ensuring proper construction and connectivity.

Materials:

Stranded Cat 5, 5e, or 6 UTP cable; RJ-45 plugs; Crimping tool; Cable stripper; Cable tester; Measuring tape; and Wire cutters OR Computer and Simulation software (Such as Cisco Packet Tracer, GNS3)

Instructions

Use the following steps (1 to 10) to:

(a) Build: Construct a Category 5, 5e, or 6 (CAT 5, 5e, or 6) Unshielded Twisted Pair (UTP) Ethernet network patch cable or patch cord.

(b) Test: Ensure the cable has continuity, correct pinouts, and the correct wire color on the right pin.

Step 1: Determine length

(a) Measure distance: Measure the distance between devices or to the plug, adding 30.48 cm (12 in.) for flexibility.

(b) Ensure proper length: Ensure the cable length does not exceed 3 m (9.84 ft). Standard lengths are 1.83 m (6 ft) and 3.05 m (10 ft).

Step 2: Cut cable

Cut to Length: Cut a piece of stranded Cat UTP cable to the desired length. Use stranded cable for patch cables as it is more durable.

Step 3: Strip cable

Remove jacket: Strip 5.08 cm (2 in.) of the jacket off one end of the cable.

Step 4: Organise pairs

Arrange Pairs: Arrange the twisted pairs according to the T568-B wiring standard, maintaining as much twist as possible for noise cancellation.

Step 5: Reorder pairs

Match color scheme: Untwist and reorder the green and blue pairs to match the T568-B color scheme. Continue with the remaining pairs.

Step 6: Prepare wires

Align wires: Flatten, straighten, and align the wires. Trim them to 1.25 to 1.9 cm (1/2 to 3/4 in.) from the jacket. Keep the jacket and wires in place to minimize noise.

Step 7: Attach RJ-45 Plug

Insert plug: Place an RJ-45 plug on the cable with the prong underside and the orange pair on the left. Push the plug until the copper ends are visible through the plug.

FOR ONLINE READING ONLY

Step 8: Crimp plug

Secure wires: Ensure the jacket is inside the plug for stress relief. Crimp the plug to secure the wires and complete the connection.

Step 9: Repeat process

Second End: Repeat steps 3 to 8 for the other end of the cable, using the same wiring scheme.

Step 10: Test cable

Verify functionality: Test the cable with a cable tester. Have the instructor check the cable to ensure it functions properly.

Deliverables

(a) Completed cable: A fully functional, properly constructed CAT 5, 5e, or 6 Ethernet patch cable.

(b) Test results: Documentation of the cable test results, confirming correct continuity and pinouts.

(c) Instructor verification: Approval from the instructor confirming the cable's proper functionality.

(d) Written report including relevant diagrams and screenshorts if using simulation.

Put all your documents in a portifolio.

Activity 3.12:

Crossover Cable Construction

Aim:

To construct and test a functional Ethernet crossover cable using T568-A and T568-B wiring standards.

Materials:

Stranded CAT 5, CAT 5e, or CAT 6 UTP cable; RJ-45 plugs; Crimping tool; Cable stripper; Cable tester; Measuring tape; and Wire cutters OR Computer and Simulation software (Such as Cisco Packet Tracer, GNS3)

Instructions

Use the following steps (1 to 10) to:

(a) Build: Create a Category 5 (CAT 5), Category 5e (CAT 5e), or Category 6 (CAT 6) Unshielded Twisted Pair (UTP) Ethernet crossover cable.

(b) Test: Ensure proper continuity, correct pinouts, and wire placement.

Step 1: Wire standards

Wire one end of the cable to the T568-A standard and the other end to the T568-B standard to cross the transmit and receive pairs.

Step 2: Measure length

Measure the required length between devices, adding 30.48 cm (12 in.) for flexibility. Standard lengths are typically 1.83 m (6 ft) or 3.05 m (10 ft).

Step 3: Cut cable

Cut a piece of stranded UTP cable to the measured length.

FOR ONLINE READING ONLY

Step 4: Strip cable

Strip approximately 5.08 cm (2 in.) of the outer jacket off one end of the cable.

Step 5: Organise pairs

Arrange the twisted pairs according to the T568-B wiring standard, ensuring to maintain the twists for effective noise cancellation.

Step 6: Untwist and reorder pairs

Untwist the green and blue pairs and reorder them to match the T568-B color scheme. Repeat for the remaining pairs.

Step 7: Prepare wires

Flatten, straighten, and trim the wires to about 1.25 cm to 1.9 cm (1/2 to 3/4 in.) from the jacket edge.

Step 8: Attach RJ-45 plug

Insert the wires into an RJ-45 plug, ensuring the prongs are facing down and the jacket is properly secured inside the plug.

Step 9: Crimp plug

Use a crimping tool to secure the RJ-45 plug onto the cable, ensuring all wires are properly terminated.

Step 10: Repeat for Other End

Repeat steps 4 to 9 for the other end of the cable, following the T568-A wiring standard.

Step 11: Test cable

Use a cable tester to verify the continuity and correct pinouts of the crossover cable. Ensure each wire is connected to the appropriate pin.

Deliverables:

(a) Functional cable: A fully constructed Ethernet crossover cable that meets T568-A and T568-B standards.

(b) Test results: Documentation showing successful testing of continuity and pin configurations.

(c) Instructor Verification: Approval from the instructor confirming the cable's functionality and adherence to standards.

(d) Written report including relevant diagrams and screenshorts if using simulation.

Put all your documents in a portifolio

Designing Local Area Network – LAN design

LAN design basics

LAN design involves selecting appropriate network topologies (such as star, bus, or ring), ensuring proper cable management and routing, and choosing suitable network devices (like switches and routers) to optimise performance, scalability, and reliability.

FOR ONLINE READING ONLY

The following guidelines describe the basic steps toward designing a computer network, with more attention on the local Area Network (LAN):

1. **Identifying the client's business and technical goals.**

   The first step in designing a LAN (Local Area Network) is to establish and document the goals of the design. The design should aim at addressing the following goals:-

   (a) *Functionality*: The network must allow users to meet their job requirements, and provide user-to-user and user-to-application connectivity, with reasonable speed and reliability.

   (b) *Scalability*: The network must be able to grow, that is, the initial design should grow without any major changes to the overall design.

   (c) *Adaptability*: The network must be designed with an eye toward future technology and should include no elements that would the limit implementation of new technology as they become available.

   (d) *Manageability*: The network should be designed to facilitate network monitoring and management to ensure ongoing stability of operation

   (e) *Technical goals*: The goals that the client wants to achieve after the network is complete. Technical goals are needed so as to enable you to design a network that meets users' needs. Technical design goals include network availability, scalability, performance, and security.

2. **Assess existing infrastructure:**

   (a) Evaluate the client's current environment, applications, and necessary upgrades.

   (b) Develop network diagrams and identify key equipment (routers, switches, firewalls, cabling).

   (c) Document server names, addresses, cabling types, and lengths.

3. **Logical network design:**

   (a) Focus on data flow and select a logical network topology.

   (b) Design a scalable, redundant, high-speed multilayer switching LAN infrastructure.

   (c) Choose interior routing protocols.

4. **Document and propose:**

   Write a proposal and present the design to the client.

Creating network topologies using a '*Packet Tracer simulator*"

Network topologies refer to the physical or logical layout of nodes (devices) and the connections between them in a computer network. Different topologies

FOR ONLINE READING ONLY

determine how data flows within a network and how devices communicate with each other. Creating network topologies using a simulator like *Cisco packet tracer* is a great way to practice and visualise how different devices and connections interact within a network.

Activity 3.13:

Creating a star topology using Cisco Packet Tracer

Aim: To design and simulate a Star Topology network configuration using Cisco Packet Tracer, demonstrating proficiency in network layout and device connectivity.

Materials: Computer, Cisco Packet Tracer software, Internet connection for accessing Packet Tracer resources

Instructions

Step 1: Launch Cisco Packet Tracer

Open the Cisco Packet Tracer application on your computer, interface as shown in Figure 3.49 will appear.

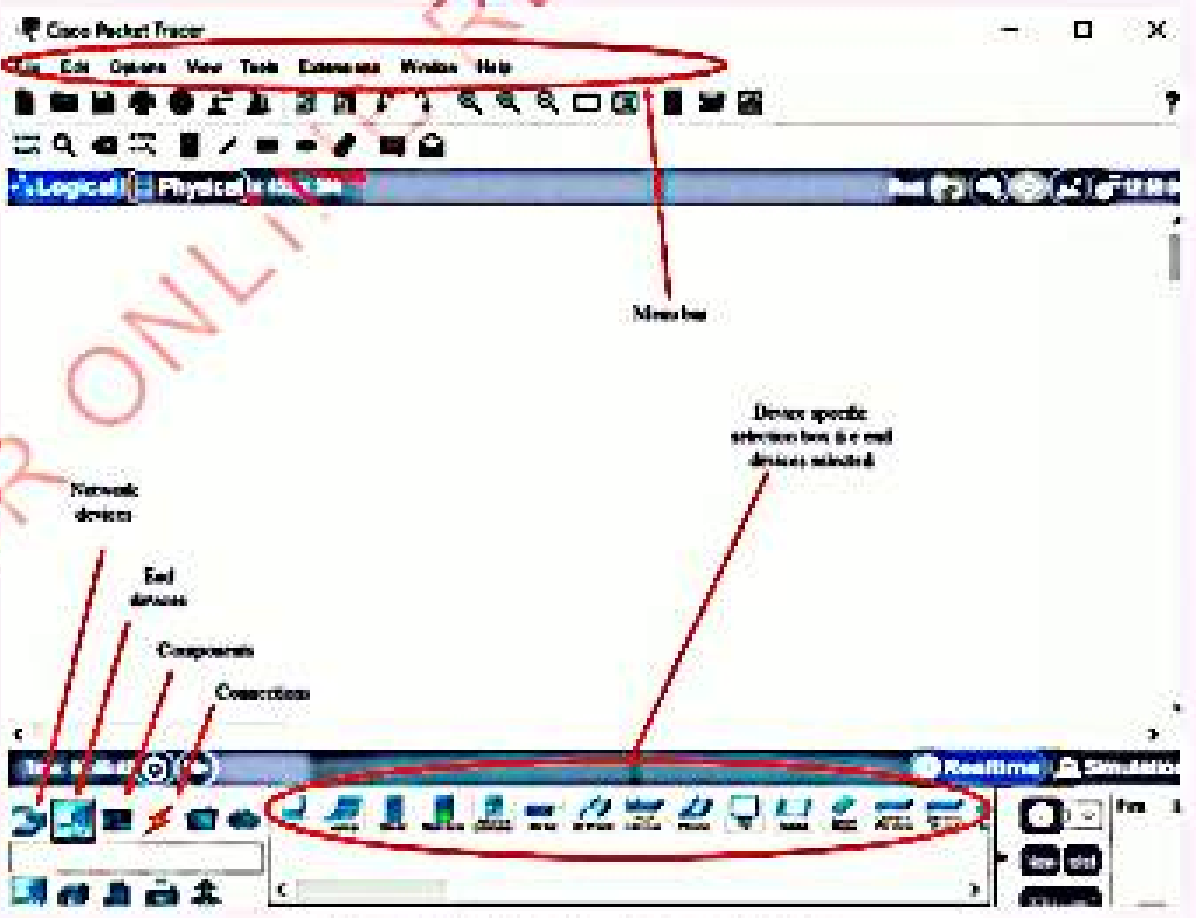


Figure 3.49: *Cisco Packet tracer interface*

FOR ONLINE READING ONLY

Step 2: Add devices

(a) Select "End Devices" as in Figure 3.49 and "Switches" from the device-type selection box as in Figure 3.50.

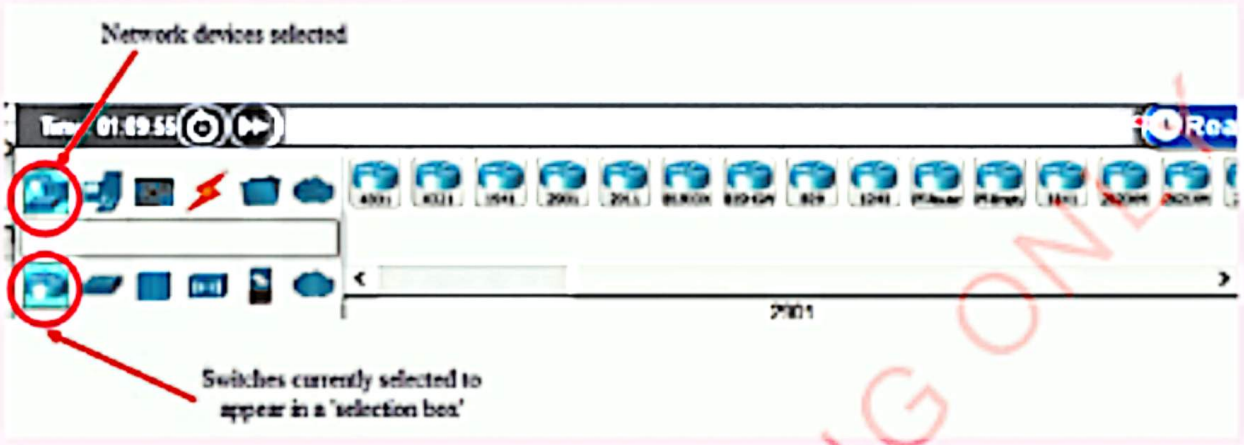


Figure 3.50: *Network devices-Switches selected*

(b) Drag and drop PCs (such as., PC0, PC1, PC2) and a switch (such as 2911 switch) onto the workspace as shown in Figure 3.51.

Move the cursor into workspace area. You will notice it turns into a plus "+" sign. Single click in the topology area and it copies the device. Add three more hosts.



Figure 3.51: *Single clicking topology area*

FOR ONLINE READING ONLY

Step 3: Connect devices

(a) Click on the "Connections" icon (lightning bolt) to select suitable cables (such as copper straight-through) as shown in Figure 3.52.



Figure 3.52: *Connections and connection type*

(b) Connect each PC to available ports on the switch using the chosen cables. Perform the following steps to connect PC2 to Switch0:

(i) Click once on *PC2*

(ii) Choose *FastEthernet*

(iii) Drag the cursor to *Switch0*

(iv) Click once on *Switch0* and choose *FastEthernet0/1*

(v) Notice the green link lights on *PC2* Ethernet NIC and amber light *Switch0* FastEthernet0/1 port. The switch port is temporarily not forwarding frames, while it goes through the stages for the Spanning Tree Protocol (STP) process.

(vi) After about 30 seconds the amber light will change to green indicating that the port has entered the forwarding stage as shown in Figure 3.53. Frames can now be forwarded out the switch port.

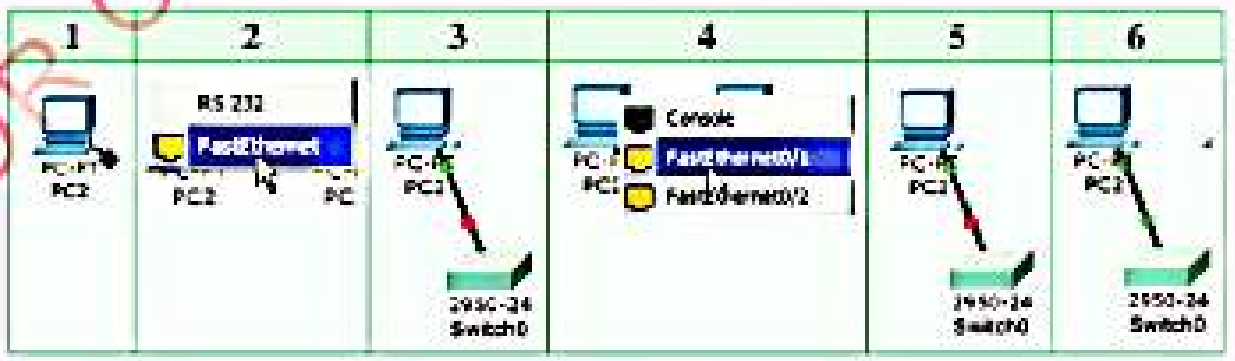


Figure 3.53: *Port forwarding stage*

Repeat the steps (i-vi) above for PC3 connecting it to Port 3 on Switch0 on port FastEtherent0/2. (The actual switch port you choose does not matter).

FOR ONLINE READING ONLY

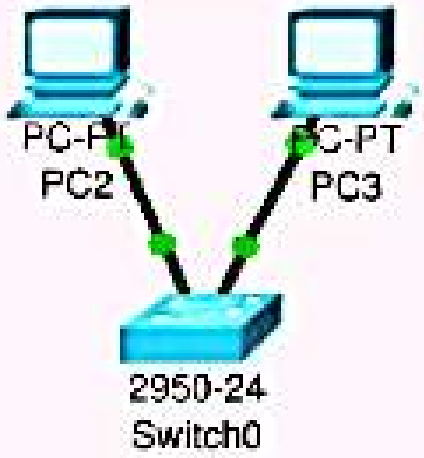


Figure 3.54: *Choosing switch port*

Move the cursor over the link light to view the port number. *Fa* means *FastEthernet*, 100 Mbps Ethernet.

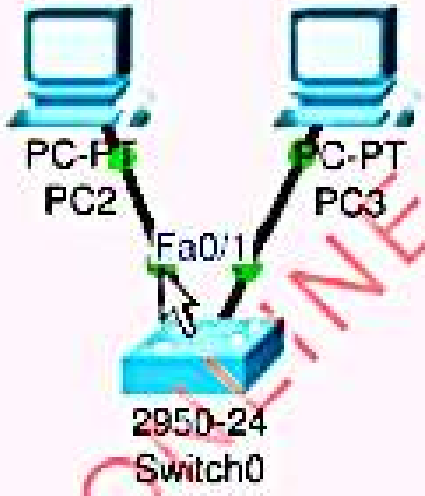


Figure 3.55: *Moving the cursor over link light*

Step 4: Verify connections

Check link lights on the switch and PCs to ensure all connections are active and properly configured.

Step 5: Assign IP addresses on each PC

(a) Click on a PC, go to the "Desktop" tab, and select "IP Configuration."

(b) Assign IP addresses (Such as PC0: 192.168.1.2, PC1: 192.168.1.3, PC2: 192.168.1.4) with subnet masks (usually 255.255.255.0).

Step 6: Test connectivity

Use the "Command Prompt" on each PC to ping other PCs' IP addresses for connectivity verification (such as from PC0, ping 192.168.1.3).

Step 7: Save your topology

Save the Packet Tracer project with a descriptive name for future reference and sharing.

Deliverables:

(a) Configured star topology: A visual representation of the network layout showing connected devices and active links.

(b) IP configuration documentation: Detailed notes documenting assigned IP addresses and subnet masks for each PC.

(c) Connectivity test results: Record of successful pings between PCs to demonstrate network functionality.

Network security principles and protocols in computer networks

Introduction to network security

Security is a critical concern, both physically and in computer systems. Despite advanced security measures, computers are more vulnerable due

FOR ONLINE READING ONLY

to global Internet access. As we move towards an information society, network security becomes more important, especially with the rise in cybererime and legal challenges.

(a) Meaning of network security

Network security protects networks and their services from unauthorized access, modification, destruction, or disclosure. It ensures that networks perform their critical functions correctly without harmful side effects. Network privacy, a part of network security, focuses on protecting user, application, device, and network data from unauthorized access. Most security issues are caused by malicious individuals trying to gain something, get recognition, or harm others.

(b) Goals of network security

Network security aims to protect not only the computers at both ends of communication but also the entire network. Whether an organisation is small or large, prioritising network security is essential when using the Internet, LAN, or other connectivity methods. The goals of network security include maintaining:

(i) *Confidentiality:* Protecting data, objects, and resources from unauthorised access or viewing.

(ii) *Integrity:* Ensuring the accuracy and consistency of data, preserving its reliability and preventing unauthorized alterations.

(iii) *Availability:* Ensuring that data, network resources, or services are consistently accessible to legitimate users whenever needed.

(c) Network security measures

Network security uses multiple layers of protection, both at the network perimeter and within its infrastructure. Each security layer implements specific strategies and follows defined policies to limit access to network resources to authorized users and prevent unauthorized activities. Effective network security involves checking and protecting different parts of the network, making sure that all the equipment, programs, and rules work together smoothly.



Figure 3.56: *Security feature for networks*

(i) *Physical security networks*

Physical security networks are designed to prevent unauthorised access to various physical network devices, such as routers and cabling cabinets. Every institution or organisation must also implement controlled access measures, such as locks and biometric passwords, to ensure security.

FOR ONLINE READING ONLY

(ii) *Technical security networks*

Technical security controls protect the information transmitted over the network, ensuring both incoming and outgoing data is safeguarded. This protection is twofold: it defends information and devices from unauthorised access, while also preventing unauthorised actions from within the organisation's workforce.

(iii) *Administrative network security*

Administrative network security controls manage end-user behavior, encompassing authentication, access levels, and the implementation of infrastructure reforms by an organization's IT staff. Administrative security involves the establishment of various security policies and processes to ensure its effectiveness.

(d) Importance of network security

The following are some of the importance of network security:

(i) It helps to reduce the risk of data loss, theft, and sabotage,

(ii) It helps to protect client data,

(iii) It helps to protect your workstation from harmful spyware,

(iv) It ensures that shared data is kept secure.

(e) Types of threats that can compromise network security

The following are common threats that attackers can use to penetrate your network:

(i) Unauthorised access

Unauthorised access refers to attackers accessing a network without receiving permission. Among the causes of unauthorised access attacks are weak passwords, lacking protection against social engineering, previously compromised accounts, and insider threats.

(ii) Distributed Denial of Service attacks

Distributed Denial of Service (DDoS) attacks is the network-based attempts to make a website, a service, or a complete infrastructure unavailable, typically by simultaneously attacking a victim from several compromised systems.

(iii) Man-in-the middle attacks

A man-in-the middle attack involves attackers intercepting traffic between your network and external sites or within your network. If communication protocols are not secured, attackers find a way to evade that security; they can steal data transmitted, obtain user credentials, and hijack their sessions.

(iv) Code and SQL injection attacks

Many websites accept user inputs and fail to validate and sanitise those inputs. Attackers can then fill out a form or make an Application Programming Interface (API) call, passing malicious code instead of the expected data values. The code is executed on the server and allows attackers to compromise it.

(v) Privilege escalation

Once attackers penetrate your network, they can use privilege escalation to

FOR ONLINE READING ONLY

expand their reach. Horizontal privilege escalation involves attackers gaining access to additional, adjacent systems, and vertical escalation means attackers gain a higher level of privileges for the same systems.

(vi) Insider threats

A network is especially vulnerable to malicious insiders who already have privileged access to organizational systems. Insider threats can be challenging to detect and protect against because insiders do not need to penetrate the network to harm.

Common network security vulnerabilities

(i) DNS hacking attacks

The Domain Name System (DNS) server is a distributed, hierarchical, and global directory that converts domain names into numerical IP addresses. As a critical infrastructure component, all hosts rely on DNS to access servers and initiate connections. Normally, hosts send queries to DNS servers, which either reply with the correct information or forward the queries to more authoritative servers. Besides translating host addresses, DNS servers also store other information.

Name-resolution services are crucial for email transmission, web navigation, and data transfer in the modern Internet. Consequently, DNS attacks can impact a large segment of the Internet, potentially compromising data authenticity and integrity.

(ii) Routing table poisoning attacks

A routing table poisoning attack involves the unauthorized modification of routing tables. Attackers can achieve this by altering the routing information update packets sent by routers. This is a significant issue because routing tables are fundamental to Internet routing. Any incorrect entry can lead to severe consequences such as network congestion, host overload, routing loops, unauthorized data access, and network partitioning.

(ii) Packet-mistreatment attacks

Packet-mistreatment attacks can occur during data transmission, where a hacker intercepts and manipulates certain data packets. These attacks are challenging to detect and can cause network congestion, reduced throughput, and denial-of-service (DoS) attacks.

(iii) Denial-of-Service attacks

Denial-of-service (DoS) attacks are security breaches that prevent users from accessing services they normally use. Although DoS attacks do not result in data theft or loss, they can be highly disruptive, potentially costing the target significant time and money. DoS attacks typically target specific network services like email or DNS. For instance, attackers might overwhelm a DNS server, rendering it inoperable by causing a buffer overflow or sending excessive queries.

DoS attacks are relatively easy to launch but difficult to detect. They can disable

FOR ONLINE READING ONLY

critical servers for several hours, denying service to all users. Various methods, such as UDP floods, TCP floods, and ICMP floods, can initiate these attacks. The primary goal of the attacker is to overwhelm the victim and disrupt the provided services.

Ways to secure network

The following measures are to be taken to secure the network:

(a) A strong firewall and proxy are to be used to keep unwanted people out.

(b) Use a Virtual Private Network (VPN).

(c) A strong antivirus and Internet security software package should be installed.

(d) For authentication, use strong passwords and change them frequently, such as on a weekly or monthly basis.

(e) When using a wireless connection, use a strong password.

(f) Employees should be cautious about physical security.

(g) Prepare a network analyser or network monitor and use it when needed.

(h) Implementation of physical security measures like closed-circuit television for entry areas and restricted zones.

(i) Security barriers to restrict the organisation's perimeter.

(j) Fire extinguishers can be used for fire-sensitive areas like server rooms and security rooms.

Firewalls

A firewall is a system that prevents unauthorised access to or from a private network. It examines each message entering and leaving the network and allows only those authorized messages to pass through. It can be implemented in hardware, software, or both. A firewall helps to keep your computer more secure. It restricts information that comes to your computer from other computers, giving you more control over the data on your computer and providing a line of defence against people or programs.

Types of firewalls:

(a) *Application gateways* are the first generation of firewalls, also known as proxy gateways or application gateways, and provide application-level control over network traffic

(b) *Packet filtering* is a technique whereby routers have Access Control Lists (ACLs) switched on. By default, a router will pass all traffic sent to it without any sort of restrictions.

(c) *Hybrid systems* merge the security of the application layer gateways with the flexibility and speed of packet filtering; some vendors have created systems that use both principles.

Importance of firewall

(a) It allows only authorised access inside a network,

(b) It prevents insider attacks on critical systems.

(c) A firewall acts as a barrier that checks information coming from the Internet or a network and allows

FOR ONLINE READING ONLY

Antivirus and antimalware software

A computer virus is a malicious software program loaded onto a user's computer without knowledge and performs malicious actions. These are man-made harmful programs. They are man's attempts to sabotage data, and they are created merely to disturb the normal functioning of a computer system. These viruses have names, and some examples include Melissa, Trojan Horse, BadTrans, and WThus.

Network Access Control

Network Access Control (NAC) verifies and enforces the security compliance of devices seeking to connect to a network, ensuring that only authorised and properly configured devices can access network resources.

Encryption

Encryption converts sensitive data into unreadable code, protecting it from unauthorised access during transmission or storage. Secure Sockets Layer (SSL) and Transport Layer Security (TLS) are common encryption protocols used to secure data transmitted over networks.

Security Information and Event Management

Security Information and Event Management (SIEM) systems collect, analyse, and correlate security event data from various network devices and applications to identify and respond to security threats in real-time.

Importance of computer network security

The significance of computer network security cannot be overstated in today's digital landscape, where interconnected systems and the continual exchange of sensitive information are prevalent. It serves a pivotal role in protecting data and privacy from a myriad of threats and potential vulnerabilities within networked environments. Network security has the following importance:

(a) *Protecting against cyber threats:* Network security tools like firewalls, intrusion detection systems, and antivirus software help keep computer systems safe from harmful online attacks. These attacks, like viruses, scams, and attempts to shut down networks, can put important information at risk and disrupt how a network works.

(b) *Keeping data safe:* Network security makes sure that private information stays private and is only available to the right people. Encryption methods are used to keep data safe when it is being sent, stopping others from listening in or getting into the information without permission.

(c) *Keeping data accurate:* Network security ensures that information stays accurate and unchanged during transmission or torage. If someone changes the data, it can result in incorrect decisions, financial losses or even hazardous conditions in critical systems.

FOR ONLINE READING ONLY

(d) *Stopping unwanted access:* Network security sets up rules to control who can get into the network and see sensitive information. This stops people who should not be there from getting in, lowering the chances of information getting stolen.

(e) Keeping business going: Good network security helps a business keep running smoothly, even if there is a cyberattack. It means important services and resources stay available, even during tough times.

(f) Protecting personal info: In today's digital world, personal information like your name, address, and bank details can be taken without permission.

(g) Following the rules: Different places have rules about how data should be protected and used. Following these rules is important to avoid getting in trouble and to keep people's trust.

Activity 3.14:

Fostering network awareness at Miakuja Secondary School

Imagine you have been assigned the role of Network Security Administrator at Miakuja Secondary School. Outline your strategy for educating students, teachers, and staff about cybersecurity best practices and emphasing the importance of safeguarding personal and school information from online threats.

Exercise 3.10

1. How do the principles of confidentiality, integrity, and availability (CIA triad) guide the development and implementation of network security policies within an organisation?
2. Using examples, explain how preventive, detective, and corrective controls function within a network security model to protect against potential threats.
3. What are the distinguishing characteristics of malware, phishing, and denial-of-service attacks, and how do these threats specifically exploit network vulnerabilities?
4. How can misconfigured network devices and weak authentication mechanisms contribute to security breaches, and what steps can be taken to mitigate these vulnerabilities?

Public Switched Telephone Networks and Mobile telephone system

Introduction to Public Switched Telephone Network

Public Switched Telephone Networks (PSTN) are the traditional circuit-switched networks used for voice communication. They are a global system of interconnected, voice-oriented public telephone networks,

FOR ONLINE READING ONLY

both commercial and government-owned. The PSTN system is primarily designed for voice communication, although it also supports data transmission through modems.

Public Switched Telephone Network

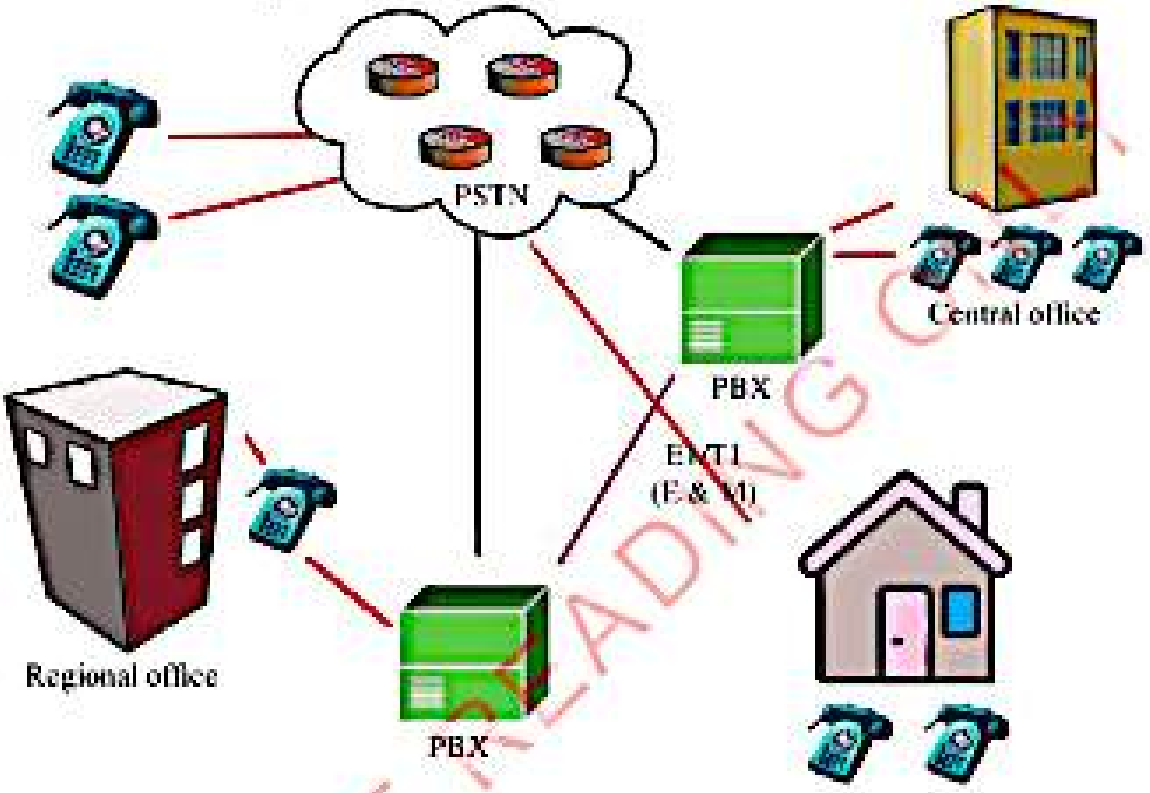


Figure 3.58: *Public Switched Telephone Network*

Key components of PSTN

(a) Local loop:

The physical link or circuit that connects from the demarcation point of the customer premises to the edge of the carrier or telecommunications service provider's network. Figure 3.59 shows a demarcation point.



Figure 3.59: *Demarcation point*

FOR ONLINE READING ONLY

(b) **Central Office (CO):** A facility that houses the switching equipment and connects calls between local loops.

(c) **Switching centers:** Equipment that routes calls based on the phone number dialed. They include local exchanges, tandem switches, and international gateways.

(d) **Transmission media:** Includes copper wires, fiber optic cables, and microwave or satellite links that transmit voice and data signals.

Figure 3.60 shows a telephone and a telephone switch. The telephone is located in the Customer's Premises (CP, CPE), while the telephone switch is located in the Central Office (CO).

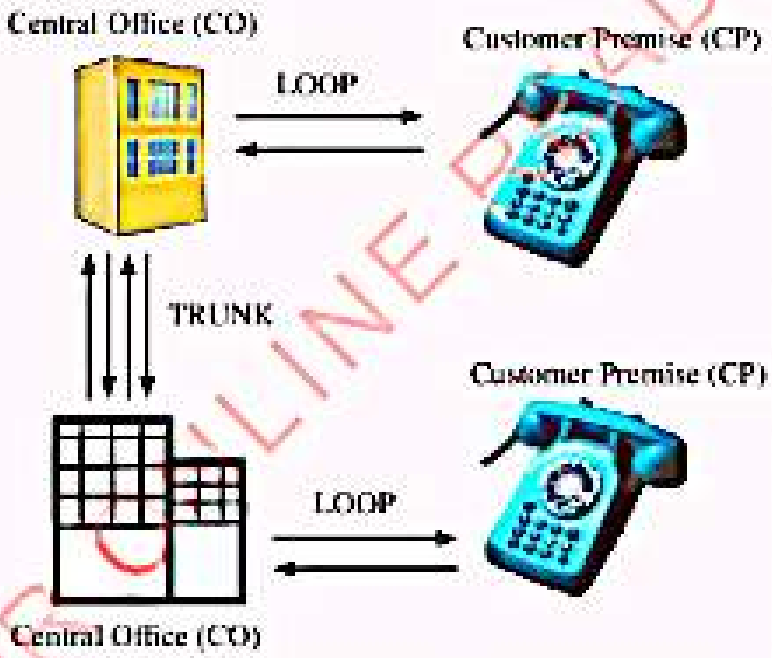


Figure 3.60: *Customer and Central Office communication*

Implementation of PSTN

(a) *Call initiation:* When a call is made, the analog voice signal is converted into a digital signal.

(b) *Routing:* The call is routed through a series of switches in the central offices.

(c) *Connection establishment:* A dedicated circuit or path is established between the caller and the recipient for the duration of the call.

(d) *Call termination:* Once the call is complete, the circuit is disconnected, freeing up the resources.

Features of PSTN

(a) *Reliable and ubiquitous:* PSTN offers reliable voice communication services and extensive global reach.

(b) *Quality of Service (QoS):* Provides a high-quality voice communication experience due to its dedicated circuit-switching approach.

(c) *Emergency services:* PSTN supports emergency call services like 911, providing location information to emergency responders.

Limitations of PSTN

(a) *Limited data Transmission:* PSTN is not optimised for high-speed data transmission compared to modern digital communication systems.

(b) *Scalability issues:* Scaling PSTN infrastructure can be costly and complex.

FOR ONLINE READING ONLY

(c) *Resource Intensive:* Establishing a dedicated circuit for each call is resource-intensive compared to packet-switched networks.

Evolution and Integration

(a) *ISDN (Integrated Services Digital Network):* An evolution of PSTN that allows the transmission of voice and data over the same line, offering higher data rates.

(b) *Transition to VoIP:* Many PSTN systems are being integrated or replaced with Voice over Internet Protocol (VoIP) systems that offer more flexibility and efficiency in handling voice and data.

Mobile telephone system and network protocols

The mobile telephone system is a wireless communication network that allows users to make voice calls, send text messages, and access data services on the go. It has evolved significantly since its inception, moving from analog to digital and now to advanced 5G networks.



Figure 3.61: *Mobile telephone system structure*

Key Components of mobile telephone system

(a) *Mobile Stations (MS):* These are the devices used by end-users, such as mobile phones and tablets.

(b) *Base Transceiver Stations (BTS):* These are the antennas and equipment that communicate directly with mobile devices.

FOR ONLINE READING ONLY

(c) *Base Station Controllers (BSC):* They manage multiple BTSs, handling resource allocation and mobility management.

(d) *Mobile Switching Centers (MSC):* These are the central nodes that manage call routing, handovers, and interface with other networks.

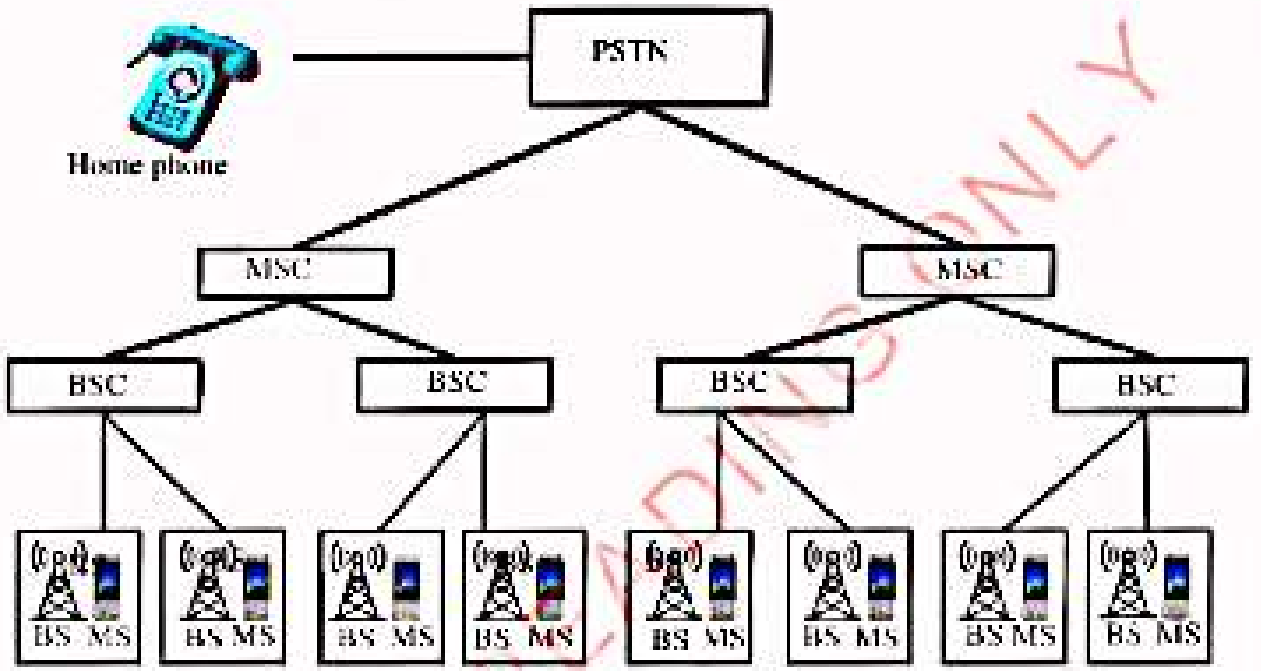


Figure 3.62: *Mobile system infrastructure*

Implementation of mobile telephone system

(a) *Cell structure:* The coverage area is divided into cells, each served by a BTS. This structure allows for frequency reuse and efficient coverage.

(b) *Handover:* When a user moves from one cell to another, the call is transferred seamlessly between BTSs, managed by the BSC and MSC.

(c) *Call setup and routing:* Calls are set up by the MSC, which connects the call to the correct destination, either within the mobile network or to other networks like the PSTN.

Evolution

(a) *1G (Analog):* Basic voice communication with limited capacity and security.

(b) *2G (GSM):* Digital communication, improved voice quality, text messaging, and basic data services.

(c) *3G:* Enhanced data capabilities, supporting mobile Internet and video calls.

(d) *4G (LTE):* High-speed Internet, better quality of service, and support for a wide range of applications.

(e) *5G:* Ultra-fast Internet, low latency, massive connectivity for IoT devices, and enhanced mobile broadband.

FOR ONLINE READING ONLY

Network protocols

Network protocols are rules and conventions for communication between network devices. They ensure that data is transmitted accurately and securely across the network.

Common mobile network protocols

(a) GSM (Global System for Mobile Communications)

(i) *MAP (Mobile Application Part):* Used for network signaling.

(ii) *BSSAP (Base Station Subsystem Application Part):* Manages communication between BSC and MSC.

(iii) *SS7 (Signaling System No. 7):* Controls signaling across the network.

(b) CDMA (Code Division Multiple Access)

(i) *IS-95:* Standard for CDMA networks.

(ii) *CDMA2000:* Enhanced version providing high-speed data services.

(c) LTE (Long Term Evolution)

(i) *S1-AP (S1 Application Protocol):* Handles signaling between eNodeB (base station) and EPC (Evolved Packet Core).

(ii) *X2-AP (X2 Application Protocol):* Manages signaling between eNodeBs.

(iii) *GTP (GPRS Tunneling Protocol):* Used for carrying data between the mobile device and the Internet.

(d) 5G Protocols

(i) *NG-RAN (Next Generation Radio Access Network):* Protocols for the new radio access network.

(ii) *5GC (5G Core):* Core network protocols handling data and signaling.

Network security protocols

(i) *SSL/TLS (Secure Sockets Layer/Transport Layer Security):* Secures data transmission over the Internet.

(ii) *IPSec (Internet Protocol Security):* Secures IP communications by authenticating and encrypting each IP packet.

(iii) *HTTPS (Hypertext Transfer Protocol Secure):* Secures web communication.

Exercise 3.11

1. Explain the fundamental differences between PSTN and mobile telephone systems in terms of infrastructure and technology.
2. Describe the process of a call setup and teardown in a PSTN. What are the roles of the central office and the local loop in this process?
3. How does handover work in a mobile telephone system? Describe the different types of handovers and the scenarios in which they occur.

FOR ONLINE READING ONLY

4. What are the key differences between the various generations of mobile networks (1G, 2G, 3G, 4G, and 5G)? Focus on their technological advancements and capabilities.
5. Discuss the security challenges associated with PSTN and mobile telephone systems. What protocols or measures are commonly used to address these challenges?
6. Describe the role of the Mobile Switching Center (MSC) in a mobile telephone system. How does it interact with other components like the Base Station Controller (BSC) and the Base Transceiver Station (BTS)?

Chapter Summary

Reflect on what you have learnt from this chapter, summarise and keep it in your portfolio.

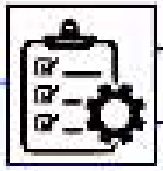
Revision Exercise 3

1. Imagine you are troubleshooting a network connectivity issue. How can your knowledge of the OSI and TCP/IP models assist you in identifying the problematic layer?
2. New technologies are continually emerging. How do you envision the OSI and TCP/IP models needing to evolve to incorporate future network functionalities?
3. Research a particular network technology, such as VPNs or wireless networking, and elaborate how it operates within the framework of the OSI or TCP/IP model
4. How does VLAN segmentation contribute to network efficiency and security, and what are the key considerations when implementing VLANs in a large-scale enterprise network?
5. What role does ARP (Address Resolution Protocol) play in modern networking, and how does it facilitate communication between devices within the same subnet?
6. Explain the significance of HTTP (Hypertext Transfer Protocol) in the context of web communication, and how its features contribute to the browsing experience for end-users.
7. What are the primary differences between FTP (File Transfer Protocol) and HTTP in terms of their functionalities, security features, and suitability for various file transfer scenarios?
8. Given a network with the IP address 192.168.1.0/24, how many subnets can be created using a subnet mask of 255.255.255.192, and what are the valid IP ranges for each subnet?
9. In a network with the IP address range 172.16.0.0/16, how many

FOR ONLINE READING ONLY

host addresses are available per subnet if subnetting with a mask of 255.255.255.240?

10. If a company requires four subnets, each accommodating at least 30 hosts, what is the minimum subnet mask that can be used, and what are the network addresses and valid host ranges for each subnet?
11. In a network with the IP address 10.10.0.0/22, how many bits are used for subnetting, and how many subnets and hosts are available in this configuration?
12. Given the requirement to create six subnets from the network 192.168.10.0/24, determine the appropriate subnet mask and calculate the valid IP ranges for each subnet.
13. Cybersecurity threats are a growing concern. How does understanding the OSI and TCP/IP models aid in identifying potential vulnerabilities and implementing security measures at different network layers?
14. Explain the concept of social engineering and how attackers exploit human psychology to gain access to sensitive information or systems. How can organisations raise awareness and mitigate social engineering risks?
15. How can vulnerability scanning and patching help identify and address potential security weaknesses?



Project: *Designing a school network*

Suppose you have been assigned to design a network for your school, which has been allocated the IP address 198.50.60.0. Describe the procedures you could undertake to complete this project.

In your project, make sure you deliver a comprehensive and detailed report covering the analysis, design decisions, and implementation process including screenshots of the Cisco Packet Tracer implementation.

FOR ONLINE READING ONLY



Chapter Four

Object Oriented Programming with C++

Introduction

In everyday real-life activities, we use Object-Oriented Programming (OOP) directly or indirectly. This is a paradigm that helps in organizing and structuring code in a way that is both intuitive and scalable. By modeling real-world entities as objects with properties and behaviors, OOP allows for a more natural mapping of complex systems into manageable software solutions. In this chapter, you will learn about Concept of Object-Oriented Programming (OOP), General structure of object oriented programs, Syntax and constructs to create Object-Oriented programs, and how to debug object oriented programs using C++. The competencies developed will enable you tackle problems in your surroundings by transforming challenges into OOP solutions and fostering collaboration within coding teams. Furthermore, you will be able to code maintenance and debug software systems.

Think:

Sorting tools is a toolbox or following guiding instructions in cooking4

Concept of Object-Oriented Programming (OOP)

Scenario 4.1: *Your school library system*

Your school library wants to efficiently organise and manage its collection of books to improve book service delivery at your school. The librarian aims to create a system that can keep track of books, handle checkouts, and provide information to students and staff.

Questions:

1. What kind of information should the system collect and from which type of system users?
2. Do you think, in this scenario, the OOP can be applied in real-world situations? Briefly explain how.

FOR ONLINE READING ONLY

Meaning of Object Oriented Programming

Object-oriented programming (OOP) is a programming concept that revolves around the idea of "objects." These objects hold data as fields (also called attributes or properties) and code as procedures (also called methods). OOP enables developers to build programs in a modular way and create code that can be reused.

In OOP, objects are created from classes, which act as templates for creating objects. Classes specify the characteristics and actions that objects of that class will possess.

Figure 4.1 represent a class named "Car" and one object instantiated from this class: Toyota. The object Toyota has specific attributes and methods. Attributes are characteristics or properties of an object, while methods are actions or behaviors that the object can perform. For instance, Toyota has attributes such as Make (Toyota), Model (Carina), Color (Silver), Year (2021), and Mileage (31500 km), and it can perform methods like StartEngine(), Accelerate(), and Brake(). This summary provides a clear overview of the attributes and methods associated with each object within the "Car" class. The structure of Object Oriented Programming can be explained using a car example, as in Figure 4.1.



Figure 4.1: *A car as an object*

Another example can be represented by a class "Plant" with objects like Apple and Orange. Apple has attributes like Name(Apple), Color (Red), Taste (Sweet), and Weight (150g). It also has a method like Ripen(), rote(), isSweet().

The structures can be further explained as follows:

(i) *Classes:* Classes are user-defined data types that act as blueprints for creating individual objects. They define the attributes and methods that objects of that class will have.

(ii) *Objects:* Objects are instances of a class. They represent specific entities with their own unique data and behaviours. Objects can correspond to real-world objects or abstract entities.

(iii) *Methods:* Methods are functions defined inside a class that describe the behaviors of an object. They are used to perform specific actions or operations on objects. Methods can be reused and encapsulated within objects.

FOR ONLINE READING ONLY

(iv) *Attributes:* Attributes, also known as fields or properties, are defined in the class template and represent an object's state. They store data associated with objects.

The organisation of an object-oriented program through these structures promotes code reusability, scalability, and efficiency.

Code reusability: Is the practice of utilising existing code for new purposes without rewriting it. For example, in Object-Oriented Programming, specific vehicle types like Cars, Trucks, and Motorcycle are created by inheriting common functionalities from a superclass like Vehicle.

Scalability: Is the system's ability to handle increased workload or expand without structural changes. For instance, in a banking application, adding new account types like checking accounts without altering existing code demonstrates scalability.

Efficiency: Refers to the optimal utilization of system resources. For example, efficiently storing student records in memory using data structures like arrays and implementing fast sorting algorithms for large datasets showcases efficiency.

OOP also facilitates collaborative development by dividing projects into groups. OOP provides benefits such as abstraction, inheritance, polymorphism, and encapsulation.

Exercise 4.1

1. What criteria must a programming language meet in order to be classified as an Object-Oriented Programming language?
2. How are classes different from objects? Illustrate with your own suitable examples.
3. How would you explain the concept of a class to someone who has never encountered object-oriented programming before? Can you think of an everyday example that resembles a class in programming?
4. Why is it important to understand the relationship between classes and objects in Object-Oriented Programming? How does this relationship contribute to the organisation and efficiency of code?

Key principles of Object-Oriented Programming

Encapsulation

To understand well the meaning of encapsulation, visualise how you are going to build a house as shown in Figure 4.2 for your household dog. In OOP this house represents a class. This house may have a bedroom for a dog to sleep in as well as a playroom. These rooms are examples of an attribute of the dog's house. In this house a dog can have a door for entering and for leaving outside the pet dog house. This dog door represents

FOR ONLINE READING ONLY

a method in a class, it is like a method in an OOP where objects use methods to access and modify data. In this example, to protect a dog is like to protect data. I hope you will not like people to reach your dog easily in this house and take or kill it. The idea of encapsulation achieves this by making one attribute, like a dog's bedroom, private. The other method in this class is a feeding plan method. This is the authorised method that can interact with the dog through other method (dog's door). In this example, a dog is like data.



Figure 4.2: *Encapsulation*

In this case, encapsulation in OOP refers to the practice of bundling data and the methods that operate on that data within a single unit, typically a class. It involves hiding the internal details of an object and providing a public interface to interact with it. Encapsulating data and methods together ensures that the object's state is accessed and modified only through defined methods, known as getters and setters.

Importance of encapsulation in OOP

(i) *Data protection:* Encapsulation ensures that only authorised methods have access to or can modify sensitive data. In our example of the pet dog house, protection prevents someone from directly accessing the dog's bedroom (data) from outside the house (class). A house as an object is also an instance of a class.

(ii) *Controlled modifications:* The dog's door (methods) regulates how the dog interacts with the surrounding environment. Similarly, methods can enforce rules regarding data modifications. For example, a method for supplying food may only allow for a measured quantity to be given to the dog.

(iii) *Information hiding:* You do not need to be aware of the inner workings of the dog door mechanism (internal details) in order to use it. Encapsulation hides the internal implementation details of a class. Instead, you interact with the methods (dog's door) to achieve what you intended (feeding the dog).

(iv) *Flexibility:* Assume you wish to add an area for scratching to the playroom (an internal change). Encapsulation allows you to change the house's interior (class structure) without impacting how people interact with the dog house (object). They still utilize the dog door (methods).

Exercise 4.2

1. How does the concept of encapsulation help in maintaining data integrity and promoting information hiding?

FOR ONLINE READING ONLY

2. Explain how encapsulation is achieved in the Book class of the library management system scenario.
3. How do getters and setters contribute to encapsulation in OOP, and what is their role in accessing and modifying object state?
4. In the Car class example, how are encapsulation principles applied to control access to the car's attributes and methods?
5. Can you describe a real-world scenario where encapsulation would be beneficial, based on the examples provided in the code?

Inheritance

Inheritance, is a core principle in OOP that enables a new class to acquire attributes and methods from an existing class, referred to as the superclass or parent class. This facilitates the reuse of code and facilitates the establishment of hierarchical connections between classes.

(a) **Parent (Superclass):** Think of an existing class as a parent class (superclass). It has its own properties and behaviors (such as nose, color or eating ability). The superclass generously allows other classes to inherit these traits.

(b) **The Inheriting child (Subclass):** A new class (subclass) can act like a child, inheriting the traits (such as nose, color, eating ability) from the superclass. This subclass can then use these inherited traits and potentially add its own unique characteristics (wearing rubber, jogging) to become something more specific.

The following is an example of inheritance in real world:

Imagine a class called "Animal." This class would involve shared attributes of animals such as name, type (such as, mammal, bats), and a makeSound() method.

Now, a class called "Cat" can inherits from "Animal." The "Cat" class automatically acquires all the properties (attributes) and methods from the "Animal" class. This eliminates the need to rewrite code for common functionalities.

Additionally, the "Cat" class can introduce its own distinct attributes and methods. For instance, it might include a breed attribute and a vocalise () method specifically tailored for cats.

(c) **Categories of Inheritance:** There are four categories of inheritance. Figure 4.3 shows these categories of inheritance.

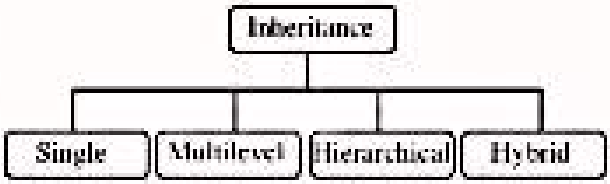


Figure 4.3: *Categories of inheritance in OOP*

(i) Single inheritance: This type of inheritance enables a subclass to inherit the properties and behaviors of a single

FOR ONLINE READING ONLY

parent class. In Figure 4.4, Class A is shown as the parent class, while Class B is portrayed as the child class inheriting properties and behavior from its parent class.

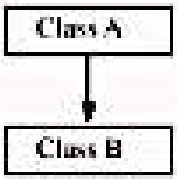


Figure 4.4: *Single inheritance*

(ii) Multilevel inheritance

Multilevel inheritance happens when a class inherits from another class, which in turn inherits from another class, forming a chain of inheritance. In Figure 4.5, Class C inherits from Class B, and Class B inherits from Class A. This means that Class C inherits both the traits of Class B and the properties and behavior of Class A. Thus, Class C automatically acquires the properties and methods of both Class A and Class B.

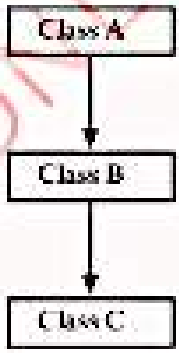


Figure 4.5: *Multilevel inheritance*

(iii) Hierarchical inheritance

A class is considered to have hierarchical inheritance when it serves as the parent class for multiple child classes or subclasses. Figure 4.6, shows the child classes B and C inheriting from the parent class A.



Figure 4.6: *Hierarchical inheritance*

(iv) Hybrid inheritance

Hybrid inheritance is formed by combining multiple inheritance and multilevel inheritance, typically using interfaces due to Java's lack of support for multiple inheritance. In Figure 4.7, Class A acts as the parent class for both classes B and C. Classes B and C, in turn, serve as parent classes for class D, which is the sole child class of both B and C.

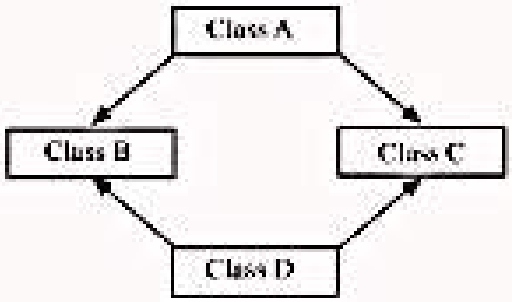


Figure 4.7: *Hierarchical inheritance*

Exercise 4.3

1. Analyse the statement that, "Inheritance is the process by which objects of one class acquire the properties of objects of another class."
2. How does multilevel inheritance differ from single inheritance? Explain the potential benefits and

FOR ONLINE READING ONLY

drawbacks of using multilevel inheritance in an OOP application.

3. Provide a simple, real-world example of multilevel inheritance involving a base class, a derived class, and a further derived class, and explain how methods and attributes are inherited and extended at each level?

Polymorphism

Polymorphism is an important concept in Object-Oriented Programming that allows objects from various classes to be viewed as members of the same superclass. It allows a single interface to represent various sorts of objects, increasing flexibility and code reusability. Polymorphism enables a same method name to perform different actions depending on the type of object it is called on.

Example of polymorphism in real world is when you consider parents and children. The human body can be viewed as an object that has properties and methods. Examples include private properties like the brain, heart, liver, stomach and lungs, as well as public properties like the eyes, ears, fingers, head, hair and legs. Another simple example is when you reflect someone who understands how to drive can get into and drive most cars since they have key traits such as a steering wheel, a gear lever, and clutch, brake, and accelerator pedals that the driver is familiar with. There may be many differences between any two cars, but consider them subclasses of a superclass that defines these operations. Also, if you imagine a super class named "Animal." As explained in the previous paragraph in inheritance, this class might have a method called "makeSound()". Coming to subclasses like "Dog" and "Cat" that inherit properties from "Animal.", both "Dog" and "Cat" inherit the "makeSound()" method. This reflects Polymorphism especially when "makeSound()" behave differently for Dog class and for Cat class. The "makeSound()" on a "Dog" object when it is called, it might bark, while calling "makeSound()" method on a "Cat" object might produce a meow.

(a) Roles of polymorphism

(i) *Code reusability:* Polymorphism makes code reuse possible by giving classes the ability to implement a common interface. In other words, a single piece of code can interact with multiple objects, which eliminates duplication and encourages effective code reuse.

(ii) *Flexibility and adaptability*: Code can be written in a more flexible and adaptive manner thanks to polymorphism. Code can handle several object types without knowing their unique class by programming to an interface instead of specific implementations. As a result, the code is more adaptable and able to handle a larger variety of situations.

(iii) *Simplifies code design:* Polymorphism makes code design

FOR ONLINE LEARNING ONLY

simpler by letting programmers concentrate on the shared behavior and interface between several objects. This concept encourages improved code organization, increases modularity, and facilitates comprehension.

(b) Relationship between inheritance and polymorphism

(i) Inheritance facilitates polymorphism by providing a mechanism for subclasses to override methods inherited from their superclass.

(ii) Polymorphism allows for the appropriate method implementation to be selected based on the actual type of the object at runtime.

(iii) Through inheritance, subclasses can focus behavior inherited from their superclass, allowing for polymorphic behaviour when interacting with objects of the superclass type.

(iv) Both inheritance and polymorphism promote code reuse, flexibility, and extensibility in object-oriented systems designs when they.

(c) Method overriding behaviour in polymorphism

Method overriding is a type of polymorphism where a subclass provides its unique implementation of a method that already exists in its superclass. This implies that a method already defined in a superclass can have a subclass define it individually. By overriding a method, the subclass can customise its behavior according to its specific requirements, while still being able to utilise the functionality inherited from the superclass. For example, consider a superclass called Animal with a method called makeSound(). The Animal class may have a default implementation of the makeSound() method, such as printing "Animal makes a sound." The subclass Dog, which extends the Animal class, can override the makeSound() method to provide its own implementation, such as printing "Dog barks." This allows for different behaviours in different subclasses. In the context of method overriding, polymorphism treat objects of different subclasses as objects of their common superclass. This promotes high flexibility and reusability of the code.

By declaring a method with the same name and parameters as the method in the superclass, a subclass can provide its own implementation of the method. Using a specific OOP language, the subclass can declare that it is purposefully overriding the method by using the "override" keyword. This ensures that the compiler will produce an error if the method is not truly overridden. During method overriding, certain behaviors occur and known as dynamic polymorphism.

Dynamic polymorphism, also known as *runtime polymorphism*, is a type of method overriding in which the choice

FOR ONLINE READING ONLY

of which method to invoke is made based on the actual type of the object. Runtime is the period of time that a computer's processor executes a compiled program. During this time, the CPU executes out the program's instructions, communicates with memory and other system resources, and generates the desired output.

(d) Method overloading behaviour in polymophism

Method overloading is a concept in object-oriented programming that permits the coexistence of multiple methods in a class with the same name but different parameters. As a result, a class may have more than one method with the same name, but each method must have a unique set of parameters. One type of polymorphism is method overloading, which allows for the definition of many methods in a class with the same name but distinct parameters. Depending on the arguments given, the compiler or interpreter decides which version of the method to execute when a call is made. As an example, imagine we have a class named Calculator.

There may be more than one "add" method in this class, and the parameter types used by each method may vary. We can, for instance, take two different "add" methods: one that accepts three integer parameters and returns their sum, and another that accepts three double parameters and returns their sum. Various behaviors are allowed depending on the arguments that are supplied to the method. A class can have numerous methods with the same name but different parameters or distinct types of parameters, this is what method overloading can do in OOP. Because several versions of the method can be called based on the unique data types or values provided as arguments, this enables more flexible and intuitive code. In this case, Python does not support method overloading by default.

The decision on which method to execute is made at compile time, based on the method signature.

Compile-time polymorphism, sometimes referred to as static polymorphism, is a type of method overloading. It offers convenience and flexibility in method invocation by enabling a class to have many methods with the same name but distinct capabilities. Compile time is the period of time that a compiler uses to translate a program's source code into machine code or an intermediate form. The activities done during compile time, are the compiler to do syntax checking, type checking, and code optimisation.

Exercise 4.4

1. Give an example of how polymorphism can improve code reusability and maintainability
2. Imagine a class named "Animal" with a method "makeSound()". In Object-Oriented Programming

FOR ONLINE READING ONLY

(OOP), how might polymorphism be used with this scenario?

3. Imagine you are designing a simple video game. How would you use classes and objects to represent different elements of the game, such as characters, items, and environments? How might inheritance and polymorphism come into play in this scenario?
4. Consider a superclass "Shape" with a method "getArea()". Subclasses like "Square" and "Circle" inherit this method. Which type of polymorphism is at play when the appropriate "getArea()" method is chosen based on the object's class at runtime?
5. You have a function "greetAnimal(animal)" in your code. This function can accept objects of various animal classes (Dog, Cat, etc.). How might polymorphism be beneficial in this scenario?
6. Imagine a class hierarchy for vehicles (Car, Truck, Bike). How could polymorphism be used to create a function "moveAllVehicles(vehicles)" that works with all vehicle types?
7. Polymorphism is often associated with inheritance. Can polymorphism exist without inheritance? Explain your answer.
8. In the real world, provide an example of something that exhibits a similar concept to polymorphism.

Abstraction

Abstraction in Object-Oriented Programming (OOP) helps you manage complexity by focusing on what an object does, rather than how it does it. Think of it like driving a car: you only need to know how to drive, not how the engine works. Abstraction lets you use objects as "black boxes" that do specific tasks, without worrying about their inner workings.

(a) Abstraction in the real world

Making coffee with a coffee machine is a good example of abstraction.

You need to know how to use your coffee machine to make coffee. You need to provide water and coffee beans, switch it on and select the kind of coffee you want to get.

The thing you do not need to know is how the coffee machine is working internally to brew a fresh cup of delicious coffee. You do not need to know the ideal temperature of the water or the amount of ground coffee you need to use.

The manufacturer of a machine resolved issues of temperature and quantity, so, created a coffee machine that now acts as an abstraction and hides all these details. You just interact with a simple interface that does not require any knowledge about the internal implementation. You can use the same concept in any Object-Oriented Programming languages.

FOR ONLINE READING ONLY

(b) Abstraction in OOP: Hiding complexity

Abstraction is a fundamental principle in OOP that allows programmers to focus on the essential details of objects and their interactions, hiding away the underlying complexities. It is like creating a simplified user interface for a complex machine.

Objects in an OOP language provide an abstraction that hides the internal implementation details. Similar to the coffee machine in your kitchen, you just need to know which methods of the object are available to call and which input parameters are needed to trigger a specific operation. But you do not need to understand how this method is implemented and what kinds of actions it has to perform to create the expected result.

(c) Levels of abstraction In OOP

Within the concept of abstraction in OOP, there are two main levels that address different aspects of hiding complexity:

(i) Data abstraction: This level, as shown in Figure 4.8 focuses on hiding the details of how data is stored and manipulated within an object. Its like providing a user interface for interacting with data without needing to know the specific data structures or memory locations used behind the scenes.

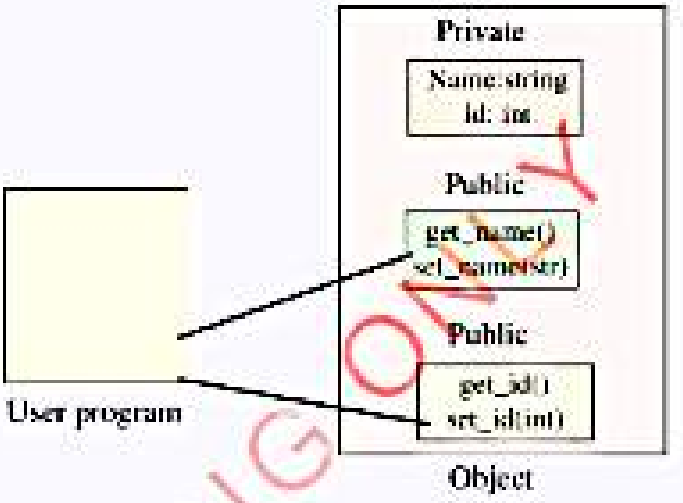


Figure 4.8: *Data abstraction*

Example 4.1:

Imagine a Bank class with an attribute balance. You can access and modify the balance using methods like deposit() and withdraw(), but you do not need to know how the balance is actually stored internally (such as, integer, and floating-point number).

(ii) Process abstraction: This level focuses on hiding the implementation details of how an object performs an action. Its like providing a single entry point (method) for a specific functionality without revealing the complex steps involved in its execution.

Example 4.2 :

Assume you have a car class that has a method accelerate(). Calling this method initiates the acceleration process, but the student does not need to see the intricate code involving engine control, fuel injection, and transmission changes.

FOR ONLINE READING ONLY

The major difference between the data abstraction and process abstraction is that data abstraction hides details about data storage and manipulation, while process abstraction hides details about how actions are implemented. For example, data abstraction is like hiding the wires and internal components of a piano while providing knobs and buttons for users. Process abstraction as shown in Figure 4.9 is like pressing a "play" button on the piano without needing to know the mechanics of how the music is played.

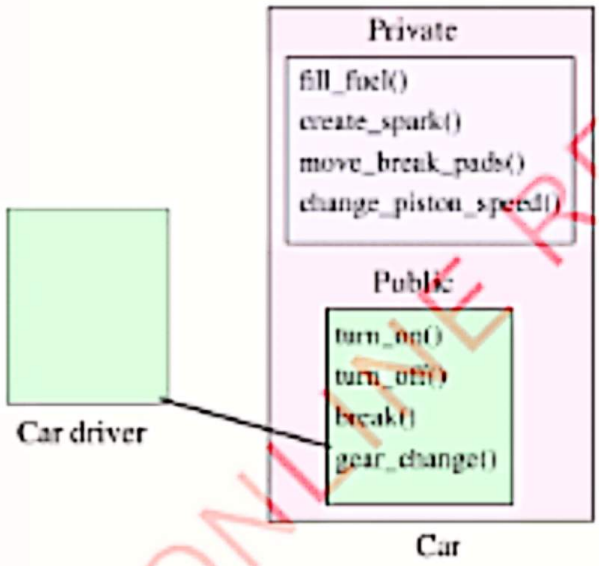


Figure 4.9: *Process abstraction*

The relationship between data abstraction and process abstraction is that both work together to achieve the overall goal of abstraction in OOP. Data abstraction ensures that data is accessed and modified in a controlled manner, while process abstraction ensures that functionalities are invoked and executed appropriately.

Exercise 4.5

1. Imagine a class called "BankAccount". Suggest the attributes that are expected for this class.
2. Describe how a principle of OOP would assist you in restricting direct access to the balance in your Bank Account so that someone may not change your account balance.
3. Recommend the methods to be used in the class "BankAccount" in order to control how the balance is accessed and modified, potentially adding security checks or validations.

   Instead, the class would provide methods like deposit() and withdraw(). These methods would control how the balance is accessed and modified, potentially adding security checks or validations.

FOR ONLINE READING ONLY

Difference between OOP and Procedure Oriented Programming (POP)

There are several differences between these two types of programming. Table 4.1 shows the differences between them.

Table 4.1: *Differences between POP and OOP*

Procedure Oriented Programming (POP)	**Object Oriented Programming (OOP)**
The program is partitioned into smaller components referred to as *functions*.	The program is divided into segments known as *objects*.
The emphasis is not on *data*, but on functions and the *sequence* of actions to be performed.	The focus is on *data* in contrast to procedures or functions, as it emulates *real-world* behaviour
POP follows *top-down approach*	Follows *bottom-up approach*.
It lacks any access specifier.	This includes access specifiers such as public, private, and protected.
Data can flow unrestrictedly between functions within the system.	Objects have the ability to relocate and interact with one another by means of member functions.
In POP, incorporating new data and functions is not a straightforward task	Offers a convenient approach for adding new data and methods
In most cases, functions rely on global data for sharing, which can be accessed without restrictions across different functions within the system	The movement of data between classes is not as straightforward. Instead, data can be designated as public or private, enabling us to control its accessibility and restrict its flow between classes
It lacks a proper mechanism for hiding data , making it less secure.	OOP provides data hiding, and it is more secure.
It is impossible to perform overloading.	Overloading is possible and achievable through both function overloading and operator overloading.
Examples include C, VB (that is, VB6), FORTRAN, Pascal	Examples include: C++, JAVA, VB.NET, C#.NET, Python, JavaScript, Ruby.

FOR ONLINE READING ONLY

Overview of C++

C++ as a high-level programming language was created as a C programming language extension. It is compatible with object-oriented, procedural, and generic programming paradigms. Low-level capabilities like pointers and manual memory management are provided by C++, allowing programmers to have exact control over system resources.

Features

(a) *Object-Oriented Programming (OOP):* C++ Support classes, objects, inheritance, polymorphism, and encapsulation.

(b) *Standard Template Library (STL):* By offering generic data structures and algorithms, the STL promotes code reuse and increases productivity.

(c) *Performance:* System programming, game development, and applications requiring high performance can benefit from well-known efficiency and performance of C++.

(d) *Platform independence:* Although C++ code can be compiled to operate on a variety of platforms, platform-specific compilation and modification are necessary.

Applications

(a) *Game development:* C++ is widely used for creating game engines, graphics programming, and game logic.

(b) *Performance-critical applications:* C++ is preferred in domains where performance is critical, such as financial applications, simulations, and scientific computing.

(c) *System programming:* C++ is used for developing operating systems, device drivers, and embedded systems.

Introduction to C++ programming language

C++ programming language is a high-level language extended from the C programming language. It contains both imperative (procedural) and object-oriented programming language features. Examples of imperative features are records while that of object-oriented programming features are classes. C++ programming is a compile-based language that uses different compilers to develop and maintain programs. It is a case sensitive language like Python, Ruby, Java, Swift, and Verilog. This makes C++ differ from other programming languages such as Visual Basics, and SQL. The C++ programming language is also a platform dependent, which means that the source code written should be compiled in every platform. This C++ characteristic does not apply to other languages like Java.

Structure of C++ program

Consider the following simple C++ program to display text "Introduction to C++ programming" as shown in Program Example 4.1.

FOR CONFIDENTIAL READING ONLY

Program Example 4.1:

Simple program to display text

Simple C++ program to display text "Introduction to C++ programming"

```
#include<iostream>
using namespace std;
int main()
{
        // Display text Introduction to C++ program
        cout<<"Introduction to C++ program"<<endl;
        system("PAUSE");
        return 0;
}
```

Description of line of codes in a program

Table 4.2 shows the application of each line from the given C++ programs as presented in Program Example 4.1.

Table 4.2: *Defining the structure of the C++ program*

Line	Description
#include<iostream>	It is a preprocessor directive that tells the compiler to include iostream, to support input and output operation. **N.B:** - Sometime iostream is known as *header file* because it is included at the head of a program.
using namespace std;	This alerts the compiler to find the names in the standard library, for a example, "cout" and "cin" are defined in the standard library.
int main()	The main function "main()" defines the body of the program to which a program execution takes place. The statements within the function should be enclosed by curl bruckets"{}." Every statement in the main function of the C++ program should end with a semicolon ";" as the statement terminator.

FOR ONLINE READING ONLY

//	This is a double forward slash. Any text written after these slashes is ignored by a compiler because it stands as a comment. N.B:- Comments are not the line of codes; hence, they are not compiled. While double slashes "//" stands, as a line comment, the enclosed lines with the star "/*……*/" stand as a paragraph comment.
cout<<"Introduction to C++ program"<<endl;	"cout" represents console output that sends a message to the compiler to display the text "Introduction to C++ program." "<<" stand as insertion operator which send string of texts to the console. A text should be enclosed by quotation marks."endl" represents the end line sending a message to the console to output a new line and ensure the text is displayed.
return 0	Located at the end of the main function responsible for exiting the program

Environment setup for C++ program

Before you create a C++ program, you must install C++ Compiler. The compiler is a software that enables a program to be *created* to communicate with the computer systems. There are many compilers that can be used in C++ programming, which include Dev C++, Boland C++, and Visual C++. Other compilers are available online. All examples in this book were prepared using Dev C++.

Installation of Dev C++ compiler

The following are steps to install Dev C++ compiler (Based on Version 5.11):

Step 1: Using any search engine, download a complete version of Dev C++ depending on your operating system by clicking the button "Download" as shown in Figure 4.10.



Figure 4.10: *Interface for downloading of Dev C++*

FOR ONLINE READING ONLY

Step 2: Install Dev C++ on your computer by double-clicking the downloaded executable file.

Step 3: Choose a language, then click "OK" to install Dev C++ as shown in Figure 4.11.



Figure 4.11: *Choose language to install*

Step 4: Read and click "I Agree" button to accept the terms of the agreement as shown in Figure 4.12.

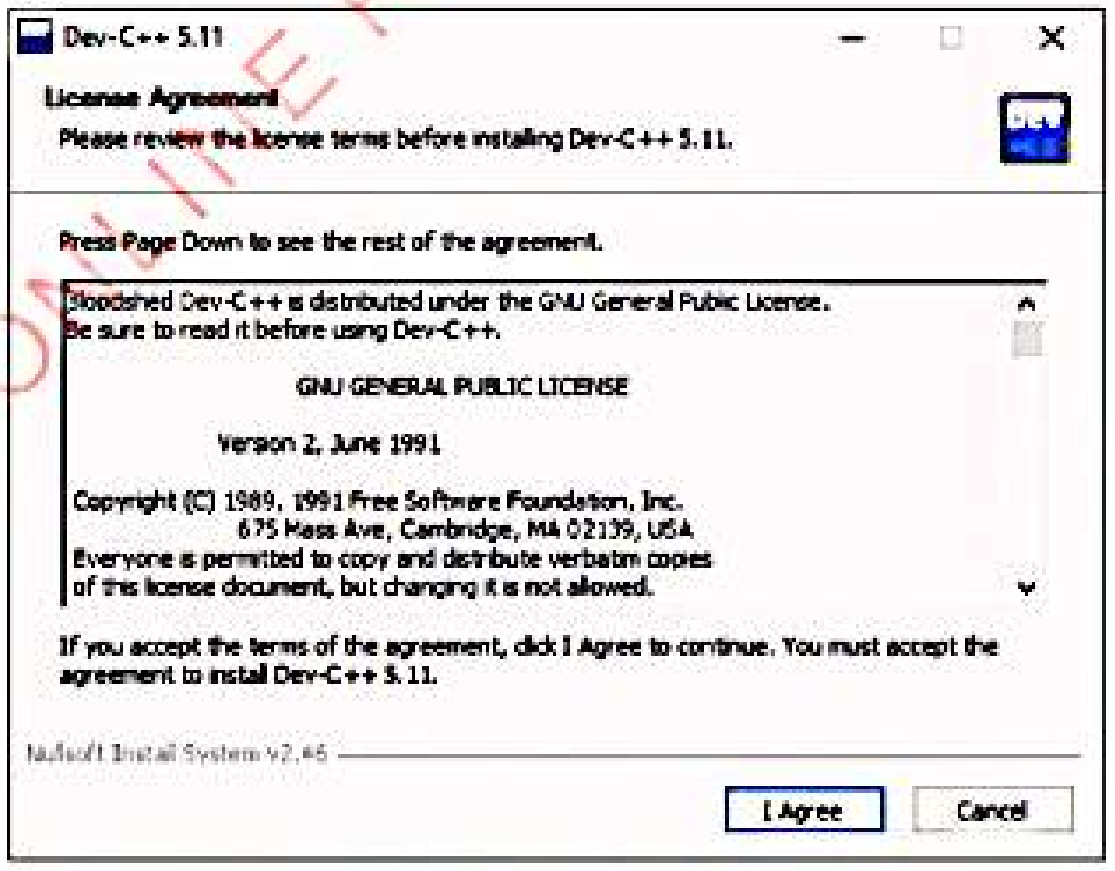


Figure 4.12: *Terms of the agreement dialog box*

FOR ONLINE READING ONLY

Step 5: Choose features you want to install within Dev C++ as shown in Figure 4.13.

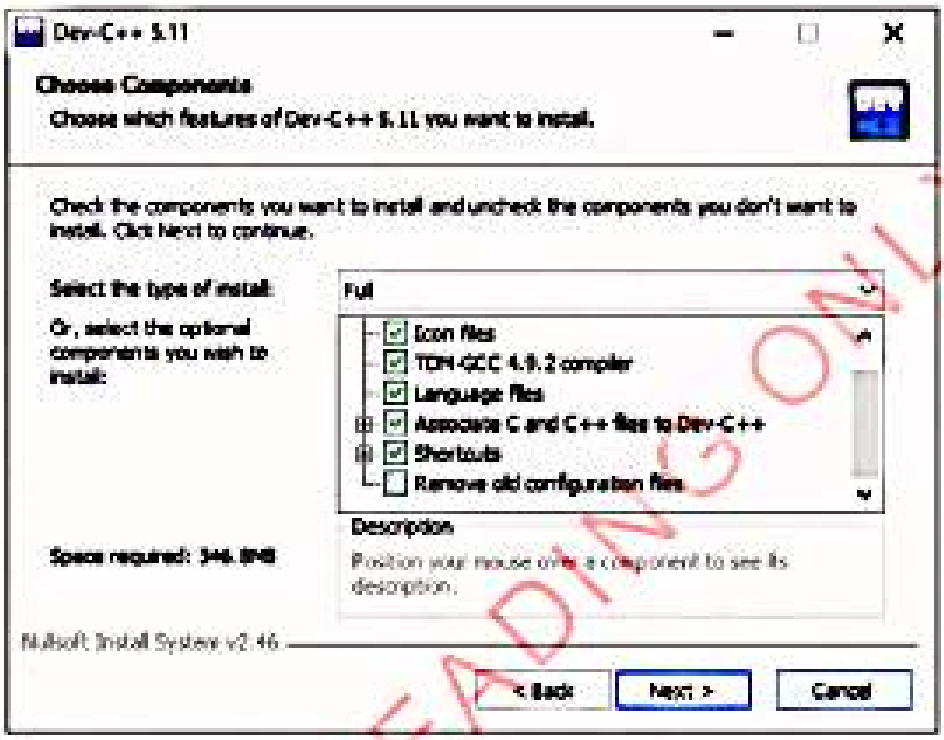


Figure 4.13: *Components of Dev C++*

Step 6: Locate the folder to install Dev C++ by clicking the button "Browse" then click "Install" button as shown in Figure 4.14.

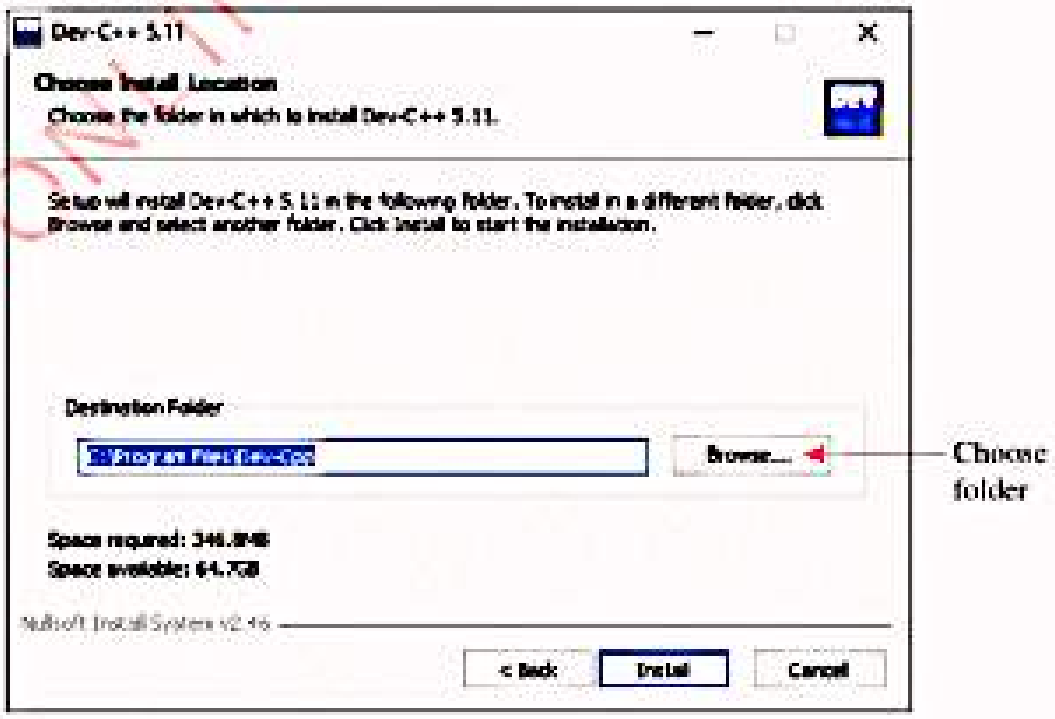


Figure 4.14: *Selecting the folder to install*

FOR ONLINE READING ONLY

Step 7: Wait for the installation of Dev C++ as shown in Figure 4.15

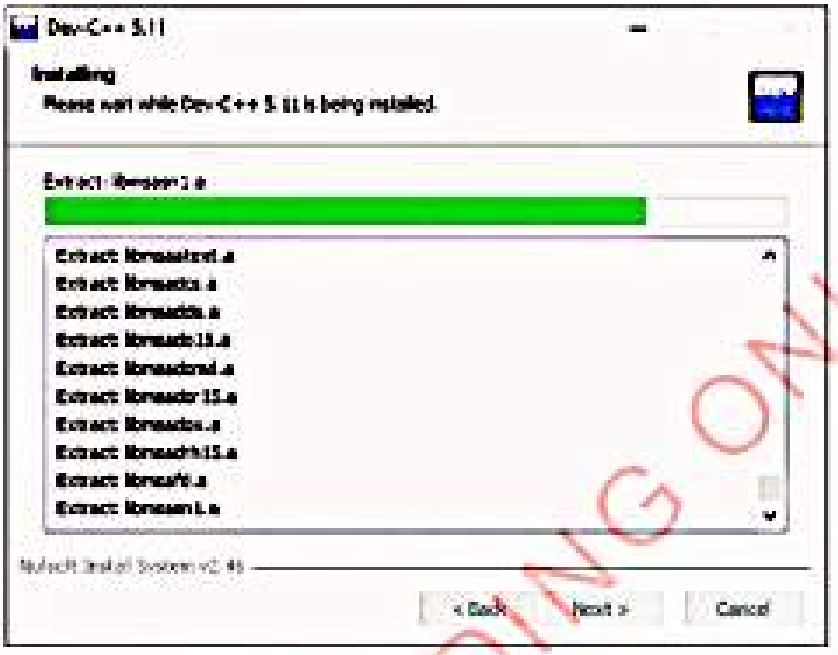


Figure 4.15: *Installation process*

Step 8: Finish your installation by clicking the "Finish" button, as shown in Figure 4.16.



Figure 4.16: *Finish installation*

Developing C++ programs using Dev C++

Opening Dev C++ compiler

The following are the steps to open the Dev C++ compiler.

FOR ONLINE READING ONLY

Step 1: Click the Dev C++ icon to open the following dialog box, as shown in Figure 4.17. Select language to 'English (Original)' and click 'Next'.

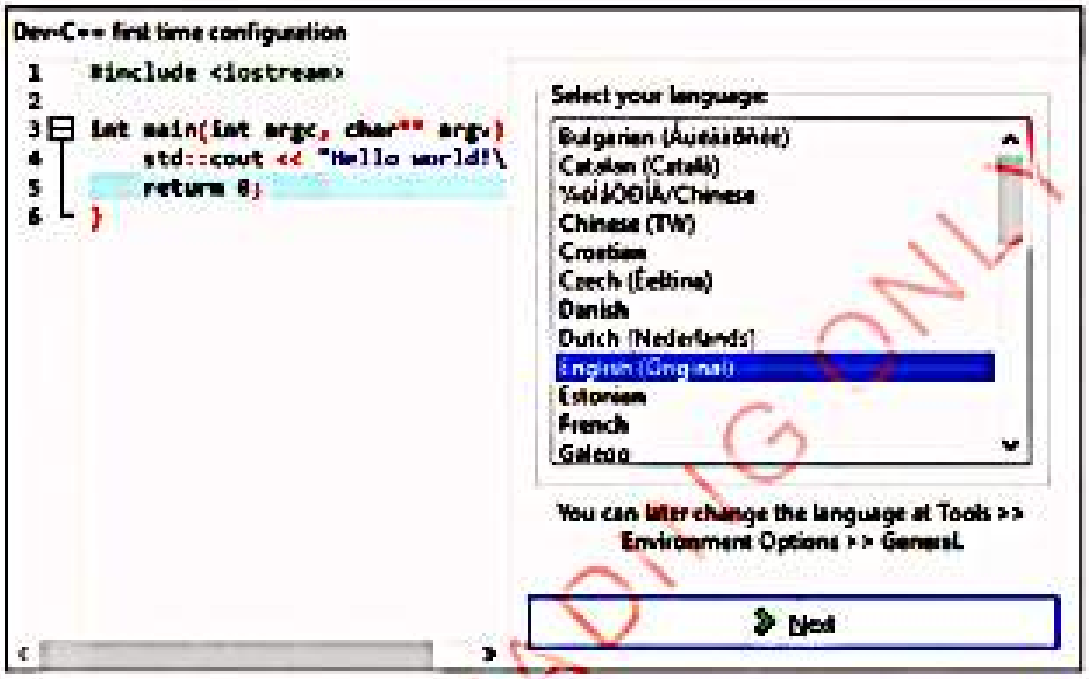


Figure 4.17: *Opening of Dev C++*

Step 2: Click the "Next" button followed by "Ok" to open the dialog box

Step 3: Click the "file" button followed by "Ok" to open the content pane as in Figure 4.18.

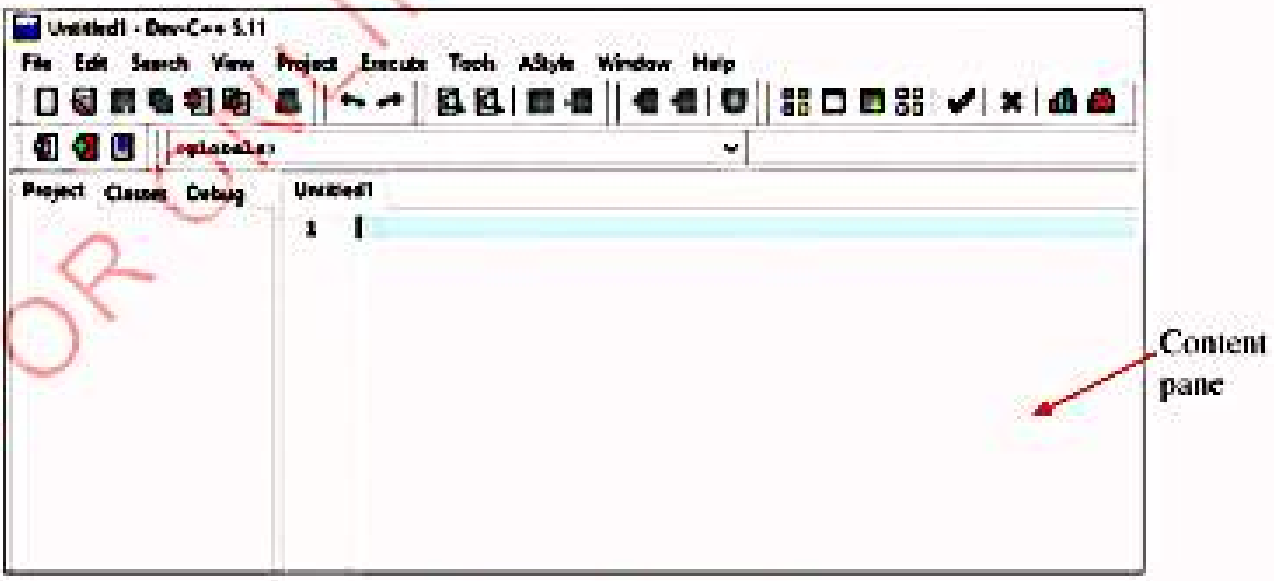


Figure 4.18: *Content pane*

Writing a C++ program in the content pane of the compiler, you can type C++ program on the content pane as shown in Figure 4.19.

FOR ONLINE READING ONLY

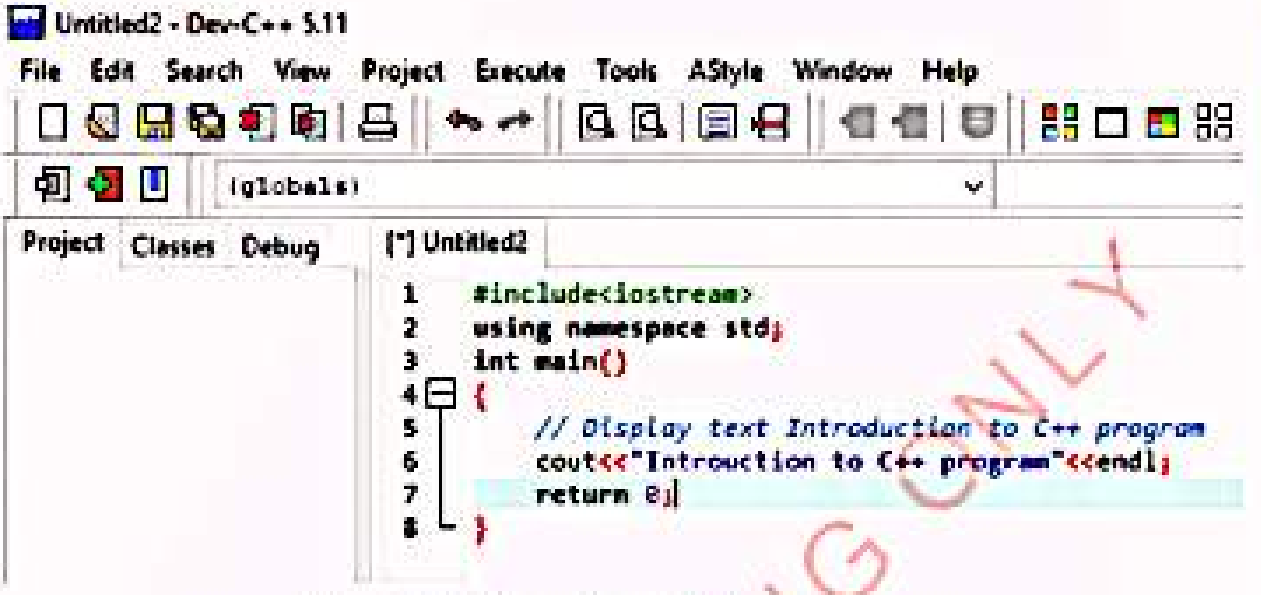


Figure 4.19: C++ codes within the content pane

Saving a C++ program

From the coding window of the C++ program, click *File* followed by *Save As* to save the C++ program. Then, the dialog box to choose a folder for saving a program will be displayed as shown in Figure 4.20. Type *Introduction.cpp* in the file name, then click *Save* to save your file.

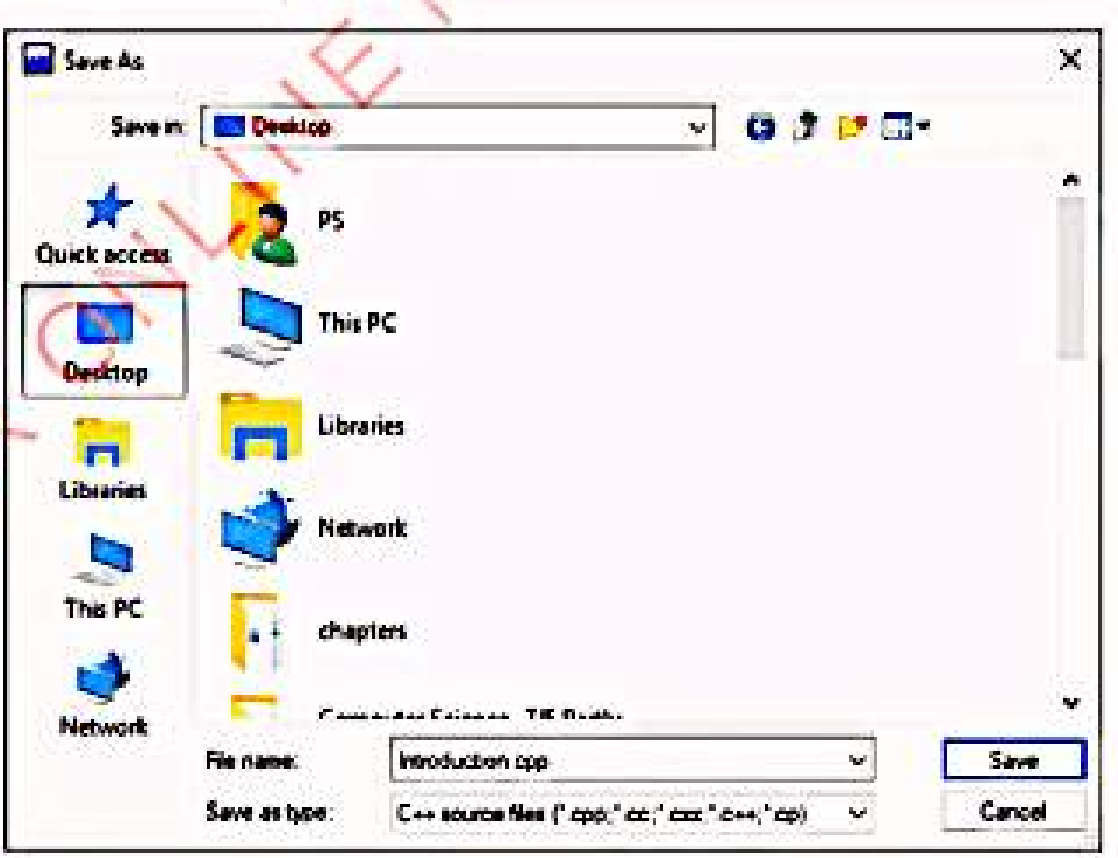


Figure 4.20: Saving a C++ program

FOR ONLINE READING ONLY

After saving a program as "Introduction.cpp" the content title will change to *Introduction.cpp* as shown in Figure 4.21.

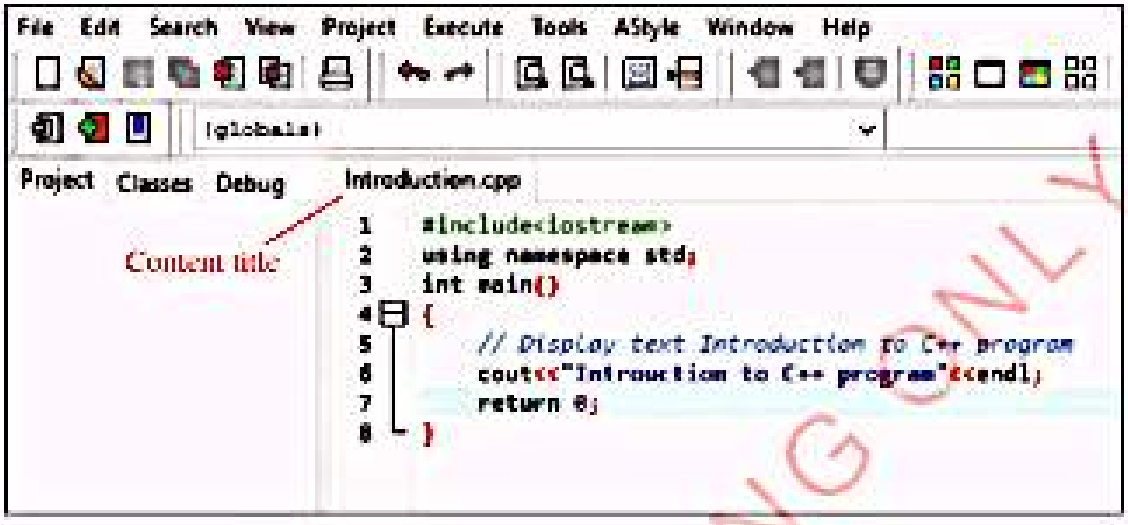


Figure 4.21: *Title change to "Introduction.cpp"*

Compiling a C++ program

It is important to compile your program not only to change source code to object code, but also to find errors in the program. Compile your program by clicking the "Execute" tab from the tool bar, then click "Compile." If there is any error the compiler will display the error for correction. Otherwise, the next step of running a program will follow.

Running a C++ program

Running a program involves copying a program to the RAM. The message is sent to the computer to prepare the environment for operating a program. Run your C++ program by clicking the "Execute" tab, then click "Run." Figure 4.23 shows the output after running the program.



Figure 4.22: *The output from a console window*

Note: When the output terminates after running the program, add 'system ("PAUSE")' to the codes to stop the execution window from terminating.

Program Example 4.2:

Adding two numbers

The following is a C++ program that adds two numbers.

```
#include<iostream>
using namespace std;
int main()
{
        cout<< 5+6<<endl;
        return 0;
}
```

The output of the program above is 11.

FOR ONLINE READING ONLY

Program Example 4.3:

Finding average

The following is a C++ program that finds the average of 50, 70, 100 and 80.

```
#include<iostream>
using namespace std;
int main()
{
        //Find average marks of 50, 70, 100 and 80
        cout<<"Average of the marks is "<<(50+70+100+80)/4<<endl;
        return 0;
}
```

The output will be average of the marks which is 75.

Installing a Borland C++ compiler

Activity 4.1:

Installation of Borland C++

Step 1: Download the complete version of Borland C++ Compiler.

Step 2: Copy the folder of Borland in the C drive.

Step 3: Write each step to install Borland C++ Compiler in your computer.

Errors in computer programming

When writing a computer program, you may make errors, such as incorrectly typing a keyword, or you may forget to declare a variable within a program, or initialise a variable with a wrong value. For example, you may write *cot << "Hello, World! \n";* instead of *cout << "Hello, World!\n";*. once you do that, the compiler will complain. Once an error has happened, you have failed to correctly adhere to the rules of writing a computer program according to the language rules, and the compiler finds it.

If the compiler finds one or more errors during the compilation phase, it will not translate the program to machine code, and as a result, such a program will not run. However, once you fix such an error and compile the program, it will run successfully.

There are four common errors that may occur when someone write a computer program. These are syntax, semantic, logical, and run-time errors.

FOR ONLINE READING ONLY

Syntax errors

A syntax error is a faulty construct that does not conform to the rules of the programming language. It is defined as a mistake in the spelling, punctuation or order of words in a program.

These errors are also called compile-time errors. They normally happen when the programmer fails to follow language syntax rules in the course of writing program. Examples of syntax errors are misspelling words using a variable that has not been declared and mismatched parentheses or braces. If you compile a program that has a syntax error, the compiler will not run. Instead, the compiler will tell you at what line a trouble is. Usually, the error can indicate at the exact line where such a syntax error belongs, or the line just before it.

In case the problem is incorrectly nested braces, the actual error may be at the beginning of the nested block. Once you fix the error, and compile it again, the program will run successfully.

The three most frequent syntax errors that novice programmers commit include: missing parenthesis ( ), printing the value of a variable without declaring it, and missing semicolon (;). The following C++ program in Program Example 4.4 illustrates syntax errors.

Program Example 4.4:

Syntax error

```
#include <iostream>
using namespace std;
int main()
{
   int a = 10;
   int b = 15;
   cout << " "<< (a, b) // semicolon
missed
return 0;
}
```

Error message:

error: expected ';' before '}'

Semantic errors

A semantic error is an error that relates to the meaning of the code, particularly a mistaken idea of how the compiler interprets the code. Semantic errors are typically syntactically valid, but they are inconsistent in their meaning or invalid such as attempting to divide a number by zero. Semantic errors happen when the statements written in the program cannot be well interpreted by the compiler. Sometimes these will cause your program to crash, such as in the case of division by zero: Semantic errors are problems with a program that runs without producing error messages, but it does not do the right thing. For example, an expression may not be evaluated in the order you expect, but yielding an incorrect result. The following Program

FOR ONLINE READING ONLY

Example 4.5 of the C++ program illustrates a semantic error.

Program Example 4.5:

Semantic error

```
#include <iostream>
using namespace std;
int main()
{
  int a, b, c;
  a + b = c; //semantic error
return 0;
}
```

Output:

Error: value required as left operand of assignment a + b = c; //semantic error

Logic errors

Logic errors are constructs that can be translated into a running program, but the resulting program does not perform the intended action. Logical errors may occur even if the program is syntactically correct and does something. However, the output of the program will be different from what was intended by the programmer. An example of a logical error is when a program uses a wrong formula, or assigns wrong value to the code assignment statements. Under this situation, the compiler cannot find the error, and it must be flushed out when the program runs by testing it and carefully looking at its output. To avoid logical errors, the programmer must plan the solution clearly before writing the program. Once the error occurs, the programmer must test the program and find any logic errors. The following Program Example 4.6 of the C++ program illustrates a logical error.

Program Example 4.6:

Logic error

```
#include <iostream>
using namespace std;
// A C++ program to illustrate // logical
error int main ()
{
   int i = 0;
   // logical error : a semicolon after loop
   for(i = 0; i < 3; i++);
   {
   cout << "loop ";
      continue;
   }
   return 0;
}
```

Output:

Error:

No output

Run-time error

Run-time errors are the types of errors that occur during program execution (run-time) after successful compilation of the program. Division by zero is one of the most common run-time errors. In this type of errors, the compiler does not directly point to the line where the error occurs. Run-time errors are also called "bugs". Before a program is released to

FOR ONLINE READING ONLY

the market, commercial software are normally checked to examine the process of bugs during the debugging process. Program Example 4.7 shows the C++ program to illustrate the run-time error

Program Example 4.7:

Run-time error

```
#include <iostream>

using namespace std;
int  main()
{
   int x = 10;
int  div = x/0;
   // wrong logic number is divided by 0,

   // so this program abnormally terminates     div = x/0;

  cout << "result = "<< div;

return 0;
}
```

Note:

Run-time errors are hard to trace because the compiler does not point to the line at which the error occurs.

Error

In the given example, there is Division by zero error. This is an example of run-time error i.e. errors occurring while running the program.

Exercise 4.6

1. Study the following C++ program:

```
include<iostream>;

using namespace std;
int main
{
cout<<Read and enjoy C++ programming<<endl;
return 0;
}
```

FOR ONLINE READING ONLY

(a) Identify all errors in the program.
(b) Correct errors before running the program.
(c) Run the program and display the results.

2. Describe five (5) characteristics of a good program.
3. Will the following lines of code in (a) and (b) print the same thing? Explain why or why not.
   (a) cout << 5 << '\n';
   (b) cout << "5" << '\n';
4. Will the following lines of code print the same thing? Explain why or why not.
   (a) cout << y<< '\n';
   (b) cout << "y" << '\n';

Data types

C++ programming language supports various types of data. Each data type defines its memory size, and improves the running speed of the program. C++ programming language provides different types of data based on numeric values, Boolean values, and characters under a specific range of values. C++ programming language is based on three categories of data types as shown in Figure 4.23. These are built-in or primary data types, user-defined data types, and derived data types.

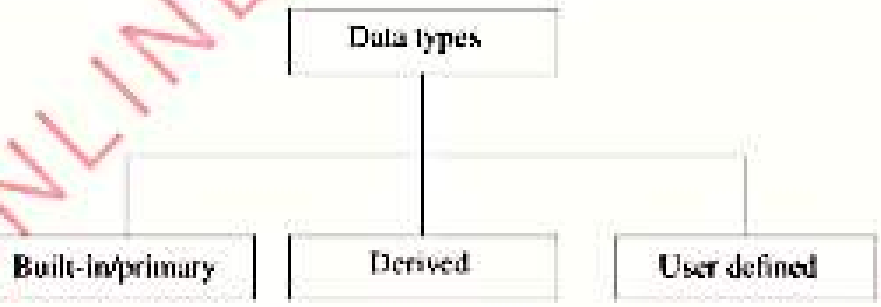


Figure 4.23: *Fundamental data types in C++*

Built-in or primary data types

Built-in or primary data types are pre-defined data types that can be applied directly without creating them. Built-in data types include *Integer, Character, Float, and Bool*.

(a) Integer data types

Integer data types involve whole number. Integer data types include, signed integer, unsigned integer, short integer, unsigned short integer, and long integer. Table 4.3 shows the classification of integer data types with their storage size and range.

FOR ONLINE READING ONLY

Table 4.3: *Integer Data types*

Type	Storage size	Range
Integer	4 byte	-2,147,483,648 to 2,147,483,647
Unsigned integer	4 byte	0 to 4,294,967,295
Short integer	2 bytes	-32,768 to 32,767
Unsigned short integer	2 bytes	0 to 65,535
Long integer	4 bytes	-2,147,483,648 to 2,147,483,647
Unsigned long integer	4 byte	0 to 4,294,967,295

(b) Character data types

Character data types comprise alphabetical data. These include char, signed char and unsigned char. The classification of character data types with their storage size and range of storing data is represented in Table 4.4.

Table 4.4: *Character data types*

Type	Storage size	Range
Char	1 byte	-128 to 127 or 0 to 255
Signed char	1 byte	-128 to 127
Unsigned char	1 byte	0 to 255

(c) Float data types

Float data types are based on decimal numbers. Float data types include float, double and long double. Table 4.5 represents float data types with their storage size and range.

Table 4.5: *Float data types*

Type	Storage size	Range
Float	4 bytes	±3.4E-38 to ±3.4E38
Double	8 bytes	±1.7E-308 to ±1.7E308
Long double	8 bytes	±1.7E-308 to ±1.7E308

(d) Bool data types

The bool data type has only two values, namely *true* and *false*. True and false are also called the logical (Boolean) values. The central purpose of this data type is to manipulate logical expressions. Use this data type for simple flags that track true/ false conditions.

FOR ONLINE READING ONLY

Derived data types

These are data types derived from primary data types. Derived data types are array, function, pointer, and reference.

User-defined data type

These are data types defined by the user. Such data types can determine the state of the data, whether public, private, or protected. User-defined data type includes structure, union, enumeration, and classes.

Activity 4.2:

Identifying data types from a program

```
#include<iostream>
#include<iomanip>
using namespace std;
void printMonth(int year,int month);
void printMonthTitle(int year, int month);
void printMonthName(int month);
void printMonthBody(int year, int month);
int getStartDay(int year, int month);
int getTotalNumberOfDays(int year, int month);
int getNumberOfDaysInMonth(int year, int month);
bool isLeapYear(int year);
int main()
{
        int year, month;

        cout<<"Enter full year: ";
        cin>>year;
        cout<<"Enter month in number between 1 and 12: ";
        cin>>month;
        cout<<endl;
        printMonth(year, month);
        cout<<endl;
        cout<<endl;
        return 0;
}
```

Questions:

1. Identify all data types applied in a program.
2. Write the role of each data type you have identified in (a).

FOR ONLINE READING ONLY

3. Type the program on the Dev C++ text editor.
4. Compile the program.
5. Identify errors obtained after compiling the program.
6. State the errors obtained after compiling the program.

Exercise 4.7

1. What is wrong with the following program?

```
#include <iostream.h>
using namespace std; /
int main()
{
int y = 0;
cout << "xy= " << y << endl;
return 0;
}
```

2. Compile and run the following program; thereafter, explain the output.

```
#include <iostream>
using namespace std;
int main()
{
        cout << "Size of char: " << sizeof(char) << " byte" << endl;
        cout << "Size of integer: " << sizeof(int) << " bytes" << endl;
        cout << "Size of float: " << sizeof(float) << " bytes" << endl;
        cout << "Size of double: " << sizeof(double) << " bytes" << endl;
        return 0;
}
```

3. Examine the following C++ code and answer the questions that follows.

```
string str1;
string str2;
char ch;
cin >> str1;
str2 = str1;
cin >> ch;
str2[0] = ch;
cout << str1 << " -> " << str2 << endl;
```

FOR ONLINE READING ONLY

Questions.

(a) What is the output if the input is Akilimali J?

(b) What is the output if the input is Salome A?

(c) What is the output if the input is Abas H?

4. Write a program that asks the user to enter the length of the sides of a triangle and outputs the shape of the triangle.
5. What is the difference between the character 'n' and the character '\n'?

Variables and constants

Variables and constants are fundamental concepts in programming that enable you to store and manipulate data. Variables are containers that hold values that can change during execution, allowing flexibility in our code. On the other hand, constants are fixed values that cannot be modified once defined, providing stability and consistency in your programs. Understanding how to effectively use variables and constants is essential in developing robust and dynamic software solutions.

Variables

Consider that you have been told to store the number 5 and 2 respectively in the computer memory. Then, you are asked to add 1 to the first number and store the results to the memory. Afterward, you subtract the second number from the result you have obtained. To the computer, the same process can be conducted as follows:

```
x=5;
y=2;
x=x+1
result=x - y;
```

A variable is a name that defines a specific location in the computer memory. It is used to store data or information within a computer program. They are called variables simply because their values can be changed. Usually, variables store data of a specific data type; this means, to store a value into variable, you need to know its data type. Each variable in a program needs an identifier (unique name) to differentiate it from other variables. For example, from the given scenario x, y and "result" stands as variable identifiers.

Rules for naming variables in C++ programming

(a) Commas and blank spaces are not required in the variable name.

(b) The first character must be an alphabet or underscore, but not a number.

(c) Variables should not be too long. Make sure it does not exceed eight characters.

(d) Special symbols such as $, &, and # are not allowed within the variable names.

(e) Combination of numbers, alphabets, and underscores are allowed in the variable names.

FOR ONLINE READING ONLY

Variable declaration

Since variables represent a certain type of data, then, if you want to use a variable in a program, you are supposed to define its name and data type. That system of expressing variables is known as variable declaration. It is a call to the compiler to prepare a memory size for a variable in relation to its data type. To declare a variable, use the following syntax: *datatype variableName*.

Example of variable declaration:

```
int x;
double area;
char t;
```

x, *area* and *t* represent variable name, while *int*, *double* and *char* represent data types.

Note: If two or more variables are of the same data type, they can be grouped as **datatype** variable1, variable2, ……., variablen;. For example, **int** x, y, z;

Initialising variables

Initialising variables involves using the equal sign "=" to assign a value to the variable. The following syntax is used to initialise variable: *datatype variable = value;*.

An example of variable initialization is *int i = 56;*

Scope of variables in C++ programming

Based on their scopes, variables in C++ can be classified into three types. These are local, global, and static variables.

(a) Local variables

Local variables are declared within the function block or inside the compound statement. The following Program Example 4.8 demonstrates variable declaration in C++.

Program Example 4.8:

Variable declaration

```
function []
{
        int a;
        double b;
}
```

Variables a and b are defined within the block of function, and they can only be accessed inside that block. Hence, a and b represent local variables.

(b) Global variables

Global variables are declared before the main function *main()*. Such variables can be applied to any function within the program. Global variables are also known as external variables. The following Program Example 4.9 demonstrates the use of global variables in C++ program.

FOR ONLINE READING ONLY

Program Example 4.9:

Global variable declaration

```
int x=5, y=2;
main()
{
fun();
}
fun()
{
int sum;
sum = x + y;
}
```

Variables x and y in this program are global variables because they have been declared before the main function, *main()*, and are also used within the function.

(e) Static variables

Static variables are types of variables declared only once within a function, but they are used in a whole program. The keyword "static" is used to declare static variables. For Example:

static int i, j;
static float a, b;
static char name;

The following is a Program Example 4.10 of a C++ program that finds the area of the rectangle. The output is shown in Figure 4.24.

Program Example 4.10:

Finding the area of the rectangle

```
#include<iostream>
using namespace std;
int main()
{
        int l, w;              //Declaration of length (l) and width (w)as integers data type
        double area;                    //Declaration of area as double data type
        cout<<"Enter a length in metres: ";
        cin>>l; //Read value of length entered by the user
        cout<<"Enter a width in metres: ";
        cin>>w; //Read value of width entered by the user
        area=l*w;
        cout<<"The area of the rectangle is "<<area<<"square metres"<<endl;
        return 0;
}
```

FOR ONLINE READING ONLY

Output:

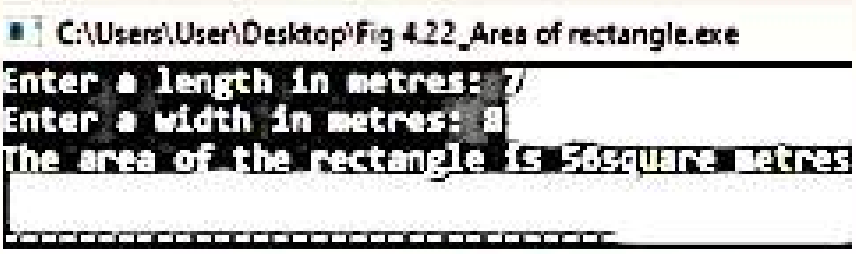


Figure 4.24: *Output of the program*

Activity 4.3:

Data type and variable in a program

Examine the following C++ program, and then answer the questions that follow:

```
#include<iostream>
using namespace std;
int main()
{
char ch;
cout<<"Enter lower case alphabet: ";
cin>>ch;
ch=ch-32;
 cout<<"\nThe uppercase alphabet is "<<ch;
cout<<endl;
return 0;}
```

Questions

1. Type the program code above on any C++ text editor program.
2. Compile and run the program to display an output.
3. Re-type the program to produce the same output without using variable and data type.
4. Compare the two programs and explain how difficult it is to practice this program without using variable and data type.

Exercise 4.8

1. What is the advantage of data type in a variable declaration?
2. Give the output of the following code fragment:

   int x;

   x = 'B';

   cout << x << '\n';
3. Write a C++ program that reads two numbers entered by the user from the keyboard, calculates their sum, and displays the result on the screen.

Constants

During program execution, a value may change. To avoid this change, you can use constants. The constant represents a value whose data cannot change during program execution. An example of the constant is Pi = 3.14; you do not need to retype it if you want to use it more than once; instead, you can declare it as a constant.

The syntax required to declare constant is **const** *datatype constantName = value;*

FOR ONLINE READING ONLY

Example of constant declaration:

```
const int averageMarks = 50;
const double bankRate = 0.15;
```

The following Program Example 4.11 demonstrates an example of a C++ program that uses constant. The program computes the area of the circle, and it displays the output on the screen. The output is shown in Figure 4.25.

Program Example 4.11:

A C++ program that uses constant

```
#include<iostream>
using namespace std;
int main()
{
        const double PI = 3.142;                    //declare PI as a constant
        double radius, area;
        cout<<"Enter a radius: ";
        cin>>radius;
        area = PI*radius*radius;                    //compute the area
        cout<<"The area of the circle is "<<area<<endl; //display the output
        return 0;
}
```

Output:



Figure 4.25: Output of the program

FOR ONLINE READING ONLY

Exercise 4.9

1. Write a C++ program that will enable customers to know the Value-Added Tax to be paid for any purchase price of the commodity. The VAT rate of 18.2% should be declared as a constant, and the formula should be (VAT = 18.2% × Purchase prices).
2. Differentiate the terms variable and constant as applied in C++ programming
3. Create a C++ program that prompts a user to enter a radius of the sphere. The program should calculate and display the volume of the sphere as output. (Hint: The formula for calculating the volume of the sphere is $\frac{4}{3}\pi r^3$, r represents the radius of the sphere). Use Table 4.6 in your declaration.

Table 4.6: Declaration

Data name	Data type	Declaration type
Pie (π) =3.1415926	Double	Constant
Radius	Float	Variable
Volume	Double	Variable

4. (a) What are the benefits of using constants in a C++ program?

   (b) What is wrong with the following C++ program? Explain.

```
#include<iostream>

#define PI = 3.142;
using namespace std;
int main[]
{
        double temp = PI*5;
        cout<<temp<<endl;
        return 0;
}
```

5. Consider the following program:

```
#include <iostream.h>
using namespace std;
int main()
{
int a;
int b;
a=2;
b=5;
a=b;
b=4;
cout << "a= " << a << endl;
cout << "b= " << b<< endl;
return 0;
}
```

   (a) What is the output of this program?

   (b) Using a diagram, illustrate how memory changes during variable assignment.

FOR ONLINE READING ONLY

Input and output streams

The tendency of reading and writing data in the C++ programming language is controlled by the input and output streams. Input and output streams enable a user to interact with the program by displaying output on the screen and getting the user's input through the keyboard. Input stream, as shown in Figure 4.26, uses extraction operator (>>) to read data from the source/ keyboard to the program. Output stream, as shown in Figure 4.27, uses insertion operator (<<) to display a program to the screen/destination. The object of the output stream required to display data is called "cout" while "cin" is used by the input stream to accept or read data. The objects "cout" and "cin" are declared within the header file "iostream."

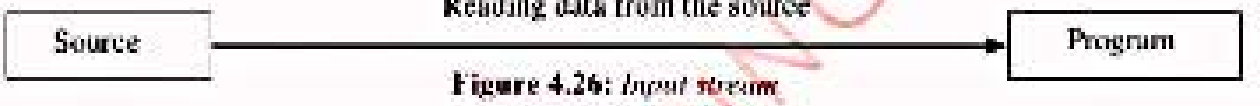


Figure 4.26: *Input stream*

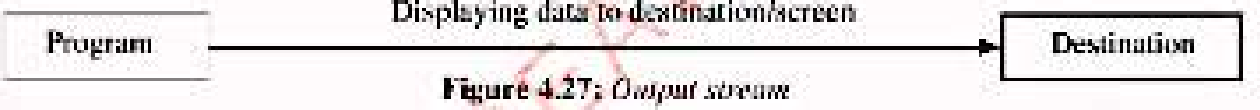


Figure 4.27: *Output stream*

Note: The insertion and extraction operators can be used more than once in one stream of data. This signifies that, you can use more than one insertion operator to display different data using one output stream. You can also use more than one extraction operators to request more than one data from the user.

For example:
```
cout<< "Ashura is"<< age<< "years old"<<endl;
cin>>firstNumber>>secondNumber;
```

The input stream "cin" will process the data entered when the user presses the RETURN key. The weakness of input stream "cin" is that it cannot hold blank space characters. The function "getline" can be used instead of "cin" to solve this problem. The following Program Example 4.12 demonstrates the uses of "cin" and "cout" commands in a C++ program. The output is shown in Figure 4.28.

FOR ONLINE READING ONLY

Program Example 4.12:

The use of "cin" and "cout" commands

```
int main()
{
        int a, b;
        cout<< "enter first and second number";
        cin>>a>>b;
        cout<< "The product is "<<a*b<<endl;
}
```

Output:

```
C:\Users\User\Desktop\Comp_Science\Comp
enter first and second numbers
4
The product is 20
```

Figure 4.28: *Output of cin and cout commands*

The following Program Example 4.13 is for 'getline' command in a C++ program which is used to solve the problem in which 'cin' cannot hold blank space characters. The output is shown in Figure 4.29.

Program Example 4.13:

The use of "getline"

```
#include<iostream>
#include<string>
using namespace std;
int main()
{
        string myname;
        cout<<"What is your name? ";
        getline(cin, myname);
        cout<<"Hello "<<myname<<endl;
        return 0;
}
```

Output:

```
C:\Users\User\Desktop\Comp_Science\Com
What is your name? Mwananzengo
Hello Mwananzengo
```

Figure 4.29: *Output of the 'getline' command*

FOR ONLINE READING ONLY

Operators, expressions and assignments

Operators, expressions, and assignments are fundamental components in programming languages, enabling the manipulation and evaluation of data. Operators perform operations on variables and values, forming expressions that produce new values. Assignments then store these resulting values into variables, facilitating dynamic and efficient data handling within a program.

Operators

Once you have an idea of variables and constants, you can decide to operate them in different ways. For that case, the C++ programming language introduces signs or symbols used to link, join, or conduct two or more data operations. Operators are required by the operands to conduct operations. There are binary operators that involve two operands and unary operators that involve only one operand.

Types of operators

C++ programming languages use different types of operators depending on the operation to be conducted. Such operators include: Arithmetic operators, Assignment operators, Relational operators, Logical operators, Increment and decrement operators, Compound assignment operators, and Short-hand assignment operator.

(a) Arithmetic operators

Arithmetic operators include arithmetic operations such as addition, subtraction, multiplication, and division. Table 4.7 shows arithmetic operators and their meanings.

Table 4.7: *Arithmetic operators*

Operator	Meaning
+	Addition
–	Subtraction
*	Multiplication
/	Division
%	Modulo division

Note: Modulo division is the division that gives the remainder after one integer divide another integer. Example 7%4 = 3.

The following C++ Program Example 4.14 demonstrates how to change seconds into minutes, hours and seconds. The output is shown in Figure 4.30.

FOR ONLINE READING ONLY

Program Example 4.14:

Demonstration of Modulo division

```
#include<iostream>
using namespace std;
int main()
{
        int sec, min, hrs;
        cout<<"Enter seconds: ";
        cin>>sec;
        min = sec/60;
        hrs = min/60;
        int remainSeconds = sec%60;
        cout<<sec<<" seconds is "<<hrs<<" hours, "<<min<<" minutes
        and  "<<remainSeconds<<" seconds"<<endl;
        return 0;
}
```

Output:



Figure 4.30: *Output of the program*

FOR ONLINE READING ONLY

(b) Assignment operators

Assignment operators are responsible for assigning value to the variable. The value assigned can either be constant, variable or expressions. The syntax used to assign value to the variable is *variable = value;* for example x=10; or y=a+1. The assignment operations always take the value from right to left. This means that, if two variables *a* and *b* are assigned as a = b, then, the value of *a* will be replaced by the value of *b* and not otherwise. The following Program Example 4.15 demonstrates the assignment operations in C++ programming. The output is shown in Figure 4.31.

Program Example 4.15:

Assignment operations in C++

```
#include<iostream>
using namespace std;
int main()
{
        int a, b;
        a=34;
        b=22;
        a=b;
        b=7;
        cout<<"a = "<<a<<endl;
        cout<<"b = "<<b<<endl;
        return 0;
}
```

Output:



Figure 4.31: Output of the program

The output of the given program in Figure 4.34 shows that variable a assigned value of b, which is 22, and variable b replaced its value with 7.

(c) Compound assignment operators

If you want to change the value of the variable by conducting an operation on the value currently stored in the variable, you can use compound assignment operators. Table 4.8 shows compound assignment operators and their respective meanings.

Table 4.8: Compound assignment operators

Operator	Example	Meaning
+ =	a + =b	a = a +b
=	a – =b	a = a – b
* =	a*=b	a = a*b
/ =	a/=b	a = a/b
% =	a%=b	a = a%b

FOR ONLINE READING ONLY

The following Program Example 4.16 demonstrates the use of compound operators. The output is shown in Figure 4.32.

Program Example 4.16:

Compound assignment operators

```
#include<iostream>

using namespace std;

int main()
{
        int x, y = 5;
        x = y;
        x+=8;
        cout<<"The value of x is "<<x<<endl;
        return 0;
}
```

Output:



Figure 4.32: *Output of the program*

(d) Increment and decrement operators

Increment (++) and decrement (--) operators are unary operators that can add or subtract one to the variable. The increment operators add one to the variable while decrements subtract one. The increments and decrements operators can be written before variable as prefixes and after a variable as the suffixes. The increment and decrement operators are mainly used to iterate integers and floating data types. Table 4.9 shows increment and decrement operators and their meaning.

Table 4.9: *Increment and decrement operators*

Operator	Meaning	Explanation
++i	Pre increment	Increment by 1 and then conduct operation
- -i	Pre decrement	Decrement by 1 and then conduct operation
i++	Post increment	Conduct operation and then increment by 1
i--	Post decrement	Conduct operation and then decrement by 1

FOR ONLINE READING ONLY

The following Program Example 4.17 demonstrates the use of increment and decrement operators. The output is shown in Figure 4.33.

Program Example 4.17:

Increment and decrement operators

```
#include<iostream>
using namespace std;
int main()
{
        int x = 25;
        cout<<"x++ = "<<x++<<endl;
        cout<<"x-- = "<<x--<<endl;
        cout<<"++x = "<<++x<<endl;
        cout<<"--x = "<<--x<<endl;
        return 0;
}
```

Output:



Figure 4.33: *Output of the program*

(e) Relational operators

Sometimes, in programming, we need to compare two values to know if they are equal or not, greater than or less than. The values can either be variables, constants or expressions.

C++ programming language evaluates the comparison between two values using relational operators. Relational operators can help you in making the decision. You can use relational operators to compare the height of two trees, the ages of two students or the prices of two items. Usually, the result obtained after evaluating relational operators is either true or false. Table 4.10 shows relational operators, their meaning, examples and return values.

Table 4.10: *Relational operators*

Operator	Meaning	Example	Evaluation	Return value
<	less than	16 < 20	True	1
< =	less than or equal to	10 < = 10	True	1
>	greater than	5 > 11	False	0
> =	greater than or equal to	45 > = 28	True	1
= =	equal to	6 = = 7	False	0
! =	is not equal to	8! = 8	False	0

FOR ONLINE READING ONLY

The following Program Example 4.18 demonstrates the use of relational operators. The output of the program is shown in Figure 4.34.

Program Example 4.18:

Use of relational operators

```
#include <iostream>
using namespace std;
int main()
{
        bool result;
        cout<<"Condition ; Return value"<<endl;
        result = (5 == 5);
        cout<<"5==5"<<"   :    "<< result<<endl;
        result = (8>13);
        cout<<"8>13"<<"   :    "<< result<<endl;
        result = (22<=12);
        cout<<"22<=12"<<" :    "<< result<<endl;
        result = (15!=17);
        cout<<"15!=17"<<" :    "<< result<<endl;
        return 0;
}
```

Output:



Figure 4.34: *Output of the program*

(f) Logical operators

A relational operator involves only one condition at a time. To combine two or more conditions, you can use logical operators. Logical operators include, !, || and && operator. Table 4.11 shows logical operators, their meaning, examples and return values.

Table 4.11: *Logical operators*

Operator	Meaning	Example	Return value
!	Logical NOT	45! = 67	1
||	Logical OR	(6 > 3) || (8! = 8)	1
&&	Logical AND	(5 < = 3) && (9 > = 5)	0

FOR ONLINE READING ONLY

Logical NOT operator "!" involves only one value. It performs the Boolean operation by reversing the value of the expression to be operated. If the value of the expression is true, it changes that value to false. Otherwise, if the expression is false, it changes that value to true. Logical OR operator "||"is used to conclude a set of the conditions. If either of the condition is true, then the statements are executed. Logical AND operator "&&" is used to make a conclusion from the conditions where the execution takes place when all conditions are true. The following C++ Program Example 4.19 demonstrates the operation of logical operators. The output is shown in Figure 4.35.

Program Example 4.19:

Use of relational operators

```
#include <iostream>
using namespace std;
int main()
{
        bool answer;
        cout<<"Condition  ; Return value"<<endl;
        answer = ((9>2)&&(17>2));
        cout<<"(9>2)&&(17>2)"<<"          :   "<< answer<<endl;
        answer = ((8>13)||(5!=3));
        cout<<"(8>13)||(5!=3)"<<"         :   "<< answer<<endl;
        answer = ((15!=20)&&(27>30));
        cout<<"(15!=20)&&(27>30)"<<"      :   "<< answer<<endl;
        return 0;
}
```

Output:



Figure 4.35: *Output of the program*

FOR ONLINE READING ONLY

Operator precedence in C++

Table 4.12 illustrates the level of precedence of the operators in C++

Table 4.12: *Operator precedence in C++*

Priority	Operators
Highest Priority ↑	(,)
	++, -- (as postfix), new, (type).typeof, sizeof
	++, -- (as prefix), +, - (unary), !, ~
	*, /, %
	+ (string concatenation)
	+, -
	<<, >>
	<, >, <=, >=, is, as
	==, !=
	&, ^, |

Expressions

In C++, an expression is a combination of variables, constants, operators, and functions that are evaluated to produce a value. Expressions are the building blocks of C++ programs, used to perform calculations, make decisions, and manipulate data. Here are some key points about expressions:

Types of expressions

(a) Arithmetic expressions

These include basic arithmetic operations such as addition (+), subtraction (-), multiplication (*), and division (/).

For, example: *int result = a + b - c * d / e;*

(i) Mathematical expressions in C++ programming

Like other languages, the C++ programming language provides rules through which different data, including variables, constants and operators, can be combined. Such an interconnection of data is known as expression. During programming, the arithmetic expression should be translated into C++ expression so that a compiler can understand what is programmed.

For example, the expression $10\left|\frac{2x}{3}-\frac{1}{3}\right|$ can be translated into C++ expression as 10*((2*x)/3 – 1/3).

FOR OUTLINE READING ONLY

(ii) Operator precedence in mathematical expressions

Levels of precedence between operators, both in arithmetic expression and C++ expressions, remain the same. That level starts from the operation enclosed by brackets, followed by arithmetic operators out of the brackets. Priority is given to division and multiplication, followed by addition and subtraction. If the expression includes several arithmetic operators of the same type the operation will be conducted from left to right.

Activity 4.4:

Writing mathematical expression using C++ programs

Practice on how to write mathematical expressions in C++ programs by following the steps below:

Step 1: Type the following expression in (a) word processor and (b) C++ text editor. Compare the arithmetic operation typed in both (a) and (b).

$$z = \frac{5x-1}{6} - \frac{7}{x}(a+2b-c)(y-5) + 10\left[\frac{4}{x} + \frac{5+2x}{y-1}\right]$$

Step 2: If x=1, y= 2, a = 3, b = 5 and c=6, use a calculator to find the value of z.

Step 3: Write a C++ program to find the value of z using the values given in step 2.

Step 4: Comment on the results obtained in step 2 and step 3.

(b) Relational expressions

These compare two values and return a boolean result (true or false). Common relational operators include ==, !=, <, >, <=, and >=.
For exampele, in program code: *bool isEqual = (a == b);*

(c) Logical expressions

These combine multiple boolean expressions using logical operators such as && for (logical AND), || for (logical OR), and ! for (logical NOT).
For example, in program code: *bool isTrue = (a > b) && (c < d);*

(d) Bitwise expressions

These operate on the binary representation of data using operators like & for (bitwise AND), | for (bitwise OR), ^ for (bitwise XOR), ~ for (bitwise NOT), << for (left shift), and >> for (right shift).

FOR ONLINE READING ONLY

For example, in program code: *int bitwiseResult = b & c;*

Function call expressions

These involve calling a function and using its return value in an expression. An example of function call expression is like program code: *int result = sqrt(a) + pow(c, d);*

Assignments

Assignments in C++ are used to store the result of an expression into a variable. The assignment operator (=) is the most common operator used for this purpose. Here are some important aspects of assignments:

Basic assignment

The value on the right-hand side of the = operator is evaluated and stored in the variable on the left-hand side. For example, in program code: *int a = 12;*

Compound assignment operators

These operators combine an arithmetic operation with an assignment. Examples include +=, -=, *=, /=, and %=. For example, in program code:

```
int a = 14;
a += 7; // Equivalent to a = a + 7;
```

Chained assignment

Multiple assignments can be chained together, allowing the same value to be assigned to multiple variables. For example, in program code:

```
int a, b, c;
a = b = c = 20;
```

Type conversion

During assignment, if the types of the variable and the expression do not match, implicit or explicit type conversion may occur. For example, in program code:

```
double x = 5; // Implicit conversion from int to double

int y = static_cast<int>(x); // Explicit conversion from double to int
```

Lvalue and Rvalue

The variable on the left side of the assignment operator is called an lvalue (left value), this refers to a location in memory. The expression on the right side is an rvalue (right value), this is a value that have no specific memory location. For example, in program code:

```
int a = 10; // 'a' is an lvalue, '10' is an rvalue
```

Activity 4.5:

Creating a program that combines expressions and assignments

Use the knowledge you have learnt in previous sections, create a simple program that combines expressions and assignments in a C++ program. Once you have created, highlighting how they are used to manipulate and store data in a C++ program

FOR NON-INTERACTIVE READING ONLY

Exercise 4.10

1. Write a C++ program that can enable a user to convert Fahrenheit to Centigrade degree. Use the formula, *Centigrade=5/9 (Fahrenheit-32).*
2. Write the following program on C++ text editor; then, display the output.

```
#include<iostream>
using namespace std;
int main()
{
cout<<"2%3 = "<<2%3;
return 0;
}
```

3. Write a C++ program that enables the user to calculate the gradient (slope) of the straight line. A program will prompt the user to enter two points (x1, y1) and (x2, y2). Then, a program will find and display a slope. Use the formula:

$$\text{Slope}=\frac{y_2-y_1}{x_2-x_1}$$

4. What are the functions of "cout" and "cin" in writing a C++ program?
5. Using output stream "cout" , write a C++ program that displays the following table as shown in Figure 4.36:

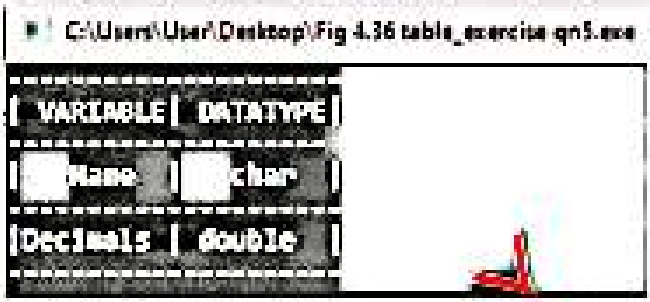


Figure 4.36: *Output*

6. Construct a C++ program that takes a character as an input from the keyboard and converts it into a capital letter or small letter depending on its form. A program should check the validity of the entered character and display a message "The entered character is not an alphabetical letter" if the input is not a letter.
7. Give the output of the given program

```
#include <iostream>
using namespace std;
int main()
{
cout << "1 + 5 = " << 1 + 5 << endl;
cout << "13 + 89 = " << 13 + 89 << endl;
cout << "30 - 20 = " << 30 - 20 << endl;
cout << "4 - 90 = " << 46 - 90 << endl;
cout << "2 * 7 = " << 2 * 7 << endl;
cout << "5 / 2 = " << 5 / 2 << endl;
cout << "21 / 7 = " << 21 / 7 << endl;
cout << "35 % 6 = " << 35 % 6 << endl;
cout << "4 % 6 = " << 4 % 6 << endl;
return 0;
}
```

FOR ONLINE READING ONLY

Control structures

If there is any repetition or condition made within the program, the order of execution should change based on condition and repetition. Control structure shows how execution of data flow within a program step-by-step. It also simplifies ways of solving problems within a program by avoiding typing many lines of code. There are three types of control structures used in C++ programming languages: Sequential, selection or decision, and iteration or looping control structure.

Sequential control structure

Sequential control structure explains step-by-step the execution of data within a program. It defines how data is executed sequentially by considering the order provided by the program. The program conducts different series of data flow from top to down by creating data link and dependency. Sequential structure controls data reading, writing, arithmetic calculation, assignment of variables, etc. Figure 4.37 shows a flow chart of a sequence control structure.

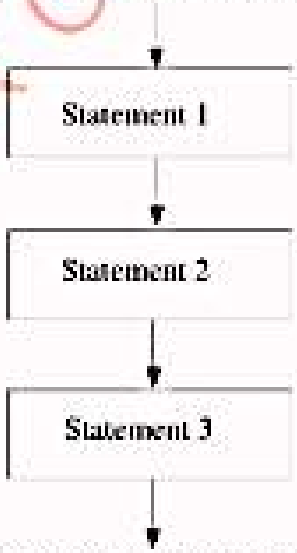


Figure 4.37: *Sequential control structure*

Selection control structure

C++ programming language uses selection or decision control structure to fulfill a certain condition. This occurs when the sequence of instructions is determined through the condition. In a selection control structure, execution takes place when the condition is satisfied whereby one side is executed when the condition is true, and another side is executed when the condition is false. Consider a program to register citizens for presidential voting. The program will prompt a citizen to enter their ages and then check if the age is greater than 18. Then, registration form will open; otherwise, the form will not open.

There are two types of decision control structure, namely If statement, and Switch-case statement in C++ programming language.

(a) If statement

If statement is the type of decision control structure used to test the logical flow of statement execution and determine whether it is true or false. If statement involves IF…THEN, IF …ELSE and Nested IF statement.

IF….THEN

If there is only one option to be tested in your program, you can use IF….. THEN statement to test your condition.

IF….THEN displays the result of the given option when the condition is

FOR ONLINE READING ONLY

true. Otherwise, the program will stop executing. This means within IF…THEN statement, other options are ignored.

The following is the syntax of IF…. THEN statement.

```
If(condition)
{
statements;
}
```

Figure 4.38 shows the execution flowchart of IF….THEN statement.

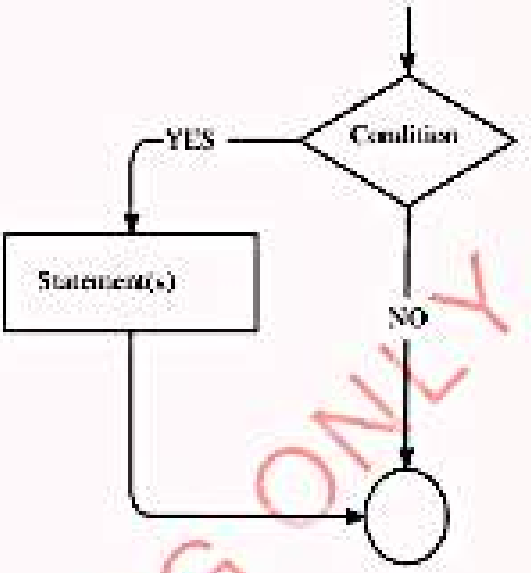


Figure 4.38: *IF… THEN flowchart*

The following code in Program Example 4.20 shows how you can write a program to find average pass mark of the students to join form five PMC by using 'If…Then', the output is shown in Figure 4.39.

Program Example 4.20:

Using 'If …Then'

```
#include<iostream>
using namespace std;
int main()
{
        int averageMark;
        cout<<"Enter marks of the student: ";
        cin>>averageMark;
        if(averageMark>=70)
{
cout<<"You have been selected to join form 5 PMC"<<endl;
}
        return 0;
 }
```

Output:

```
C:\Users\User\Desktop\Fig 4.39 if_then.exe
Enter marks of the student: 70
You have been selected to join form 5 PMC
```

Figure 4.39: *Output of program for marks*

FOR ONLINE READING ONLY

IF…..ELSE statement

If you have two options to test in C++ programming, *IF... THEN statement* is not a good choice to use. Instead, you can use IF…ELSE statement. IF…ELSE statement specifies two decision-making statements where one is true and the other is false. It carries a logical test among two options and selects one option that correlates to the condition. The application example of IF…..ELSE statement as shown in Figure 4.40 is to set condition when you need to make choice among the people by considering their gender, either male or female. The syntax of *if..else statement* is given as:

```
if(condition)
{
statement(s); // this block will execute when condition is true.
}
else
{
statement(s); // this block will execute when condition is false
}
```

Figure 4.40 shows the execution flowchart of IF….ELSE statement.

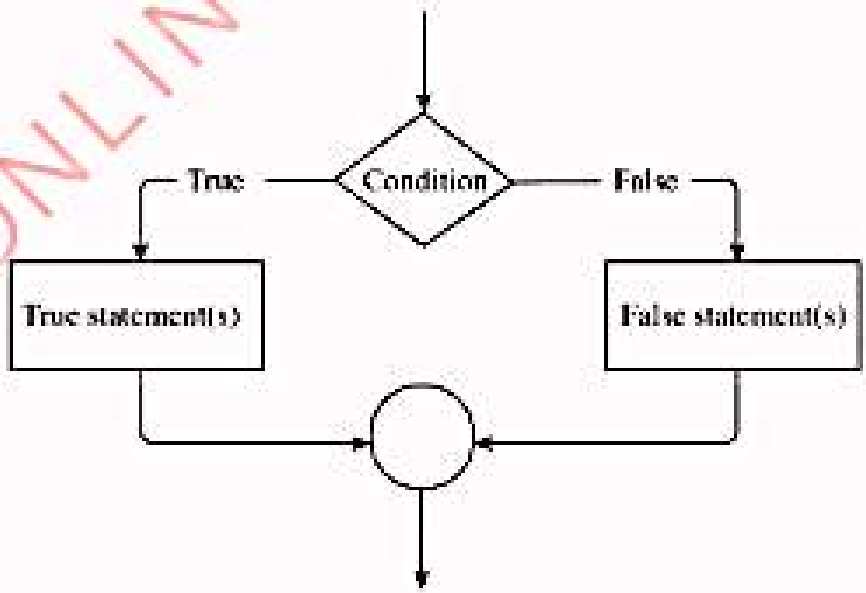


Figure 4.40: *If...else flowchart*

You can learn from the following Program Examples 4.21 on how to use the 'If else' to write the program.

FOR ONLINE READING ONLY

Program Example 4.21:

Using 'if ...else' to write a program

```
#include<iostream>
using namespace std;
int main()
{
        int marks;
        cout<<"Enter marks of the student: ";
        cin>>marks;
        if(marks>=50)
{
    cout<<"You have been selected to join form five"<<endl;
    }
else
    {
     cout<<"You are not selected"<<endl;
    }
     return 0;
 }
```

Output:

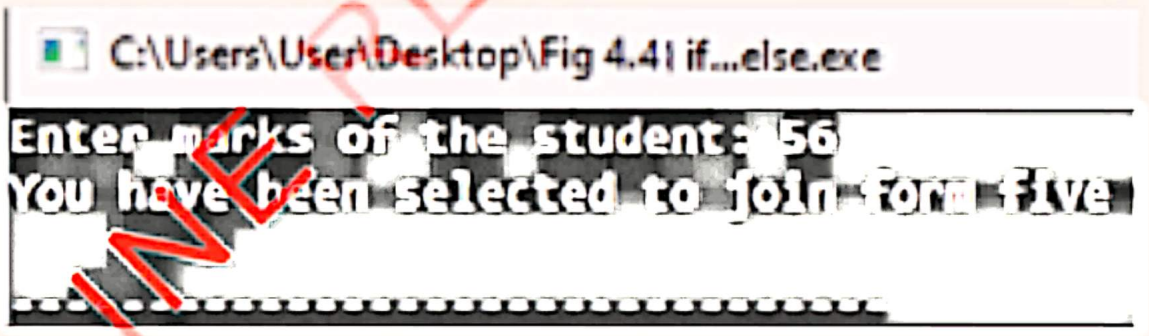


Figure 4.41: *Output of the program*

Note: If there is only one statement within the if block, using braces is an option. For example, in the program IF...*ELSE* statement can be written as:

```
if(marks>=50)
cout<<"You have selected to join form 5"<<endl;
else
cout<<"You are not selected"<<endl;
```

FOR ONLINE READING ONLY

depending on its score. The syntax of the *nested if* statement.

```
If (condition 1)
{
// this block will execute when condition 1 is true.
statement 1
}
else if (condition 2)
{
//this block will execute when condition 2 is true.
statement 2;
}
else if (condition 3)
{
//this block will execute when condition 3 is true.
statement 3;
}
else
{
//this block will execute if no any condition is true.
default-statements;
}
```

Figure 4.42 shows the execution flowchart of *Nested if* statement.

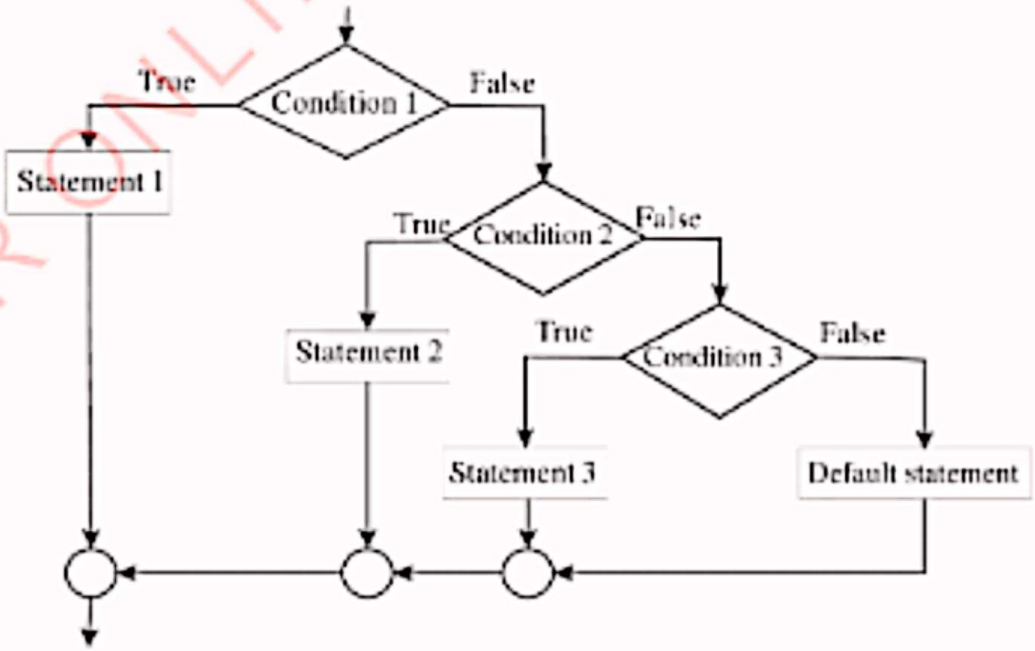


Figure 4.42: *Nested if ....flow chart*

FOR ONLINE READING ONLY

In order to practice on the use of *Nested if*...to write a program, learn from the following Program Example 4.22 that demonstrates the use of Nested if statement. The program enables users to know the type of the game through the number of players. The output of the program is given in Figure 4.43.

Program Example 4.22:

The use of Nested if statement

```
#include<iostream>
using namespace std;
int main()
{
        int n;
        cout<<" Choose 5, 6, 9 or 11 players to know type of the game "<<endl;
        cout<<"Enter number of players to know type of the game: ";
        cin>>n;
        if(n==11)
        cout<<"Football"<<endl;
        else if(n==5)
        cout<<"Basketball"<<endl;
        else if(n==6)
        cout<<"Volleyball"<<endl;
        else if(n==9)
        cout<<"Baseball"<<endl;
        else
        cout<<"Wrong choice"<<endl;
        return 0;
        }
```

Output:

```
C:\Users\User\Desktop\Fig 4.43_nested...if.exe
Choose 5, 6, 9 or 11 players to know type of the game
Enter number of players to know type of the game: 11
Football
```

Figure 4.43: *Output of the program*

FOR ONLINE READING ONLY

Exercise 4.11

1. Identify errors made in the following program

```
if (marks > = 60)
        Grade = 'D';
else if (marks > = 70)
        Grade = 'C';
else if (marks > = 80)
        Grade = 'B';
else if (marks > = 90)
        Grade = 'A';
else
        Grade = 'F';
```

2. Use *if statement* to create a C++ program that allows the user to enter five numbers. Then, the program arranges the number from the smallest to the largest number.
3. Write the program that prompts the user to enter a month and year. The program should display the number of days of that month as output.
4. Study the following C++ program that prints one line of the text, then answer the questions that follows:

```
#include <iostream>
Using namespace std;
int main()
{
int input;
cin >> input;
if (input < 11) {
if (input != 6)
cout << "wow ";
else
input++;
}
else {
if (input == 18)
input += 11;
else
cout << "Oyaaa!";
}
cout << input << '\n';
}
```

What will the program print if the user enters the following numbers as inputs? (a) 4, (b) 20, (c) 6, (d) 18, (e) -6

5. Write a C++ program that requests six integer values from the user. The program should give the message to the user as follows: if any of the values entered are duplicates, the message output should read "DUPLICATE VALUES"; otherwise, it prints "ALL ARE UNIQUE VALUES".

(b) Switch case statement

If you have many options in a condition statement, you need more *if...else* statement within *nested...if* statement. Overusing *if...else* for more options makes a program very difficult to design. For that case, to reduce lines of codes, C++ programming

FOR ONLINE READING ONLY

language introduces a *switch-case* statement to solve this problem. The syntax of the *Switch case* statement is shown as follows:

```
switch(variable)
    {
    case 1: statement 1;
    break;
    case 2: statement 2;
    break;
    case 3: statement 3;
    break;
    default: default statement
    break;
    }
```

Figure 4.44 shows the execution flowchart of the *switch case* statement.

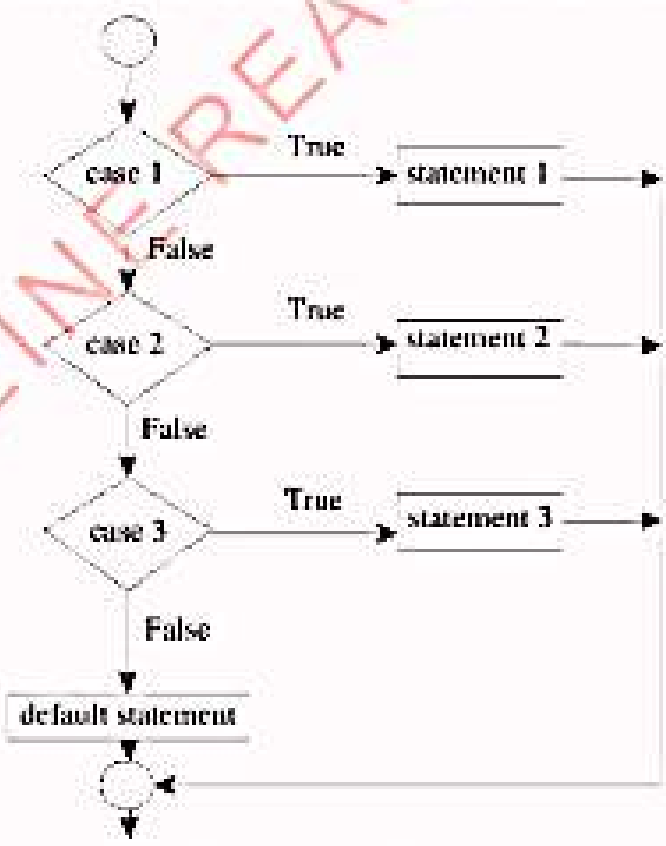


Figure 4.44: *A flowchart for a Switch case statement*

Note:

It is a good practice in programming to write cases in sequential order and end with default case, but it is not a must to follow that order. The switch expression should be enclosed in brackets.

FOR ONLINE READING ONLY

Program Example 4.23:

Identifying the type of games using a switch case statement

```
#include<iostream>
using namespace std;
int main()
{
        int n;
        cout<<"Enter number of players to know type of the game: ";

        cin>>n;
switch(n)
{
        case 5:
        cout<<"Basketball"<<endl;
        break;

        case 6:
        cout<<"Volleyball"<<endl;
        break;
        case 9:
        cout<<"Baseball"<<endl;

        break;
        case 11:
        cout<<"Football"<<endl;

        break;
        default:
        cout<<"Select 5, 6, 9 and 11 only"<<endl;
}
return 0;
}
```

Output:

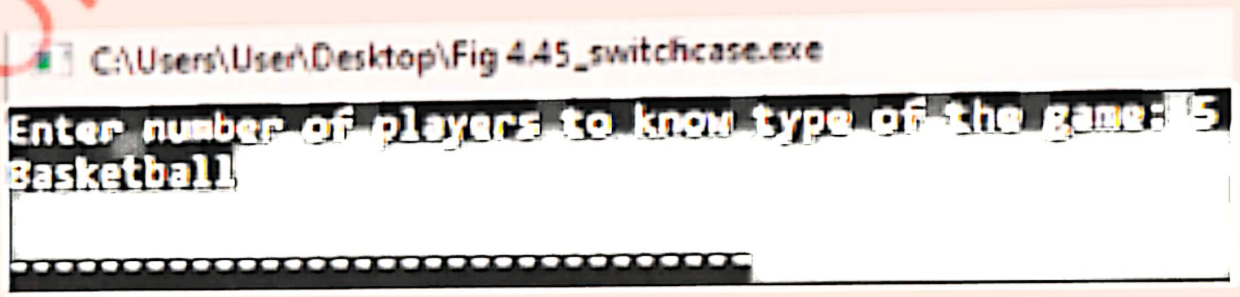


Figure 4.45: *Output of the program*

FOR ONLINE READING ONLY

4. Rewrite the following if statement by using the *switch-case* statement. Then, draw a flowchart of the given switch-case statement.

```
if (k==1)
m+=5;
else if (k==2)
m+=10;
else if (k==3)
m+=16;
else if (k==4)
m+=34;
```

5. Considers the following code fragment.

```
int x;
cin >> x;
switch (x + 3) {
case 5:
cout << x << '\n';
break;
case 10:
cout << x - 3 << '\n';
break;
case 20:
cout << x + 3 << '\n';
break;
}
```

Determine the output of the above program when the user enters 3, 7, 9, 19 and 22 respectively.

Iteration or looping control structure

Suppose you are required to type the text "C++ programming" ten times, you will type the code *cout*<< "C++ *programming*"<<*endl;* ten times which is very tedious. Also, consider you are a teacher and you want to find the average marks of each student for a class of 100 students. To avoid repetition of performing the same task, C++ programming languages uses looping or iteration control structure. Looping control manages repeated executions in a program. The loop contains two parts: the body of the loop and the control statement. The control statement tests the condition provided, and it allows the repeated execution of the statements within the loop body. There are three types of looping or iteration used in the C++ programming language. These are: While loop, Do while loop and For loop.

While loop

The *while* loop allows a program to test the conditions first before performing the execution. If the condition is true, the loop is executed according to the loop body. Otherwise, the loop terminates. The syntax of the *while loop* is given as:

```
while (loop condition)
{
Statement (s);
}
```

Figure 4.46 shows the execution flowchart of the *while loop*.

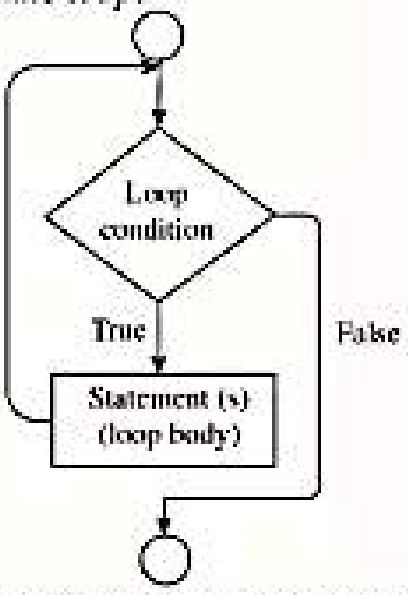


Figure 4.46: *A flowchart of a While loop*

FOR ONLINE READING ONLY

The following C++ Program Example 4.24 finds Greatest Common Factor (GCF) of two numbers. The output is shown in Figure 4.47.

Program Example 4.24:

Finding the Greatest Common Factor (GCF) of two numbers

```
#include<iostream>
using namespace std;
int main()
{
        int n1, n2;
        cout<<"Enter first number: ";
        cin>>n1;
        cout<<"Enter second number: ";
        cin>>n2;
        int gcf=1, x=2;
        while(x<=n1&&x<=n2)
        {
        if(n1%x==0&&n2%x==0)
        gcf=x;
        x++;
    }
cout<<"The greatest common factor (GCF) for "<<n1<<" and
 "<<n2<<" is "<<gcf;
      return 0;
}
```

Output:

```
C:\Users\User\Desktop\Fig 4.47_whileloop.exe
Enter first number: 24
Enter second number: 36
The greatest common factor (GCF) for 24 and 36 is 12
--------------------------------
```

Figure 4.47: *Output of the program*

FOR ONLINE READING ONLY

Do-while loop

The *do-while loop* allows the execution of the statement at least once, and then, evaluates the loop. If the loop condition is true, the execution continues until the condition becomes false. The do-while loop terminates when the loop condition is not met. It is useful where the loop body needs to be executed before the loop

condition is tested. For example, when you look for your favorite food at the restaurant. First, you will go, and if the food is attractive, you will continue to go. Otherwise, you will quit and go to another restaurant. The syntax of the *do-while* loop is given as:

```
do {
// The statement will be executed once even if the condition is false.
Statement(s);
}
while(loop condition);
```

Figure 4.48 shows the execution flowchart of the *do-while loop*.

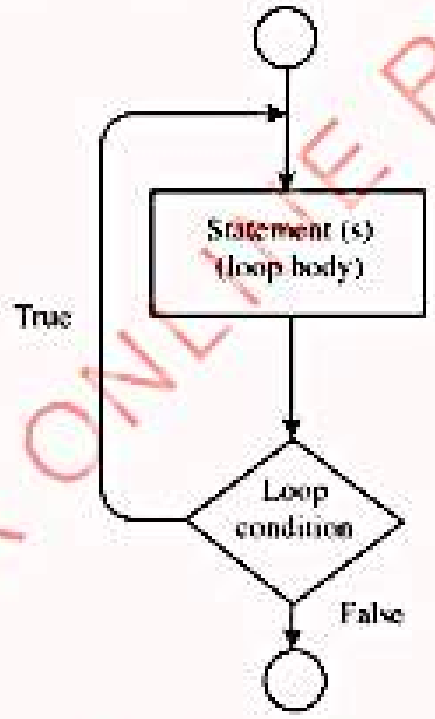


Figure 4.48: *do-while loop flowchart*

The following C++ Program Example 4.25 demonstrates the use of do…while loop. The output is shown in Figure 4.49.

Program Example 4.25:

The use of do-while loop

```
#include<iostream>
using namespace std;
int main()
{
        int sum = 0;
        int i = 0;

do
{
        sum+=i;
        cout<<"Enter an integer: ";
        cin>>i;
}
while(i!=0);
cout<<"The sum is "<<sum<<endl;
return 0;
}
```

Output:

```
C:\Users\User\Desktop\Fig 4.49-dowhile.exe
Enter an integer: 78
Enter an integer: 67
Enter an integer: 34
Enter an integer: 123
Enter an integer: 0
The sum is 302
```

Figure 4.49: *Output of the program*

FOR ONLINE READING ONLY

Table 4.13 shows the difference between while...loop and do-while...loop statements.

Table 4.13: *Difference between while....loop statements and do-while.....loop statements*

While loop	Do-while loop
Test the condition before execution; if the condition is true, the execution continues until the condition becomes false.	Body execution takes place once before testing the condition. Then, if the condition is true, the execution continues otherwise, the execution stops.
Testing of the loop body occurs at the top.	Testing of the loop body occurs at the bottom.
The loop stops execution if the condition is false.	Loop is executed at least once even though the condition is false

For loop

For loop is a predetermined type of loop. It allows a user to enter the required number of iterations to be done before looping. For example, the program will prompt a teacher to enter the number of students before calculating their average marks to prepare the number of iterations. *For loop* contain variables that control how many times the loop execution can occur. Unlike other control structures, *for...loop* contains three body features: initialisation, condition, and increment/decrement counter. The syntax of *for... loop* is given as :

```
for (initialisation; condition ;
increment/decrement counter)
{
loop body
}
```

Table 4.14 shows the function of each body part of the *for...loop*

Table 4.14: *Body parts of 'for....loop'*

Part	Function
Initialisation	It initialises a variable counter
Condition	It tests the loop condition
Increment/ decrement	It provides an increment or decrement variable counter.

Figure 4.50 shows the execution flowchart of the *for loop*.

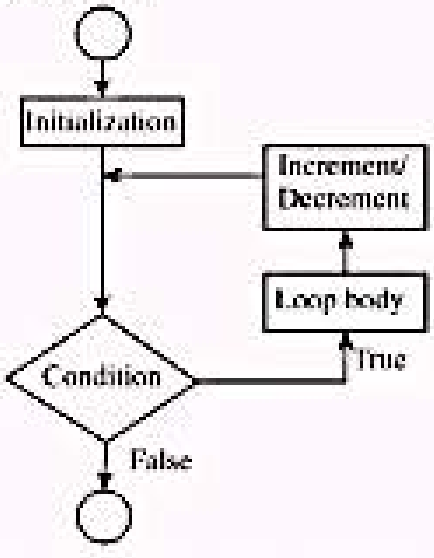


Figure 4.50: *A flowchart of "for loop"*

FOR ONLINE READING ONLY

The following C++ program example 4.26 computes the sum of 10 positive integers using for loop. The output is shown in Figure 4.51.

Program Example 4.26:

Finding the sum of numbers using for loop

```
#include<iostream>
using namespace std;
int main()
{
        int i, sum=0;
        for(i=1; i<=10; i++)
        {
sum=sum+i;
        }
        cout<<"The sum of 10 numbers is: "<<sum<<endl;
        return 0;
}
```

Output:

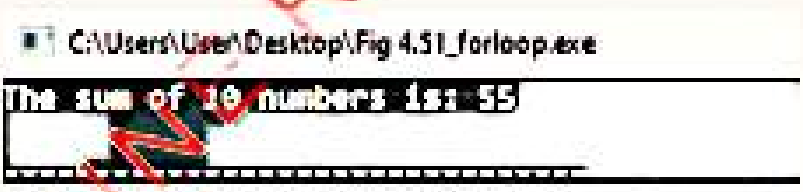


Figure 4.51: *Output of the program*

Activity 4.6:

Difference between the 'while loop, do....while loop and for...loop'

(a) Write three C++ programs to display students' results. Each program prompts a user to enter the number of students in the class and the marks of five (5) subjects for each student. Then, the program displays total marks and average marks for each student, using:

(i) while.....loop

(ii) do-while...loop

(iii) for.... loop

(b) Compare the results generated from each loop

(c) Comment on the results obtained from each loop.

FOR ONLINE READING ONLY

Exercise 4.13

1. Write a C++ program that displays the Pascal triangle, as shown in Figure 4.52, when a user enters the power of expansion.



Figure 4.52: *Output of the program*

2. Use *for loop* to write a C++ program which outputs the multiplication table as shown in Figure 4.53:



Figure 4.53: *Output of the program*

3. Create a simple calculator using C++ programming language to solve the factorial of a number and read an exponent when the user enters the base and power of a particular exponent. A calculator should allow the user to choose either to solve factorial or exponent of the number using a *switch case* statement.
4. Write a C++ program to display even numbers, odd numbers, or prime numbers which lie within a range entered by the user.
5. What is printed by the following code fragment?

```
int x = 0;
do {
cout << x << " ";
x++;
} while (x < 10);
cout << '\n';
```

Functions

Concept of functions

In programming, you can sometimes find the same instruction is applied repeatedly throughout the main program. The instruction does not iterate the solution according to the condition like what is happening within the loop. Still, it is used repeatedly through the program according to the user tasks. To avoid retyping such instructions many times within the main program, you can use a *function*. A function is a block of code in a program which run only when it is called. For example, a program has a block of code that calculates interest of money deposited by one customer in a bank. We do not want to rewrite the same instructions, if we wish

FOR ONLINE READING ONLY

to compute the interest of another person who also deposited money in the bank. Alternatively, we would rather move to a 'code section' that calculates interest and then return to where you left off.

Advantages of functions

(a) Using functions facilitates the reuse of the codes in writing computer programs

(b) The use of functions help to improve the program's readability because it reduces the complexity of the main function

(c) Functions help you to focus on just that part of the program and construct it, debug it, and perfect it.

(d) Different people can work on different functions concurrently.

The following is the syntax of typing a function:

```
returnValueType functionName (list
of parameters)
{
//Function body;
}
```

Types of functions

There are two types of functions applied in C++ programming, namely user-defined function and built-in/pre-defined function as shown in Figure 4.54.

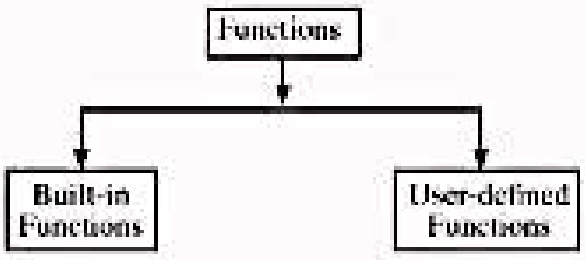


Figure 4.54: *Types of functions*

(a) User-defined function

A user-defined function is the type of function designed by a programmer according to the user's requirement. Programmers use the function to break down large programs to simplify debugging, testing, and maintenance of the program. A user-defined function that returns value is called 'returning function'. The function that does not return value is called 'void function'.

Elements of user-defined function

User-defined function contains three elements: function definition, function call and function prototype.

(i) Function definition

Function definition specifies all characteristics and elements required in a user-defined function. Function definition includes function header and function body. Function header includes data type or returns type, function name, and a list of parameters. On the contrary, function body involves all executable statements within it. Figure 4.55 shows the components of function definition.

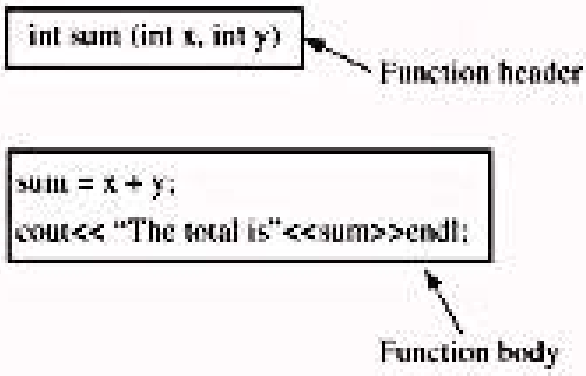


Figure 4.55: *Components of function definition*

FOR ONLINE READING ONLY

Note: Parameters are values which need to be passed into the function. They create a link or communication between the defined function and the main function. The following is the structure of the program to represent function parameters.

```
funct1(x,y)
{
// function body
.....
}
main()
{
funct1(a,b); //function calling
}
```

Where a, b are the actual parameters and x, y are formal parameters. Actual parameters are variables or expressions listed in call to a function. Formal parameters are variables in the function heading.

(ii) Function call

If you want to use a defined function within the main function, you have to call that function. During calling the function, you have to specify the name of the function, return value and parameters if they are included within the function. There are different ways of calling a function depending on whether the function returns a value or not.

Returning value function

For the function which returns a value, the function will be treated as a value. To call it, you have to define its data type and variable. An example of the calling function with the integer data type is *int total = sum(10,20);*

The following Program Example 4.27 demonstrates a C++ program that uses function call to find maximum number of the two numbers. The output is shown in Figure 4.56.

Program Example 4.27:

Using function to find a maximum number

```
#include<iostream>
using namespace std;
int maximum(int x, int y)
{
        int answer;
        if (x>y)
        answer=x;
        else
        answer = y;
        return answer;
}
int main()
{
        int a, b;
        cout<<"Enter two number to find which is maximum"<<endl;
        cin>>a>>b;
        int t = maximum(a,b);       //function call
        cout<<"The maximum between "<<a<<" and "<<b<<" is "<<t;
        return 0;
}
```

FOR NEW PEARSON EDITION

Output:



Figure 4.56: Output of the program

Function that does not return value (Void function)

A void function is a function that does not return any value. To call such a function, you will define *function name* without data type or variable. Program Example 4.28 demonstrates a C++ program that changes marks to grade using void function. The output is shown in Figure 4.57.

Program Example 4.28:

Changing marks grade using void function

```
#include<iostream>
using namespace std;
void grade(double marks)
{
        if (marks>80)
        cout<<'A';
        else if (marks>60)
        cout<<'B';
        else if (marks>40)
        cout<<'C';
        else if (marks>20)
        cout<<'D';
        else
        cout<<'F';
}
int main()
{
        cout<<"Enter marks: ";
        double marks;
        cin>>marks;
        cout<<"The grade is ";
        grade(marks);  // function call
        return 0;
}
```

Output:



Figure 4.57: Output of the program

FOR ONLINE READING ONLY

Note: There are two ways to call functions, which are function call by value (pass by value) and function call by reference (pass by reference).

(a) Call by value involves copying the values of actual parameters into the formal parameters.

(b) Call by reference involves copying the address of the arguments into actual parameters within the function.

Program examples 30 and 31 demonstrate call by value and call by reference. Program Example 4.29 demonstrates a C++ program that finds cube of the number. The output is shown in Figure 4.58.

Program Example 4.29:

Finding the cube of the number using function

```
#include<iostream>
using namespace std;
int cube(int x)
{
        x=x*x*x;
        return x;

}
int main()
{
        int a;
        cout<<"Enter number to find its cube ";
        cin>>a;
        cout<<"Cube of "<<a<<" is "<<cube(a);// call by value
}
```

Output:

```
C:\Users\User\Desktop\Fig 4.58 Find Cube of number.exe
Enter number to find its cube: 6
Cube of 6 is 216
```

Figure 4.58: *Output of the program*

You can use the following program example to learn on program that interchanges values entered by a user. The following Program Example 4.30 demonstrates a C++ program that interchanges value entered by the user. The output is shown in Figure 4.59.

FOR ONLINE READING ONLY

Program Example 4.30:

A program to interchange value entered by the user

```
#include<iostream>
using namespace std;
void increment(int &m)
{
        m++;
        cout<<"The address of m is "<<&m<<endl;
        cout<<"m inside the function is "<<m<<endl;
}
int main()
{
        int x = 1;
        cout<<"The address of x is "<<&x<<endl;
        cout<<"Before the call, x is "<<x<<endl;
        increment(x); //call by reference
        cout<<"After the call, x is "<<x<<endl;
        return 0;
}
```

Output:



Figure 4.59: *Output of the program*

(iii) Function prototype

Before calling any function into the main function, it should be defined. This is possible through writing function definition or function prototype. A function prototype is the function header without function body. The function body will be given, but the function should be introduced by writing its prototype. A function prototype contains return type, name, and parameters usually terminated by a semicolon. An example of a function prototype is: *int average (int n1, int n2);*

(b) Built-in function

Built-in functions are functions stored within standard library files. Such functions comprise mathematical, string manipulation, conversions, and console-based I/O function. The application of the built-in function should involve their header files. For example, <emath> for the mathematical functions and <cctype> for character manipulation. Table 4.15 shows some of the built-in function and their application.

FOR ONLINE READING ONLY

Table 4.15: *Built-in function*

Built-function	Application	Header	Example
sqrt (x)	Find the square root of x	<cmath>	sqrt (x)
abs (x)	Find absolute of x	<cmath>	abs (x)
exp (x)	Find the natural exponent of x	<cmath>	exp (x)
pow(x, y)	Find x power y	<cmath>	pow(x, y)
tolower(x)	Convert character to lower case	<cctype>	tolower(x)
toupper(x)	Convert character to upper case.	<cctype>	toupper(x)
isupper(x)	Returns true if x is uppercase letter, otherwise it returns false.	<cctype>	isupper(x)
islower(x)	Returns true if x is lowercase letter, otherwise it returns false.	<cctype>	islower(x)
isPrime	Returns true if x is prime numbr, otherwise it returns false.	<cmath>	isPrime

Casting

C++ uses casting technique to avoid implicit type conversion. Casting is used to convert data type from one data type to another through the use of a cast operator. The cast operator is also called type conversion or type casting. The following C++ Program Example 4.31 demonstrates the use of casting. The output is shown in Figure 4.60.

Program Example 4.31:

Casting

```
#include<iostream>
using namespace std;
int main()
{
        float l, w; //Declaration of length and width as integers data type
        double area; //Declaration of area as double data type
        /* This program asks you to enter length and with in float,and
        displays the results in (i) float and (ii) integer
        */
        cout<<"Enter a length in metres using float : ";
        cin>>l; //Read value of length entered by the user
        cout<<"Enter a width in metres using float : ";
        cin>>w; //Read value of length entered by the user
        area = l * w;
        cout<<"The area of the rectangle in float (before casting) is "<<area<<" square metres"<<endl;

        //use casting int(area) to get area in integer
        cout<<"The area of the rectangle (after casting) is "<<int(area)<<" square metres"<<endl;
        return 0;
}
```

FOR ONLINE READING ONLY

Output:

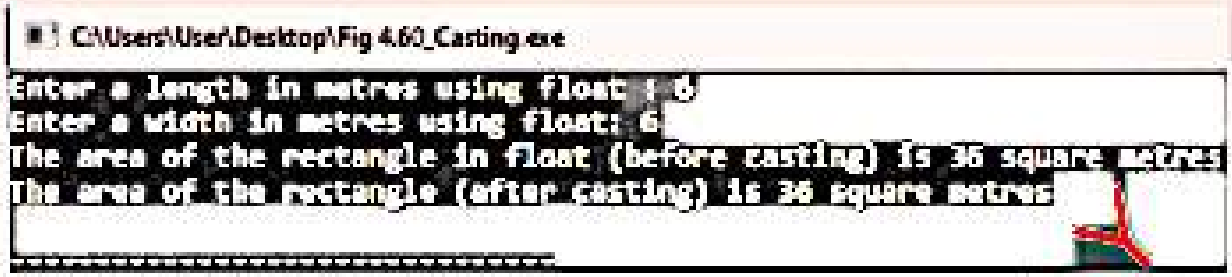


Figure 4.60: *The program demonstrating the use of casting in C++*

The following Program Example 4.32 will help you to learn how to write a program that changes, letter case using built-in function. Follow the commands properly and practice them. This will demonstrates a C++ program that changes letter case using built-in function. The output is shown in Figure 4.61.

Program Example 4.32:

A C++ program to change letter case using built-in function

```
#include<iostream>
#include<cctype>
using namespace std;
int main()
{
        char ch;
        cout<<"Enter a character: ";
        cin>>ch;
        cout<<"You have entered: "<<ch<<endl;
        cout<<endl;
        if(islower(ch))
{
        cout<<"It is lowercase letter "<<endl;
        cout<<"Its equivalent uppercase letter is "<<static_cast<char>(toupper(ch))<<endl;
                                }
        else if(isupper(ch))
                                {
        cout<<"It is an uppercase letter "<<endl;
        cout<<"Its equivalent lowercase letter is "<<static_cast<char>(tolower(ch))<<endl;
                                }
        if(isdigit(ch))
                                {
                                cout<<"It is digit character "<<endl;
                                }
return 0;
}
```

FOR ONLINE READING ONLY

Output:



Figure 4.61: *Output of the program*

Note: - A function"static_cast<char>( )" enable to translate the value within the function *tolower() and toupper( )* into character.

Activity 4.7:

Understanding the application of some of the built-in function

(a) Search on the internet built-in functions which are used to
   (i) Change a character into a number using the ASCII system
   (ii) Round the decimal number into the whole number
   (iii) Solve modulus division by returning remainders after the user divides two numbers.

(b) Write header files of each function

(c) Write a C++ program to show the application of each function.

Exercise 4.14

1. (a) What is the difference between passing parameters by value and by reference?

   (b) Write a C++ program to demonstrate how parameters are passed by value and by reference.

2. (a) Differentiate between a function prototype and function header

   (b) Write a C++ program to show the application of the function prototype.

3. Create a C++ program that will prompt a user to enter the range of the number and choose to display either an even, odd or a prime number. Then, the program will display selected types of number according to the range the user enters.

   Example:

   Enter a range of the number: 5

   Enter a type of the number: prime

   Output: 2, 3, 5, 7, 11.

FOR ONLINE READING ONLY

Note: Use "isPrime" built-in function to display prime numbers.

4. Write a C++ program that can help WAMO bank to calculate the total amount of money (A) gained by the customer after "n" number of years. Given, the interest rate "R" provided by the bank is 12%. The program will prompt the user to enter the amount of money (P) invested by the customer and then display the result on the computer screen. Hint: Use the formula, $A = P(1 + R)^n$, Use function pow (a, b) for ab.

Arrays and strings

Arrays

Suppose you are required to create a program that can help a teacher to read the marks of 50 students. To accomplish this task you will have to declare 50 variables each storing one mark of a student. Such a plan is entirely difficult because you may not remember and track the names of variables. To solve such problem, you can store all marks of 50 students using a single declaration of a variable. A typical data structure to help you achieve this is known as an *array*. Figure 4.62 shows a structure of array to solve a problem of creating numerous declarations.

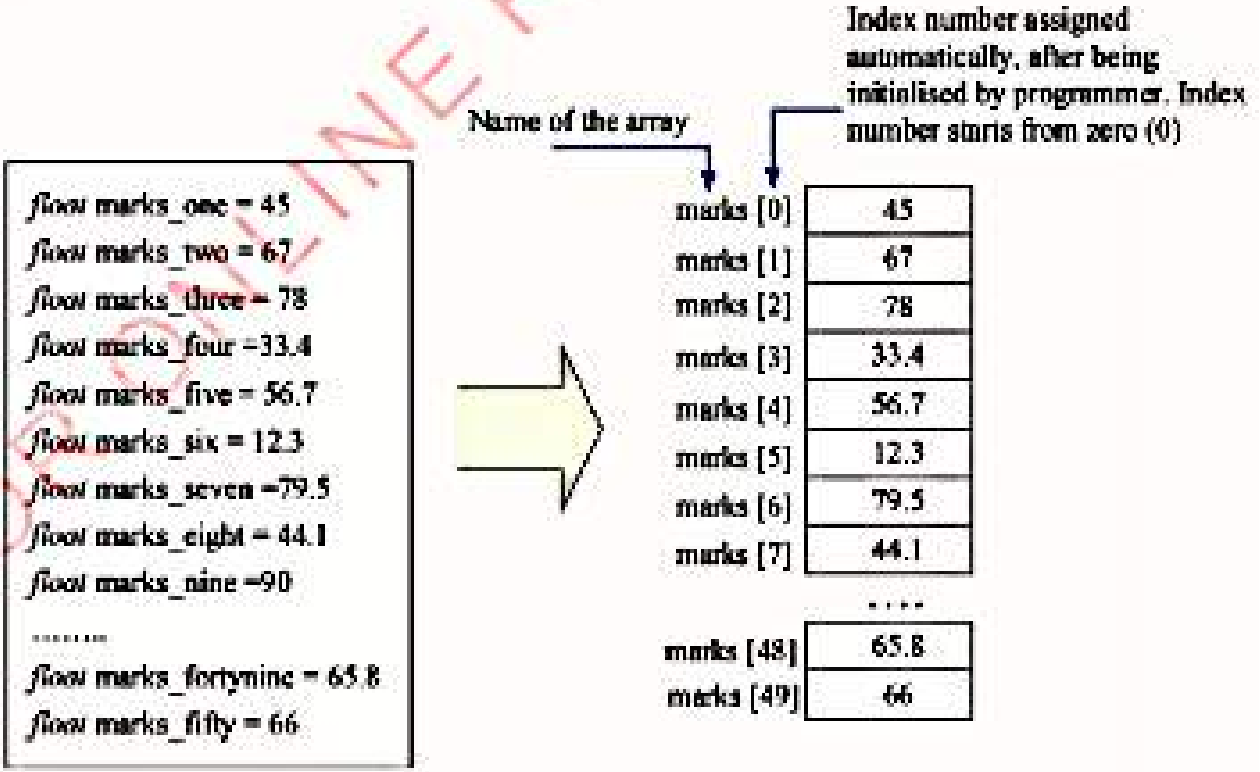


Figure 4.62: *A structure of array*

An array is a collection of elements of the same data types that can be stored in a fixed size, placed in a contiguous memory location, and that can be individually

FOR ONLINE READING ONLY

referenced. An array should be specified by its name and index. Instead of declaring each variable separately, an array is used. Each index must be within square brackets. Array can be classified into two types: One dimensional array and Multidimensional array.

One dimensional array

One dimensional array is the type of array that involve one array name and only one subscript.

Declaration of one-dimensional array

An array can be declared using the syntax:
dataType *arrayName[arraySize]*;
where:

(a) dataType represents the data type of all elements;

(b) arraySize must be an integer greater than zero; and

(c) arrayName must be a character or any word to represent an array.

Examples of declared array are:

int *a[5];*
double *sum[7];*

Initialisation of one dimensional array

Array initialisation means introducing values to the arrays. The following is the syntax of array initialisation:
dataType *arrayName[arraySize]* = {value0, value1, value3, ..., valuek}

Example: int *marks[5]* = {10, 35, 77, 98, 100};

The given example of initialisation can also be defined as:

```
int marks[5];
marks[0] = 10;
marks[1] = 35;
marks[2] = 77;
marks[3] = 98;
marks[4] = 100;
```

The following C++ Program Example 4.33 uses array to print the sum of five integers. The output of the program is shown in Figure 4.63.

Program Example 4.33:

Using array to print the sum of integers

```
#include<iostream>
#include<conio.h>
using namespace std;
int main()
{
        int a[5], sum = 0, i;
        cout<<"Enter 5 integers to store in array"<<endl;
        for(i=0; i<5; i++)
{
        cin>>a[i];
        sum = sum+a[i];
}
        cout<<"The sum of the given numbers is: "<<sum<<endl;
        return 0;
}
```

FOR ONLINE READING ONLY

Output:

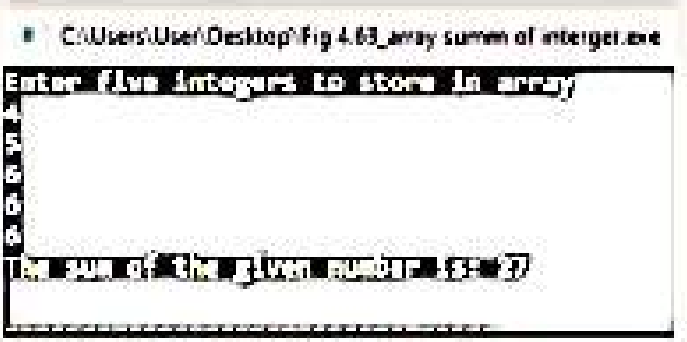


Figure 4.63: *Output of the program*

Multidimensional array

One-dimensional array represents linear data structure. In order to represent *n*-dimensional structure you are required to use multidimensional array. For example, we use two dimensional array to describe a matrix or table.

Declaration of two – dimensional array

The syntax to declare two-dimensional array is given as

dataType *arrayName[row size][column size];*

An example of a two-dimensional array is **int** *matrix[3][3];* The array contains three elements in a row and four elements in a column. Such an array can be tabulated as shown in Table 4.16:

Table 4.16: *Two-dimensional array*

matrix[0][0]	matrix[0][1]	matrix[0][2]
matrix[1][0]	matrix[1][1]	matrix[1][2]
matrix[2][0]	matrix[2][1]	matrix[2][2]

Initialisation of two-dimensional array

Initialise two-dimensional arrays by listing all elements horizontally or arranging members according to the feature of the array then enclosed them by curl brackets. Here is the syntax for initialising two-dimensional arrays,

dataType *arrayName[row size][column size]* = {list of values};

Example: int matrix[1][6] = {0,2, 7, 8, 9, 5}

or

```
int matrix[3][2]=
    {
      {0,2},
      {7,8},
      {9, 5},
    };
```

FOR ONLINE READING ONLY

The following C++ Program Example 4.34 demonstrates an example of a two-dimensional array. The output is shown in Figure 4.64.

Program Example 4.34:

Two-dimensional array

```
#include<iostream>
#include<iomanip>
using namespace std;
int main()
{
        int i, j, matrix[3][3];
        for(i=0; i<3; i++)
        for(j=0; j<3; j++)
        matrix[i][j]=i*3+j+1;
        cout<<"Display an array: "<<endl;
        for(i=0; i<3; i++)
{
        for(j=0; j<3; j++)

        cout<<setw(2)<<matrix[i][j];
        cout<<endl;
}
        return 0;
}
```

Output:

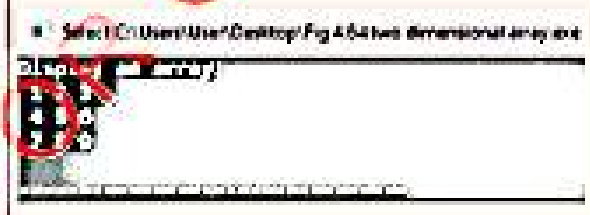


Figure 4.64: *Output of the program*

Note: A function setw(2) creates a gap of 2 size between two numbers as it is displayed in the output.

Activity 4.8:

Filling number in the table

Consider a box of 9×9 grid divided into 3×3 small boxes as shown in Table 4.17.

Table 4.17: *Sudoku tables*

6		9		4				1
7	1		5		9	6		
	5							
2		7		8			9	
				6			2	4
	6		9					8
		8				3		
			4					7
				5				

(a) Fill all blanks where each row and column should complete number 1 to 9 without repeating any number.

(b) Write a C++ program using array that prompts a user to fill the box. The program should prompt the user to enter the given number to each box and display the completely filled table.

Exercise 4.15

1. Write a C++ program using array that allows a teacher to enter the number of students and their marks. Then, the program should find average class marks and use

FOR ONLINE READING ONLY

a "function" to state a grade class mark. Apply Table 4.18 to display the grade class marks of students.

Table 4.18: *Grade pass mark*

AVERAGE CLASS MARK	GRADE
81 – 100	A
61 –80	B
41 – 60	C
21 – 40	D
0 – 20	F

2. Use a multi-dimensional array to create a C++ program that asks a student to enter the order and elements of the matrix. Then, the program should display a matrix and its transpose as represented in Figure 4.65.



Figure 4.65: *Output of the program*

3. Consider the declaration: *int majumuisho[10][20];*

   Write the C++ program that prints all the elements in majumuisho. All elements in the same row should appear on the same line, and each successive row should appear on its own line.

4. Consider the declaration *int mayah[50][100][100][25];*. How many elements does mayah hold?

String

A string can be defined as a collection of characters. It is a one-dimensional array made by characters. A string should be terminated by a null character represented by '\0'. Termination of the string is essential to make the function know the end of the string.

Declaration of a string

A string can be declared as *char stringName[stringSize] ;*

Initialisation of the string

Initialisation of the a string is done by introducing the string value to the string. An example of string initialisation is:

char name[ ] = {'J','U','M','A','\0'};.

String standard function

C++ programming language provides different string standard library handling functions. Some of the string library functions are represented in Table 4.19.

FOR ONLINE READING ONLY

Table 4.19: *String library function*

Library function	Application
strlen( )	Find length of the string
strcpy( )	Copy one string to another
strcat( )	Combine two strings
strcmpi( )	Compare characters of two string

String operation

Reading and writing in a C++ program are controlled by *'cin'* and *'cout'*. While *'cin'* reads a string in a program, *'cout'* is responsible for writting that string to the console window. Program Example 4.35 of a C++ displays name and age. The output is shown in Figure 4.66.

Program Example 4.35:

Using character to demonstrate string function

```
#include <iostream>
using namespace std;
int main()
{
        char name[10];
        int age;
        cout<<"Enter your first name and age:"<<endl;
        cin>> name>>age;
        cout<<"You entered: "<<name<<" "<<age<<endl;
}
```

Output:

```
C:\Users\User\Desktop\Fig 4.66_String function.exe
Enter your first name and age:
Kwanamalundi
19
You entered: Kwanamalundi 19
```

Figure 4.66: *Output of the program*

FOR ONLINE READING ONLY

The following are some examples that use the string operations *strcpy( ), strcat( ), strlen( ), strcmp( ), strchr( ), and strstr( ).*

strcpy( )

The following Program Example 4.36 shows the use of a string function for copy.

Program Example 4.36:

Using Strcpy( ) string function

Taking the string s1 and s2 in strcpy(s1,s2). This string function of C++ will copy the string s2 into string s1.

```
#include <iostream>
#include <cstring>
using namespace std;
int main()
{
char string1[]="Hujambo";
char string2[]="Mwananzengo";
strcpy(string1, string2);
cout << "The result of strcpy is : " <<string1 << '\n';
}
```

Output: The value of string2 (Mwananzengo) will be copied to string1 (Hujambo) and string2 value will display (Mwananzengo) as shown in Figure 4.67.



Figure 4.67: *Output of string2*

strcat( )

The following Program Example 4.37 shows the use of the string function for concatenation.

Program Example 4.37:

Using strcat( ) string function

Taking the string s1 and s2 in strcat(s1,s2). This string function of C++ will concatenate the string s2 at the end of string s1.

FOR ONLINE READING ONLY

```
#include <iostream>
 #include <cstring>
 using namespace std;
 int main()
 {
 char string1[]="Hujambo";
 char string2[]="Mwananzengo";
 strcat(string1, string2);
 cout << "The result of strcat is : " <<string1 << '\n';
 }
```

Output: The value of string2 (Mwananzengo) will be concatenated to string1 (Hujambo) and string2 value will be printed as shown in Figure 4.66.



Figure 4.68: *Output of string concatenation*

strlen( )

The following Program Example 4.38 shows the use of the string function for calculating length of a string.

Program Example 4.38:

Using strlen( ) string function

Taking the string s1 in strlen(s1), this string function of C++ will calculate and return the length of string s1.

```
#include <iostream>
 #include <cstring>
 using namespace std;
 int main()
 {
 char string1[]="Mzalendo madhubuti wa nchi yake";
 cout << "The result of strlen is : " <<strlen(string1)<< '\n';
 }
```

FOR ONLINE READING ONLY

Output: The length of string1 will be as shown in Figure 4.69.

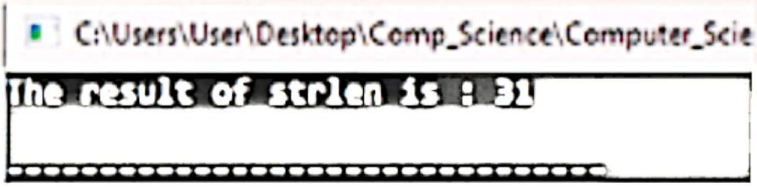


Figure 4.69: *Output of string length function*

strcmp( )

The following Program Example 4.39 shows the use of the string function that compares the two string values and returns 0 if the two strings are the same; returns -1 if the value of string1<string2, and 1 when string1>string2.

This function is used in a scenario where you are required to enter your name and password to login to a particular website. In such cases, at the administrator side , it is needed to build and script functions that will check and compare the input string (login details) with the string stored in the data base of the site.

Program Example 4.39:

Using strcmp( ) string function

Taking the string s1 and s2 in strcmp(s1, s2), this string function of C++ will compare the string s1 and s2 and return 0, -1 or 1

```
#include <iostream>
#include <cstring>
using namespace std;
int main()
{
char string1[]="Hujambo";
char string2[]="Mwananzengo";
cout << "The result of strcmp is : " <<strcmp(string1, string2) <<
"\n";
}
```

Output: The result will be -1 as shown in Figure 4.70. Find the result when you put *char string1[]= "Mwananzengo"* and *char string2[]* = "Hujambo", and when you put *char string1[]*= "Mwananzengo" and *char string2[]= "Mwananzengo"*.



Figure 4.70: *Output of string compare function*

FOR ONLINE READING ONLY

Activity 4.9:

Applying string library function in C++ program

Write a C++ program to show application of string library functions. In your program include the following functions:

(a) strlen( )

(b)strcopy( )

(c) strcat( )

(d) strcmp( )

(e) strchr( )

(f) strstr( ).

Exercise 4.16

1. Differentiate between string and array.
2. Why do we use the null character '\0' in string initialisation?
3. Write a program that displays the statement "C++ programming language".
4. Can we declare an array to hold the mixture of both and doubles and ints? Explain.
5. Explain what will happen if you attempt to access an element of an array using a negative index?

Implementation of key OOP features in C++

Classes in C++

In C++, classes serve as blueprints for constructing objects. They serve as a blueprint by bundling together data and methods that manipulate that data. Essentially, a class defines a specific type and outlines the structure and behavior of objects that belong to that type. Visualise it as a template that specifies the characteristics and capabilities an object will possess.

To illustrate, consider a real-life example: a car represents an object. This car possesses attributes like weight and color, as well as methods such as drive and brake. Attributes and methods can be compared to variables and functions that are associated with a class. They are commonly known as "class members".

A class, as a custom data type can be utilised in a program. It acts as a constructor for objects or a "blueprint" that outlines how to create objects. The syntax for class in C++ is like this:

```
class ClassName {
public:
    // public members
private:
    // private members
protected:
    // protected members
};
```

Program Example 4.40 shows a sample code for structure of a class

FOR ONLINE READING ONLY

Creating a class in C++

Program Example 4.40:

Syntax and structure for class

```
class Car {     // This is class
public:         // This is access specifier
    void drive() {
        // The code to drive the car will be put here
    }
private:
    int speed;
    int amountOfCoolant;
};
```

The keyword "class" is used to create a class named Car. The keyword "public" serves as an access specifier, indicating that the attributes and methods of the class can be accessed from outside the class. Private members declared under the "private" specifier can only be accessed and modified from within the class itself. They are not accessible or modifiable from outside the class, except for certain exceptions like friend classes or functions that have been given special access.

There are two variables in class "Car", speed and amountOfCoolant, both of type integer. These variables are referred to as attributes when declared within a class. Finally, the class definition is concluded with a semicolon (;).

Object in C++

An object is an instance of a class. When you create an object, you are creating the actual instance based on your class blueprint. For , example the piece of program code shows method to create object:

```
Car yourObject; // This code is creating an object of the Car class
yourObject.drive(); // This code is calling a method on the object
```

Creating object in C++

To create an object in C++, you need to use a class. To create an object of the Mmea class in Program Example 4.41, you have to specify the class name followed by the

FOR ONLINE READING ONLY

object name. To access the attributes of the class (miche and amountOfWaterBucket), you can use the dot syntax (.) on the object as shown in an example.

Program Example 4.41:

Creating object in C++

```
#include <iostream>
#include <string>
using namespace std;
class Mmea {       // The class
  public:          // Access specifier
     int miche;       // Attribute (int variable)
     int amountOfWaterBucket;  // Attribute (int variable)
};
int main() {
    Mmea yourObj;  // Create an object of class Mmea
    // These codes will access attributes and set values
    yourObj.miche = 180;
    yourObj.amountOfWaterBucket =  5;
   // These codes will print attribute values
    cout << yourObj.miche  << "\n";
    cout << yourObj.amountOfWaterBucket ;
  return 0;
}
```

Exercise 4.17

1. Create a Book class that have attributes called author, title, and ISBN. Show methods that will display book details.
2. Create a Car class with attributes called make, model, year, and speed. Include methods such as accelerate and brake.
3. Judge the following codes:

```
class Animal {
   int y;
};
int main()
{
  Animal h;
  cout << h.y;
  return 0;
}
```

FOR ONLINE READING ONLY

Inheritance in C++

The following Program Example 4.42 demonstrate how inheritance is implemented.

Program Example 4.42:

Program code demonstrating inheritance

```
#include <iostream>
using namespace std;
// Parent class
class Animal {
public:
    void eat() {
        cout << "The animal of this class will eat: mnyaa! mnyaa!mnyaa!" << endl;
    }
    void sleep() {
        cout << "The animal of this class will sleep" << endl;
    }
};
// Child class
class Dog : public Animal {
public:
    void bark() {
        cout << "The Dog will bark: wo!wo!wo!" << endl;
    }
};
int main() {
    // This will create an object of the child class
    Dog thisDog;
    // This will call methods from the parent class
    thisDog.eat();
    thisDog.sleep();
    // This will call method from the child class
    thisDog.bark();
    return 0;
}
```

Output:

```
The animal of this class will eat: mnyaa! mnyaa!mnyaa!
The animal of this class will sleep
The Dog will bark: wo!wo!wo!
```

Figure 4.71: *Output of inheritance program*

FOR ONLINE READING ONLY

In the code above, the parent class is called Animal and it has two methods: eat() and sleep().

The child class, Dog, is created based on the Animal class. It inherits the methods from Animal and adds a new method called bark(). In the main() function, an object of type Dog is created. This object can access and call methods from both the parent class (Animal) and the child class (Dog).

Activity 4.10:

Developing program to deposit and withdraw money

Assume you have been hired by a bank to develop a program for customers to deposit and withdraw money. The program should check if the requested withdrawal amount exceeds the current balance before processing. It should also perform additional functions

Polymorphism in C++

This example shows how to implement polymorphism.

Program Example: 4.43:

Program code demonstrating polymorphism

```
#include <iostream>
using namespace std;
// Parent class
class Shape {
public:
    // This code shows Virtual function
    virtual void draw() {
        cout << "The drawn shape is expected here" << endl;
    }
};
// Child class Circle
class Circle : public Shape {
public:
    // This code will override the draw() method
    void draw() override {
        cout << "The drawn Circle is expected here" << endl;
    }
```

FOR ONLINE READING ONLY

```
};
// This code shows a child class Rectangle
class Rectangle : public Shape {
public:
    // Overriding the draw() method
    void draw() override {
        cout << " The drawn rectangle is expected here " << endl;
    }
};
// This function will draw any shape
void drawShape(Shape* shape) {
    shape->draw();
}
int main() {
    //This will create objects of various shapes
    Circle circle;
    Rectangle rectangle;
    // This shows behaviors for Polymorphism
    drawShape(&circle);
    drawShape(&rectangle);
    return 0;
}
```

Output:

```
The drawn Circle is expected here
The drawn rectangle is expected here
```

Figure 4.72: *Output of polymorphism program*

In the code above, there is a parent class called Shape, which has a virtual function named draw().

Two child classes, Circle and Rectangle, are created. These classes override the draw() method inherited from the Shape class. In the main() function, the concept of polymorphism is demonstrated. Different objects of shapes are passed to the drawShape() function, which accepts a pointer to the base class Shape.

Polymorphism is achieved through the use of virtual functions. The draw() function in the parent class Shape is declared as virtual, allowing child classes to override it. The function drawShape() takes a pointer to a Shape and calls draw() on it, demonstrating runtime polymorphism.

FOR ONLINE READING ONLY

Activity 4.11:

Developing program using polymorphism to calculate areas for various shapes

Develop a program using polymorphism to calculate the areas of a rectangle, square, trapezium, and rhombus.

Encapsulation in C++

Program Example 4.44:

Program code demonstrating encapsulation

```
#include <iostream>
using namespace std;
// Class definition
class BankAccount {
private: // This shows data members are private, therefore, encapsulated
   string accountHolderName;
   double balance;
public: // This shows member functions are public
   // Constructor
   BankAccount (string name, double initialAmount) {
      accountHolderName = name;
      balance = initialAmount;
   }
   // This method will deposit money
   void deposit (double amount) {
      balance += amount;
   }
   // This method will withdraw money
   void withdraw (double amount) {
      if (amount <= balance) {
         balance -= amount;
      } else {
         cout << "No enough cash!" << endl;
      }
   }
   // This method will display account information
```

FOR ONLINE READING ONLY

```
    void display () {
        cout << "accountHolder: " << accountHolderName << ", Balance: TZS"
<< balance << endl;
    }
};
int main () {
    // Create an object of BankAccount class
    BankAccount account ("Mkulu Mkuvangwa", 100000);
    // Display initial account information
    cout << "Initial account information:" << endl;
    account.display ();
    // Deposit and withdraw some money
    account.deposit (20000);
    account.withdraw (15000);
    // Display updated account information
    cout << "\nUpdated account information:" << endl;
    account.display ();
    return 0;
}
```

Output:

```
Initial account information:
accountHolder: Mkulu Mkuvangwa, Balance: TZS100000

Updated account information:
accountHolder: Mkulu Mkuvangwa, Balance: TZS105000
```

Figure 4.73: *Output of encapsulation program*

This code example demonstrates the concept of encapsulation, which is one of the pillars of object-oriented programming (OOP). The code defines a class called BankAccount, which encapsulates data members (accountHolder and balance) and member functions (deposit, withdraw, and display). The data members are declared as private, meaning they can only be accessed and modified within the class itself. The member functions are declared as public, allowing them to be accessed and called from outside the class. The constructor is used to initialize the object with an owner name and initial balance. The deposit and withdraw methods modify the balance based on the given amount, ensuring that the balance is not negative. The display method is used to print the owner's name and balance.

FOR ONLINE READING ONLY

Activity 4.12:

Developing program using encapsulation to manage student's examination

Imagine you are an examination officer at Mpemba High school. Create a program to manage students' examinations, including their names, admission numbers, and scores for some subjects. The program should calculate total and average scores of students.

Abstraction in C++

Abstraction is the process of hiding the complex implementation details and showing only important features of an object. It focuses on what an object does rather than how it does it. The way abstraction relates with encapsulation is when encapsulation is used to achieve abstraction. In encapsulation, this is done by hiding the internal details of an object leaving only essential features.

Public and Private Members in Abstraction

Public Members

Members declared as public are accessible from outside the class, forming part of the class interface. These members, sometimes referred to as access specifiers, define the accessibility of class elements.

For example, following part of code shows public member:

```
class Library {
public:
   void addBook() {
      // The code to add books of class Library will be written here
   }
};
```

Private members

Members declared as private cannot be accessed directly from outside the class. They can only be accessed by the class's own methods. This helps in hiding the implementation details.

For example, following part of code shows private member:

```
class Car {
 private:
   int mileage;
```

FOR ONLINE READING ONLY

```
    float fuelConsumption;
public:
    void setMileage(int m) {
        mileage = m;
            }
};
```

In the context of this example, abstraction can be achieved by using private members to hide certain details of the car's internal workings. The private attributes, such as the engine type or fuel consumption, are not directly accessible or modifiable from outside the class. Instead, they can only be accessed and modified through public (set and get) methods or functions provided by the car class.

Implementing abstraction in C++

Program Example 4.45:

Code program to demonstrate abstraction in C++

```
#include <iostream>
using namespace std;
class demoAbstraction {
private:
        int a, b;
public:
        //This method will set values of private members
        void set(int x, int y)
        {
                a = x;
                b = y;
        }
        void display()
        {
                cout << "a = " << a << endl;
                cout << "b = " << b << endl;
        }
};
int main()
{
        demoAbstraction obj;
        obj.set(103, 230);
        obj.display();
        return 0;
}
```

FOR ONLINE READING ONLY

Output:

```
a = 100
b = 200
```

Figure 4.74: *Output of abstraction program*

In this program example, the variables a and b cannot be accessed directly, however, you can call the function set() to set the values in a and b and the function display() to display the values of a and b.

Activity 4.13:

Implementing a simple school management system

Aim:

To understand and implement the concept of abstraction in C++ by creating a simple school management system with different types of members.

Materials:

Computer installed with C++ compilers, or online IDEs.

Instructions:

Step 1: Implement base class:

(a) Create an abstract base class SchoolMember with common attributes like name, age, and id.

(b) Define pure virtual functions for actions like introduce() and getRole().

Step 2: Implement derived class:

(a) Implement a Teacher class that inherits from SchoolMember with specific attributes like subject.

(b) Implement a Student class that inherits from SchoolMember with specific attributes like grade.

(c) Ensure both derived classes provide their own implementation of the introduce() and getRole() methods.

Step 3: Implement main function

(a) Instantiate objects of Teacher and Student.

(b) Call their respective methods to demonstrate polymorphic behavior.

(c) Print out the introductions and roles of the school members.

Exercise 4.18

1. Create a C++ program that simulates a bank account system. Define a base class BankAccount with private attributes like accountNumber, balance, and ownerName. Provide public methods to deposit and withdraw money, and to display account details. Then, derive a

FOR ONLINE READING ONLY

class SavingsAccount that adds an attribute for interest rate and a method to apply interest. Explain how abstraction helps in managing the complexity of the system.

2. Design a C++ program for a library management system. Create a base class LibraryItem with attributes like title, author, and ISBN. Include methods to get the details of the item. Then, derive classes Book and Magazine from LibraryItem, adding specific attributes and methods for each. How does abstraction make the system more manageable and user-friendly?
3. In a banking system, how would you use inheritance to model different types of accounts such as SavingsAccount, CheckingAccount, and LoanAccount, all inheriting from a base class BankAccount? Discuss how you would structure the classes to maximize code reusability and minimize redundancy.
4. Consider a class representing a bank account. How would you use encapsulation to ensure that the account balance can only be modified through specified methods, such as deposit() and withdraw(), while preventing direct access to the balance variable from outside the class?
5. How would you use polymorphism to implement a system for processing different types of shapes, such as circles, rectangles, and triangles, where each shape has a calculateArea() method? Discuss how you would design the classes and utilize polymorphism to calculate the area of each shape without explicitly checking the type of the object.
6. This code demonstrates how inheritance, encapsulation, polymorphism, and abstraction can be used together to create flexible and reusable code in C++. Study the code carefully and extract block of codes that represents inheritance, encapsulation, polymorphism, and abstraction. Evaluate each block of code in each pillar.

```
#include <iostream>
using namespace std;

// Base class
class Shape {
public:
    // Virtual function
    virtual void draw() {
        cout << "Drawing a shape"
        << endl;
    }
};
// Derived class Circle
class Circle : public Shape {
public:
    // Overriding the draw() method
    void draw() override {
        cout << "Drawing a circle"
        << endl;
    }
```

FOR ONLINE READING ONLY

```
};
// Derived class Rectangle
class Rectangle : public Shape {
public:
  // Overriding the draw() method
  void draw() override {
    cout << "Drawing a rectangle"
    << endl;
  }
};
// Function to draw any shape
void drawShape(Shape* shape) {
  shape->draw();
}
int main() {
  // Creating objects of different
    shapes
  Circle circle;
  Rectangle rectangle;
  // Polymorphic behavior
  drawShape(&circle);
  drawShape(&rectangle);
  return 0;
}
```

Chapter Summary

Reflect on what you have learnt from this chapter, summarise and keep it in your portfolio.

Revision Exercise 4

1. Read the following C++ program and answer the question that follow:

```
#include<iostream>
using namespace std;
int main()
{
int num, temp, sum;
cout<< "Enter a positive integer: ";
cin>>num;
cout<<endl;
temp=num;
sum=0;
do
{
sum=sum+num%10;
num=num/10;
}
while(num>0)
cout<< "sum is: "<<sum<<endl;
system("PAUSE");
return 0;
}
```

   (a) What are the functions of the line *#include<iostream>and system("PAUSE");*

   (b) Write an error made from this program.

2. Why do we need to state data type when declaring variables in a C++ program?
3. How will you guarantee and ensure that a variable's value can never be changed after its initialisation?

FOR ONLINE READING ONLY

4. If a = 7, y = 4, z = 3, and u= 2, evaluate each of the following statements, if possible. If it is not possible, state the reason.

   (a) (a + z) % y (b) (a + y % u

   (c) (y + u) % a (d) (a + y) *u

   (e) (a % y) % z (f) (y% z) % a

   (g) (a *z) % y (h) ((a *y) *u) *z

5. Evaluate the following expressions.

   (a) 15 / 3 (b) 17 - 12 / 4 * 2

   (c) 34 % 7 (d) 3 - 7% 7

   (e) 18.0 / 4 (f) 20 - 5 / 2.0

   (g) 13 + 5 % 2 - 3

   (h) 1570 + 3.0 * 2.0 / 5.0

6. Write the syntax of the following C++ statements:

   (a) Switch case statement

   (b) Variable declaration and initialisation.

7. Suppose you are developing a game where there are various types of vehicles, such as cars, trucks, and motorcycles. How would you use polymorphism to implement a drive() method for each vehicle type, ensuring that the correct behavior is executed based on the specific type of vehicle object? Provide a code example demonstrating the use of polymorphism in this scenario.

8. Write a C++ program to find all roots of a quadratic equations. The program should prompt a user to enter the coefficient and then, display roots. The program should also be able to test the type of the root displayed and state whether those roots are real distinct, identical or imaginary.

   Note: For any quadratic equation $ax^2 + bx + c = 0$, if

   $b^2 > 4ac$, the root is real distinct

   $b^2 = 4ac$, the root is real identical

   $b^2 < 4ac$, the root is imaginary

9. Use a function named "largest" to create a C++ program that determines the largest number from a set of 10 numbers.

10. By using the concept of function, write a C++ program that prompts the user to input three integer values and find the greatest value of the three values.

11. Write a C++ program that finds the reverse of number entered by the user. When the number entered is '5261' then the reverse value should be '1625'.

12. Given the declaration orodha[100];

    (a) What expression represents the very first element of orodha?

    (b) What expression represents the very last element of orodha?

    (c) Write the code fragment that prints out the contents of orodha?

    (d) Is the expression list[3.0] legal or illegal?

FOR ONLINE READING ONLY

13. The NMB ATM allows a customer to withdraw a maximum of Tshs 1,000,000/= per day. If a customer withdraws more than Tshs 335,000/= the service charge is 6% of the amount over Tshs 335,000/=. If the customer does not have sufficient money in the account, the ATM informs the customer about the insufficient fund and gives the option to withdraw the money for a service charge of Tshs 1250/=. If there is no balance in the account or if the account balance is negative, the customer will not be allowed withdraw any money from the ATM. If the amount to be withdrawn exceeds Tshs 1,000,000/= , the ATM informs the customer about the maximum amount that can be withdrawn. Write an algorithm that allows the customer to enter the amount to be withdrawn. The algorithm should check the total amount in the account, dispense the money to the customer, and debit the customer account by the amount withdrawn and the service charges, if any.

14. Explain the concept of polymorphism and its different types, such as compile-time polymorphism (function overloading) and runtime polymorphism (virtual functions), in C++. Provide examples illustrating each type of polymorphism and discuss their advantages and use cases.

15. Write a C++ program to implement a class called Shape with virtual member functions for calculating area and perimeter. Derive classes such as Circle, Rectangle, and Triangle from the Shape class and override virtual functions accordingly.

16. Write a C++ program to display the following patterns:



Figure 4.75: (a)

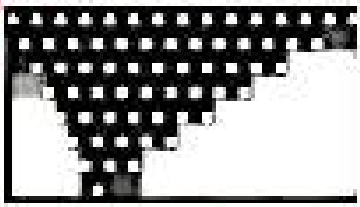


Figure 4.75: (b)

17. Create a calendar using a C++ programming language. A program will prompt a user to enter a year followed by a month in a number. Then, the calendar will be displayed as shown in Figure 4.76.

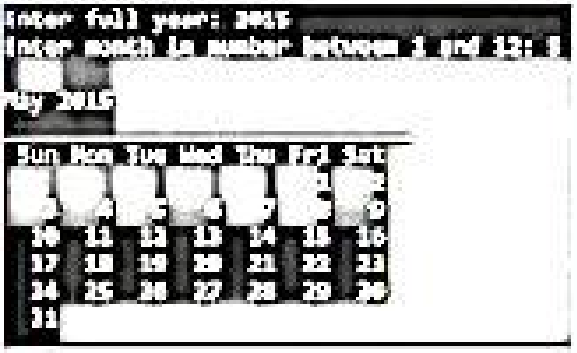


Figure 4.76:

Hints:

(a) Use *Switch case* statement to change the month entered by the user in a

FOR ONLINE READING ONLY

number form to the name as it is displayed from the given snapshot.

(b) Use *function* to read the year and month entered by the user and translate them into days.

(c) Use *if...then* statement to help a system to know the number of days for each month, where:-

(i) January, March, May, July, August, October and December has 31 days

(ii) April, June, September and November has 30 days.

(iii) February has 29 days for leap year and 28 days for an ordinary year.

(d) Use *isLeapYear()* built-in function to help a system identify the leap year(with 366 days) made by 29 February.

(e) Make your program displays months' calendar for all years dated after 1800.

(i) Example of the snapshot from the leap year, 2012 is shown in Figure 4.77.



Figure 4.77:

(ii) Example of the snapshot from the ordinary year, 1991 as shown in Figure 4.69.

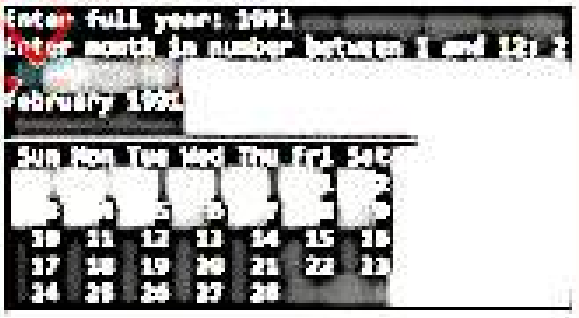


Figure 4.78:

FOR ONLINE READING ONLY

Project: *Designing and implementing a simple library management system*

Design and implement a simple library management system by creating a console-based application that manages a library's collection of books. The system should allow users to add, delete, search, and display books. The system should also demonstrate the use of classes, inheritance, polymorphism, encapsulation, and data abstraction. In your project you may use resources like Computer with C++ development environment (Such as Code::Blocks, Visual Studio, or any IDE that supports C++). Also, you may use Text editor for writing code (if not using an IDE). After you accomplish, submit the functional library management system with detailed documentation.



Chapter Five

Object Oriented Programming with Java

Introduction

Java is a versatile and widely adopted programming language used for developing various software applications. It is a powerful, Object-Oriented programming (OOP) language known for its portability, making it ideal for creating desktop, mobile, and web applications. In this chapter, you will explore the concepts of Java Programming Language. You will learn the syntax and constructs needed to create object-oriented programs. Additionally, you will gain insights into debugging object-oriented programs using Java, enabling you to tackle challenges and transform them into OOP solutions. The competencies developed will enable you to solve problems and collaborate effectively within coding teams. Furthermore, you will be able to perform code maintenance and debug software systems using Java.

FOR ONLINE READING ONLY

Think

Possibilities that Java can unlock in this world

Concept of Object Oriented Programming with Java

Scenario 5.1: *"Java kinywaji" coffee shop*

Imagine you are own a coffee shop called "Java Kinywaji" and want to improve operations. You develop a Java-based system to manage inventory, handle orders, and provide real-time information. This system allows you to track inventory, customise orders, process payments, and display promotions. As a result, your coffee shop becomes efficient, delivering exceptional service that keeps customers coming back.



Questions:

(a) How does the Java-based system help you track inventory and improve efficiency? What features enable effective inventory management?

(b) How does the Java system enhance the customer experience at "Java kinywaji"? How does it streamline ordering and provide real-time information to customers?

Overview of Java programming language

Java is a general-purpose, high-level programming language, was created by James Gosling at Sun Microsystems (now Oracle) in the mid-1990s. It is designed to be platform-independent, where it can run anywhere through bytecode compilation and the Java Virtual Machine (JVM). Java is known for its simplicity, portability, and extensive ecosystem of libraries and frameworks.

Features

(a) Object Oriented Programming (OOP): Java is centered around OOP principles, supporting classes, objects, inheritance, polymorphism, and encapsulation.

(b) Platform independence: Java code is compiled to bytecode, which can be executed on any device with a JVM, allowing for high portability.

(c) Rich standard library: Java offers a comprehensive standard library, providing utilities for networking, I/O, collections, and GUI programming.

(d) Memory management: Java features automatic memory management through garbage collection, reducing the risk of memory leaks.

Applications

(a) *Enterprise software*: Java is widely used for developing web applications, server-side applications, desktop applications, games and middleware.

(b) *Mobile development:* Java is the primary language for Android app development using the Android SDK.

(c) *Big data and cloud computing:* Java is utilized in technologies such as Hadoop, Spark, and various cloud-based services.

The following are some of the reasons to use Java programming language:

(a) *Platform independence:* refers to the ability of Java code to "write once, run anywhere" on any platform with the Java Virtual Machine (JVM) installed. This eliminates the necessity of developing code again for various operating systems. Java

FOR ONLINE READING ONLY

works on Windows, Mac, Linux, Raspberry Pi, etc.

(b) *Java programs are object-oriented:* they are constructed around objects that have functions (methods) and data (attributes). This encourages maintainability and reusability of the code.

(c) *Robust and secure:* Java is a popular choice for enterprise applications since it comes with built-in memory management and security capabilities.

(d) *Large community:* Java has a vast and active developer community, that provides extensive resources and support for learning and development.

(e) Java is an object-oriented language that reduces development costs by providing programs with a clear structure and enabling code reuse.

(f) Programmers can easily transition from C++ and C# to Java because of its closeness to these languages.

Objects and classes in Java

Object

State and behavior are attributes of objects. A good example would be a dog's color, breed, name and behaviors, such as eating, barking, and tail-wagging. Therefore, an object is an instance of a class.

Class

Behavior or state of the object can be described by a class, which is a kind of template or blueprint.

Objects in Java

In the actual world, there are a various items all around us, including humans, dogs, cars, and other objects. Each of these items has a behavior and a state. A dog has the following state: name, breed, color, and behavior: running, barking, and tail-wagging. The software object and a real-world item share many similarities in their features. Additionally, software objects possess a state and a behavior. Fields are used to store a software object's state, and methods are used to display its behavior. Thus, in software development, methods work with an object's internal state and facilitate communication between objects.

Classes in Java

A class is a blueprint from which individual objects are created. The following is an example of a class.

```
public class Dog {
 String breed;
        int age;
 String color;
        void barking(){}
        }
}
```

A class can contain any of the following variables:

(a) *Local variables:* These are variables which are defined inside methods, constructors or blocks. The variable

FOR ONLINE READING ONLY

is declared and initialised within the method and destroyed when the method has completed.

(b) *Instance variables:* These are variables within a class but is outside any method. They are initialised when the class is instantiated. They can be accessed from inside any method, constructor or blocks of that specific class.

(c) *Class variables:* These variables have the static keyword applied and are declared inside a class, independent of any methods.

A class can have any number of methods to access the value of various kinds of methods. In the above example, methods are such as barking(), hungry() and sleeping().

Class and Instance in Java

A class is a group of items of the same kind. As shown in Table 5.1, a class is a three-compartment box that contains data and operations.

Table 5.1: *Class structure*

Class Name
Static Attributes
Dynamic Behaviors

Table 5.2 shows an example of two classes "Car" and "Fruit"

Table 5.2: *Example of class*

Name (Identifier)	**Variables (Static attributes)**	**Methods (Dynamic behaviors)**
Car	Make, model, color	startEngine(), accelerate(), brake()
Fruit	Name, color, taste	Ripen(),rote(),isSweet

As shown from the table, a class can be viewed in three dimensions:

1. *Name or identity:* This is used to identify the class.
2. *Variables (sometimes called attribute, state, or field):* these represent the static attributes of the class.
3. *Methods:* These contain the class's dynamic behaviors, functions, and operations.

An instance, is an object built from a class (blueprint) which has all the characteristics defined in the class, but with specific values. For example, we can define a class called "Car" and create three instances of the class, "Car" for "Nissan", "Toyota" and "Honda".

Table 5.3 shows three instances of the class Car, identified as "Nissan", "Toyota" and "Honda".

FOR ONLINE READING ONLY

Table 5.3: *Instance of a class "Car"*

Nissan	Toyota	Honda
Make= "Nissan" Color= "Silver"	Make= "Toyota" Color= "Black"	Make= "Honda" Color= "Purple"
StartEngine() Accelerate()	StartEngine() Accelerate()	StartEngine() Accelerate()

Exercise 5.1

1. Develop a class named Car with properties make, model, and year. Create three objects of this class and print their details.
2. Implement a class 'Rectangle' with properties 'length' and 'width'. Add methods to calculate the area and perimeter of the rectangle. Create an object of the class and display the area and perimeter.
3. Write a program of a class 'Student' with instance variables 'name' and 'age', and a class variable 'schoolName'. Create three instances of the 'Student' class and assign different values to the instance variables but the same value to the class variable. Print the details of each student.
4. Create a class 'BankAccount' with instance variables 'accountNumber' and 'balance'. Add methods to deposit and withdraw money, and check the balance. Instantiate an object of the class and perform some transactions.

Key principles of OOP in Java

Abstraction

The term abstraction means dealing with ideas rather than events. It essentially deals with keeping information hidden and only presenting the user with what is necessary.

Activity 5.1:

Exploring of abstraction in the surrounding environments

Aim:

To identify and describe examples of abstraction in various environment.

Materials: TV, Surrounding environments (such as classroom, play ground, home and library) or Internet

Instructions:

Everyone knows how to turn on the TV, but understanding how it operates is not necessary. In abstraction, simple entities such as variables, classes, and objects can represent more complex underlying code and data. It prevents doing the same task more than once.

FOR ONLINE READING ONLY

Use the relevant material or resources, including library search, explore other issues or things that are examples of abstraction.

Deliverable:
Summary report that includes sketches and photographs.

Document your findings in a portifolio describing in details how they are regarded as abstraction.

Java allows abstraction through the use of abstract classes and interfaces.

(a) Abstract class

In Java, the "abstract" keyword is a component of the abstract class. A class cannot be instantiated if it is declared abstract. Therefore, you cannot create an abstract class object. Furthermore, both abstract and concrete methods may be present in an abstract class.

In order to utilise an abstract class, you must inherit it from another class and supply implementations for the abstract methods. Otherwise, the original class will likewise turn into an abstract class.

(b) Interface

Interface in Java is a group of abstract methods and static constants. Every method in an interface is public, abstract, and does not contain a constructor. In Java, interfaces aid in achieving multiple inheritance in addition to abstraction. So an interface is a collection of related methods with empty bodies.

Encapsulation

Encapsulation is the binding, or wrapping, of data and code into a single unit. In Java, encapsulation is similar to keeping important things in a locked box. It hides inside details and stops others from accessing them without permission, making sure the data stays safe and unchanged.

Activity 5.2:

Exploring encapsulation in the surrounding environments

Aim:
To understand encapsulation in OOP by exploring real-world examples where data is protected and secure.

Materials: Medical capsule, Surrounding environments or Internet

Instructions:
Hiding your data so that it is protected from modifications is same as encapsulation. In a medical capsule, the medication is always secure inside the capsule, is the best way to comprehend encapsulation. Use the relevant material or resources, including library search, explore other issues or things that are examples of encapsulation.

Deliverable:
Summary report that includes sketches, diagrams or photographs.

Document your findings in a portifolio describing in details how they are regarded as encapsulation.

FOR ONLINE TEACHING ONLY

Inheritance

This is a unique feature of Object Oriented Programming in Java. It enables programmers to construct new classes with some of the attributes of pre-existing classes shared.

Activity 5.3:

Exploring of inheritance in the surrounding environments

Aim:
To understand the concept of inheritance in OOP with Java by exploring real-world examples where properties are inherited from a parent entity to a child entity

Materials: TV, Toy animals, Family tree chart, surrounding environments or Internet

Instructions:
In human families, a child inherits the properties from his father. In Java perspective the two groups are known as parent class (Super or Base class) and Child class (derived class or subclass) A class that receives inheritance is referred to as a "Child class," while a class that whose properties are inherited is referred to as a "Parent class. Use the relevant material or resources, including library search, explore other issues or things that are examples of inheritance.

Deliverable:
Summary report that includes sketches and photographs.

Document your findings in a portifolio describing in details how they are regarded as abstraction.

Polymorphism

In Greek language, polymorphism is regarded as taking many forms. In Java, polymorphism is like different types of vehicles (cars, bicycles, trucks) which shares the same road. Although each vehicle behaves differently, but they all follow the same traffic rules. Similarly, objects of different classes can share common methods and be treated uniformly, despite their unique behaviors.

Activity 5.4:

Exploring of polymorphism in the surrounding environments

Aim:
To identify and understand the real world examples of polymorphism and its roles in allowing objects to exhibit different behaviours based on their contexts.

Material:
Vehicles, various shapes, various types of bank accounts, various media files, surrounding environments or Internet

Instructions:
In Java, a good example is observed in different banks in offering different interest rates. Use the relevant material or resources, including library search, explore other issues or things that are examples of polymorphism.

Deliverable:
Summary report that includes sketches and photographs.

Document your findings in a portifolio describing in details how they are regarded as polymorphism.

FOR ONLINE READING ONLY

Setting up the development environment (IDE and JDK Installation)

When you need to write Java programs, it is important to make sure Java Runtime Environment (JRE), Java Virtual Machine (JVM) and Java Development Kit (JDK) are already installed in your computer.

The Java Runtime Environment (JRE) is a collection of software tools used to develop Java applications. It includes supporting libraries, platform core classes, and the Java Virtual Machine (JVM). While JRE is available for standalone download, it is a component of the Java Development Kit (JDK).

Java Development Kit (JDK): The JDK is a software development kit that provides the necessary tools, libraries, and documentation for developing Java applications. It includes the Java compiler, which converts Java source code into bytecode that can be executed by the JVM.

The JDK also includes other development tools like debuggers, profilers, and application servers. It is essential for writing, compiling, and packaging Java programs. Some components of JDK includes:

(a) Java compiler (javac): This converts your human-readable Java code (source code) into bytecode, a machine-independent intermediate representation.

(b) Java Archive tool (jar): This tool packages your compiled class files (bytecode) and additional resources into a single archive file (JAR file) for easier distribution.

(c) Java debugger (jdb): This tool helps you identify and fix errors (bugs) in your code during development.

(d) Documentation: This tool provides extensive documentation on the Java language, APIs, and libraries.

Java Runtime Environment (JRE): The JRE is a software package that provides the necessary runtime environment for executing Java applications. It includes the JVM, which is responsible for executing Java bytecode. The JRE also contains Java class libraries and other components required to run Java programs. If you only need to run Java programs and not develop them, installing the JRE is sufficient.

Java Virtual Machine (JVM): The JVM is the runtime engine that executes Java bytecode. It is responsible for loading, verifying, and executing Java programs. The JVM converts bytecode into machine-specific code, allowing Java programs to run on different platforms without modification. It also provides memory management, garbage collection, and other runtime services. Therefore, in short, JDK is used to develop and compile Java OOP programs, JRE is used to run compiled Java OOP programs (bytecode) on any platform installed with a JRE. JVM is used to interpret and execute the bytecode instructions in Java OOP program.

FOR ONLINE READING ONLY

Java Installation

Before you can install Java in your computer, you may check if yet it is installed. Check this from various alternatives. In this case, you may check this through using the command prompt.

Activity 5.5:

Checking Java installed file

Aim:

To verify if Java is installed on a computers and ensure it correctly configured for use.

Materials: Computers with operating system (such as, windows, MacOS or Linux) internet access, access to command promp.

Instructions

Step 1: Type the following in Command Prompt in search field at the start bar:
C:\Users\Your Name>java -version
Note: Your Name here is the name of user in admin account

Step 2: Press enter after finishing typing.
If Java is installed, the window like in Figure 5.1 will appear depending on the version of Java.

```
java version "22" 2024-03-19
Java(TM) SE Runtime Environment (build 22+36-2370)
Java HotSpot(TM) 64-Bit Server VM (build 22+36-2370, mixed mode, sharing)
```

Figure 5.1: *Version of Java*

Activity 5.6:

Installing Java in Windows (i.e Windows 10)

Aim:

Installing Java on a Windows 10 computer by downloading the software from oracle.com, following the installation steps, and configuring environment variables to ensure Java is properly set up for development

Materials: Computers, Java Installer, web browser, and Internet

Instructions

Step 1: Download Java for free at oracle.com

FOR ONLINE READING ONLY

Step 2: Double click the downloaded executable file to install by following the steps that will show up. (when you finish go for previous learning activity verify if Java is installed)

Step 3: After installing,

Step 4: Go to "System Properties" by using the Control Panel > System and Security > System > Advanced System Settings)

Step 5: Under the "Advanced" tab Click the button "Environment variables"

Step 6: Click on the "Edit" button found in "Path" variable in the System variables section

Step 7: Click on the "New" button, then add the path found from where Java is located when installed, followed by **\bin**. Always by default, Java is installed in C:\Program Files\Java\jdk-11.0.1, the path will be **C:\Program Files\Java\jdk-22\bin** depending on the version. The window will show up like in Figure 5.2.

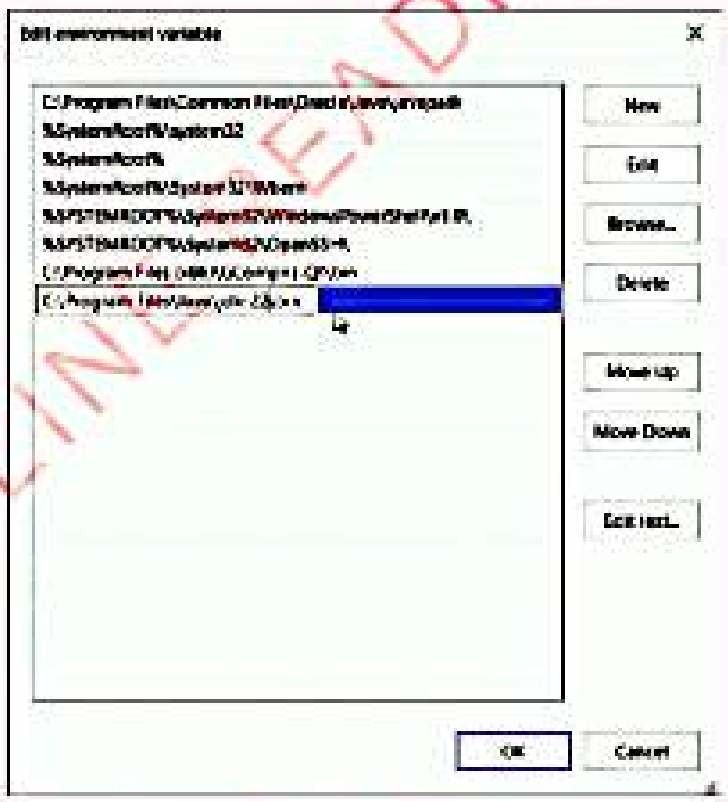


Figure 5.2: *Program files*

Step 8: Click "OK", and save the process.

When beginning to write Java programs, you'll likely require Java editors. Several popular Java editors are available, including: Notepad - a basic text editor, NetBeans - an open-source and free Java IDE, and Eclipse - a Java IDE developed by the Eclipse open-source community, among others.

FOR ONLINE READING ONLY

Writing and running a simple program

When developing applications in Java, it is essential to start with a class name that matches the filename. As a practice, let you create a simple program that prints "Hello Tanzanians". You can create your first Java file, named Tunalianzisha.java, using any text editor like Notepad. The file should contain a greeting message: "Hello Tanzanians".

Activity 5.7:

Writing a simple program-Hello Tanzanian

Aim:

To create a simple program that prints "Hello Tanzanians" to the console and learn how to compile the Java program using Command Prompt.

Materials: Computers, text editors, Java Development Kit, Command prompt

Instructions:

Step 1: Open a text editor, Notepad

Step 2: Type the following code:

```
class Tunalianzisha {
 public static void main(String[] args) {
 System.out.println("Hello Tanzanians");
    }
}
```

Step 3: Save your first program in a Notepad as "Tunalianzisha.java" at any location.

Compiling a Java program:

Step 4: Open Command Prompt (cmd), write the path where you saved your file then type "javac Main.java":

(When you open the cmd.exe, it might come with default path like C:\Users\admin> ; if you saved your file on the desktop, change the directory to Desktop using Help "cd Desktop" as seen in the Figure 5.3. Here, you are going to use the JDK compiler (called javac) to change your text file in a notepad into an executable program.

FOR ONLINE READING ONLY

```
C:\Users\admin>cd Desktop

C:\Users\admin\Desktop>Tu
'Tu' is not recognized as an internal or external command,
operable program or batch file.

C:\Users\admin\Desktop>javac TunaHanzisha.java

C:\Users\admin\Desktop>java TunaHanzisha
Hello Tanzanians

C:\Users\admin\Desktop>
```

Figure 5.3: *Change the directory*

Step 5: After compiling the file successfully, the command prompt will jump to next line. Type "java" followed by file name, in this case, "Tunalianzisha" then enter to run the program. The message "Hello Tanzanians" will appear, as in Figure 5.4.

```
Hello Tanzanians
```

Figure 5.4: *Hello Tanzanians message*

FOR ONLINE READING ONLY

Note:

(a) Make sure a class starts with an uppercase first letter. For example "class Tunalianzisha", T-is an uppercase.

(b) Java is Case-sensitive, meaning that if you type "tunalianzisha" instead of "Tunalianzisha" might bring an error.

(c) The compiled code in Java is known as bytecode, after compiling the program the new file with the .class extension will be created.

Description of each line of codes in a simple program in Activity 4.2

The keywords used in your first program are as follows:

(a) The "class" keyword in Java is utilised to declare a class.

(b) public: keyword serves as an access modifier. It indicates that the main method is accessible from anywhere outside the class.

(c) static: It indicates that the main method belongs to the class itself rather than to any specific instance of the class. When a method is declared as "static", it's referred to as a static

metho d. The primary benefit of static methods is that they can be invoked without needing to create an object. For instance, the "main()" method, executed by the JVM, does not require an object instantiation, thus conserving memory.

(d) The "void" return type denotes that the method does not return any value.

(e) "main" denotes the program's starting point.

(f) "String[] args" or "String args[]" is employed for command line arguments. It represents the arguments accepted by the main method, forming an array of strings. These arguments are utilised for passing command-line inputs to the Java program .

(g) "System.out.println()" is used to print a statement, where "System" is a class, "out" is a PrintStream class object, and "println()" is a method of the PrintStream class. You will observe further details of "System.out. println()" when you will be encounted with program examples.

Basic Java syntax and constructs

(a) The main method

In any Java program, there must be a method written The main() method is required and you will see it in every Java program:

```
public static void main(String[] args)
```

Any code within the main() method will be executed. Every Java program has a class name that must match the filename, and each program must include the main() method.

(b) Java output or print

System.out.println(): While inside the main() method, you can use the println() method to display a line of text on the screen:

```
public static void main(String[] args) {
  System.out.println("Hello Tanzanians");
}
```

Double quotes: When working with texts in this method, you have to make sure you wrap texts inside the double quotation, such as "Hello Tanzanians". Several lines of text can be printed, each in its own println() method. For example:

```
System.out.println("Hello Tanzanians");
System.out.println("Tanzania has National parks for Tourism attraction");
System.out.println("Tanzanian is a blessed country, full of natural resources");
```

The other format of printing information in Java is by using the Print() method which works the same to println(). For example,

FOR ONLINE READING ONLY

System.out.print("Protect environments from e-wastes");
System.out.print("Java is meant for Object Oriented Programming Design");

However, you can also use the println() method to print numbers. Unlike text, double quotes are not needed for numbers. For example, you can demonstrate the println() method for numbers as follows:

System.out.println(116);
System.out.println(985); and
System.out.println(5 + 6);

(c) Java comments

The java comments are statements that are not executed by the compiler and interpreter. The comments can be used to provide information or explanation about the variable, method, class or any statement. It can also be used to hide program code for specific time. There are 3 types of comments in java, namely: single line comment, multi line comment, and documentation comment

(i) *Java single line comment*
A single line comment is used to comment only one line. A single-line comment begins with a
// and ends at the end of the line. For example *"//This is a comment"*

(ii) *Java multi line comment*
A multi-line comment in Java starts with /* and ends with */. Anything between these symbols is ignored by the compiler. This type of comment can span several lines. For example, */* *This is a comment that is multi line. The comments may have several lines*/*

(iii) *Java documentation comment*
This type of comment, as shown in Table 5.4, is used to produce an HTML file that documents our program. The documentation comment begins with a /** and ends with a */.

Table 5.4: *Documentation comment*

Syntax	Example
/**Starting of the comments	/**
* The first comment	These
*Second comment and so on…	Are
*	important
*	comments to
*	use in Java
*Comments ends like this */	*/

FOR ONLINE READING ONLY

Data types

Java is a statically typed and also a strongly typed language. in Java, each type of data (such as integer, character, hexadecimal) is predefined as part of the programming language and all constants or variables defined within a given program must be described with one of the data types. Data types represent the different values to be stored in the variable. In Java, there are two categories of data types, namely: primitive data types and non-primitive data types.

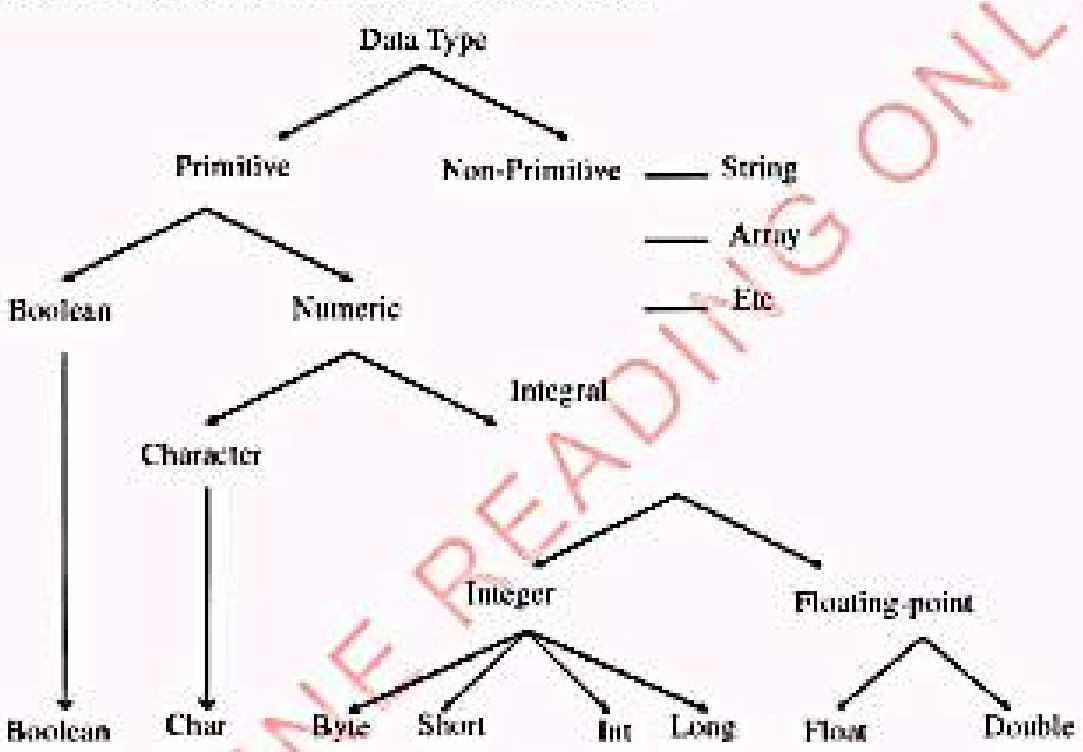


Figure 5.5: *Data types*

The primitive types

Java defines eight primitive types of data: byte, short, int, long, char, float, double, and boolean. The primitive types are also commonly referred to as simple types and they are grouped into the following four groups:

(i) Integers - This group includes byte, short, int, and long. All of these are signed, positive and negative values. The width and ranges of these integer types vary widely, as shown in Table 5.5:

Table 5.5: *Integers*

Name	Width in bits	Range
Long	64	-9, 223, 372, 036,854,775,808 to 9,223,372,036,854,775,807
Int	32	-2,147,483,648 to 2,147,483,647
Short	16	-32,768 to 32,767
byte	8	-128 to 127

FOR ONLINE READING ONLY

(ii) Floating-point numbers – They are also known as real numbers. This group includes float and double, which represent single- and double-precision numbers, respectively. The width and ranges of them are shown in Table 5.6:

Table 5.6: *Floating-point datatypes*

Name	Width in bits	Range
double	64	4.9e – 324 to 1.8e+308
float	32	1.4e – 045 to 3.4e + 038

Double uses 64 bits for storage. It provides a wider range and greater precision compared to float. Float uses 32 bits for storage. It offers a smaller range and less precision compared to double.

(iii) Characters - This group includes char, which represents symbols in a character set, like letters and numbers. Char is a 16-bit type. The range of a char is 0 to 65,536. In this case, there are no negative chars.

(iv) Boolean - This group includes boolean. It can have only one of two possible values, true or false.

Variables, operators and constants

Variables

A variable is the holder that can hold the value while the java program is executed. A variable is assigned with a datatype. It is name of reserved area allocated in memory. In other words, it is a name of memory location. There are three types of variables in java: local, instance and static. A variable provides us with named storage that our programs can manipulate. Each variable in Java has a specific type, which determines the size and layout of the variable's memory; the range of values that can be stored within that memory; and the set of operations that can be applied to the variable.

(a) Variable declaration in Java

Before using any variable, it must be declared. The following statement expresses the basic form of a variable declaration –
datatype variable [= value][, variable [= value] ...] ;
Here, data type is one of Java's data types and the variable is the name of the variable.

To declare more than one variable of the specified type, use a comma-separated list.

For, example:

```
int a, b, c; // Declaration of variables a, b, and c.
int a = 20, b = 30; // initialisation
byte B = 22; // Declaration initialises a byte type variable B.
```

FOR ONLINE READING ONLY

(b) Types of Variable

There are three types of variables in java: local variable, instance variable and static variable.

(i) Local variable

Local variables are declared inside the methods, constructors, or blocks. It is usually created when the method, constructor or block start to be executed. It is destroyed once the method, constructor, or block finishes execution. Local variables are visible only within the declared method, constructor, or block. They are implemented internally at stack level. There is no default value for local variables, so local variables should be declared and an initial value should be assigned before the first use. In this case, access specifiers cannot be used for local variables. Program Example 5.1 shows the local variable program:

Program Example 5.1:

Java program to demonstrate local variable

```
public class Car {
        public static void main(String[] args) {
   // Local variable
   int numOfDoors = 5;  // This variable only exists within this method
   System.out.println("My father bought a car with " + numOfDoors + " doors.");
   }
}
```

(ii) Instance variable

This refers to a variable declared within a class but outside any method. They are declared in a class, not inside a method, constructor, or block. When an object is allocated in the heap, a storage space is created for each instance variable. These variables hold values that are needed by multiple methods, constructors, or blocks, or they represent essential parts of an object's state that should be accessible throughout the class. Instance variables can also be declared at the class level. Access modifiers in Java, such as Default, Private, Protected, and Public, restrict the scope of these variables, methods, and constructors. By default, instance variables are visible to all methods, constructors, and blocks within the class. It is recommended to make these variables private, but they can be made visible to subclasses using access modifiers. Instance variables have default values:

(1) Numbers, the default value is 0,

FOR ONLINE READING ONLY

(2) Booleans it is false,

(3) Object references it is null.. Values can be assigned during declaration or within the constructor. Static methods cannot declare instance variables directly; they should be accessed using the fully qualified name within static methods. Program Example 5.2 shows an instance variable program:

Program Example 5.2:

Java program to demonstrate instance variable

```
public class PlantTrees {
    // Instance variable declared at class level but outside any method
    int AfforTrees = 783;

    public void digHoleMethod() {
        // Accessing the instance variable within the method
        System.out.println("Afforested trees value: " + AfforTrees);
    }
    public static void main(String[] args) {
        // Creating an object of the class to access the instance variable
        PlantTrees obj = new PlantTrees();
        // Calling the method which uses the instance variable
        obj.digHoleMethod();
    }
}
```

FOR ONLINE READING ONLY

(iii) Static variable

Class variables, also called static variables, are defined using the "static" keyword outside any method, constructor, or block. Only one instance of each class variable exists, no matter how many objects are created. Typically declared as public/private, final, and static, they act as constants with fixed values throughout the program's execution. Static variables are stored in static memory, initialised at program start, and discarded at program end. They share the same visibility as instance variables and are often public for accessibility. They default to 0, false, or null, similar to instance variables. Static variables are accessed using the class name and variable name. When declared public static final, they are constants and named in uppercase. Program Example 5.3 shows a static variable program.

Program Example 5.3:

Java program to demonstrate static variable

```
public class LibraryBooks {
 // Static variable declared using the static keyword
    static int ChkBookValue = 30;
    public void getTitle() {
        // Accessing the static variable within the method
        System.out.println("Number of books in shelves: " + ChkBookValue);
    }
 public static void main(String[] args) {
        // Calling the static variable directly without creating an object
System.out.println("Number of books in shelves (direct access): " + LibraryBooks.
ChkBookValue);
        // Creating an object of the class to access the method
        LibraryBooks obj = new LibraryBooks();
        // Calling the method which uses the static variable
        obj.getTitle();
    }
}
```

FOR ONLINE READINGS ONLY

Operators

In Java, an operator refers to a symbol utilised for executing specific actions or operations. Java offers a diverse array of operators designed to manipulate variables. These include symbols such as +, -, *, and /, among others. Java operators can be categorised into several groups, including:

(a) Arithmetic operators

This include Multiplicative: *, /, % and Additive: +, -. The arithmetic operators function similarly to those in algebra, allowing for various arithmetic operations. The Table 5.7 below illustrates these operators. For example, when integer A = 5, B = 15, the output is shown.

Table 5.7: *Arithmetic operators*

Operator	Description	Example	Output
+(Addition)	Adds values A &B	A+B	20
-(Subtraction)	Subtracts B from A	A-B	-10

Operator	Description	Example	Output
*(Multiplication)	Multiplies values A&B	A*B	75
/(Division)	Divides B by A	B/A	3
% (Modulus)	Divides left-hand operand by right-hand operand and returns remaainder	B%A	0
++ (Increment)	This adds 1 to its operand	A++ or ++A	6
– – (Decrement)	This operator subtracts 1 from its operand	A- - or -A	4

Arithmetic operators like +, -, *, and / in Java work similarly to other programming languages and algebra. They apply to any numeric data type, including char. However, note that when using the division operator (/) with integers, Java performs integer division, discarding any remainder (e.g., 10/3 results in 3). To get the remainder, use the modulus operator (%) (e.g., 10 % 3 equals 1).

In Java, the modulus operator % can be used with both integer and floating-point types. Hence, 10.0 % 3.0 also equals 1. Workout the following Program Example 5.4 on modulus in any editor then compile and run the program in JDK using cmd.exe:

Program Example 5.4:

Java program to demonstrate the % operator

```
class Modulus {
   public static void main(String args[]) {
int answer1, rem1;
double answer2, rem2;
answer1 = 10 / 3;
rem1 = 10 % 3;
answer2 = 10.0 / 3.0;
rem2 = 10.0 % 3.0;
    System.out.println("Answer and Remainder of 10 / 3: " +
    answer1 + " " + rem1);
    System.out.println("Answer and Remainder of 10.0 / 3.0: " +
    answer2 + " " + rem2);
    }
}
```

FOR ONLINE READING ONLY

Output:

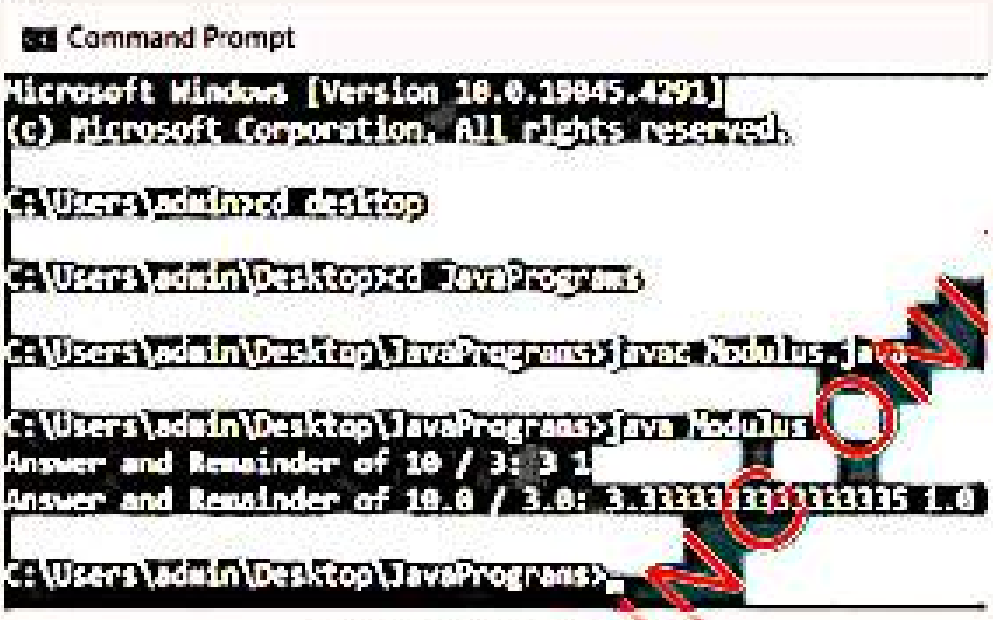


Figure 5.6: *Program in JDK*

In Program Example 5.4, you observed that for both integers and floating-point numbers, the remainder is the same, equals to 1.

Increment and decrement in Java

In Java, the increment operator (++) increases the value of a variable by 1, while the decrement operator (--) decreases it by 1. These operators can be applied as either prefix or postfix forms, altering whether the variable is incremented or decremented before or after its value is used in an expression. They are commonly used in loops for iterating through collections or controlling program flow based on numeric conditions. Workout Program Example 5.5 on increment and do extra in decrement in any editor then compile and run the program in JDK using cmd.exe:

Program Example 5.5:

Java program to demonstrate increment operator

```
public class Increment {
    public static void main(String[] args) {
        int x = 10;
        // Postfix, Using x++ when x=10
        int y1 = x++;
        System.out.println("Answer after x++: " + y1); // Output: 10
        System.out.println("Incremented value of x: " + x);  // Output: 11
```

FOR ONLINE READING ONLY

```
// Next for prefix when x is reset to 10
        x = 10;
        // Using ++x
        int y2 = ++x;
        System.out.println("Answer after ++x: " + y2); // Output: 11
        System.out.println("Incremented value of x: " + x);    // Output: 11
    }
}
```

Output:

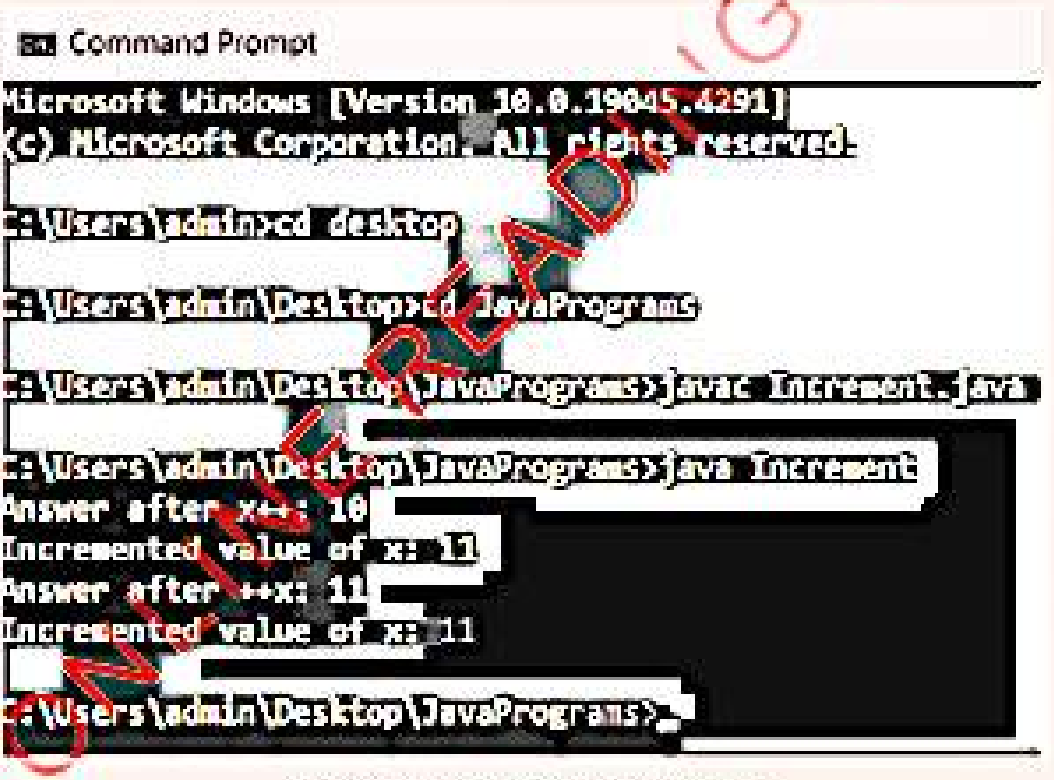


Figure 5.7: *Increment and decrement*

When an increment or decrement operator comes before its operand, Java executes the operation before retrieving the operand's value for the remaining expression. However, if the operator comes after its operand, Java fetches the operand's value first and then performs the increment or decrement operation. For example in the first case (x++) , y1 is set to 10 while still x becomes 11. For the second case, (++x), y2 is set to 11 when x will still be 11.

To see more about how the program works in Java operators, write the following Program Example 5.6 in any editor then compile and run the program in JDK using cmd.exe:

FOR ONLINE READING ONLY

Program Example 5.6:

Java program to demonstrate arithmetic operators

```
public class Operators {
    public static void main(String[] args) {
        int a = 12, b = 23, c = 0, d = 33, e = 10, f = 50;
        String x = "Welcome Java", y = "Tujifunze";
        System.out.println("a + b = "+(a + b));
        System.out.println("a - b = "+(a - b));
        System.out.println("x + y = "+x + y);
        System.out.println("a * b = "+(a * b));
        System.out.println("a / b = "+(a / b));
        System.out.println("a % b = "+(a % b));
            }
     }
```

Output:

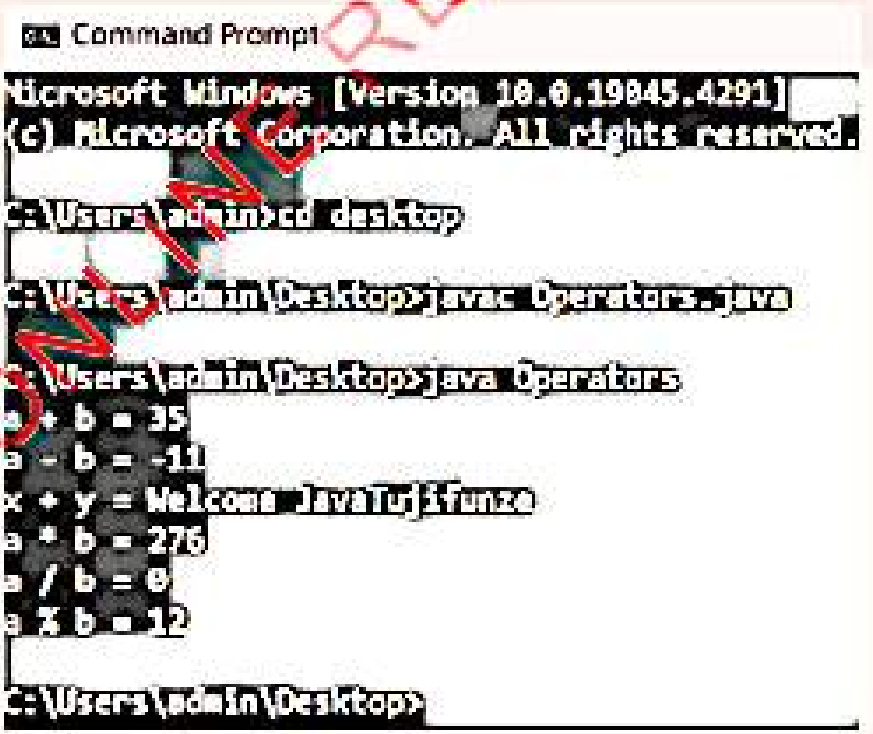


Figure 5.8: *Arithmetic operator*

(b) Relational operators

This include comparison: <, >, <=, >=, instanceof and Equality: ==, != . Table 5.8 shows the Relational operators that are supported by Java language.

FOR ONLINE READING ONLY

Table 5.8: *Relational operators*

Operator type	Symbol	Description	Syntax example
Less than or equal to	<=	Evaluates if the left operand is less than or equal to the right operand	A <= B
Greater than or equal to	>=	Evaluates if the left operand is greater than or equal to the right operand	A >= B
Instance of operator		This verifies if the object belongs to a specific type (class or interface) by using the syntax: (Object reference variable) instanceof (class/interface type).	boolean result = name instanceof String;
Equal to	==	Evaluates if the values of two operands are equal	A == B
Not equal to	!=	Evaluates if the values of two operands are not equal	A != B
Greater than	>	Evaluates if the value of left operand is greater than the value of right operand	A > B
Less than	<	Evaluates if the value of left operand is less than the value of right operand	A < B

Practice how the program works, just write the Program Example 5.7 in any editor then compile and run the program in JDK using cmd.exe:

FOR ONLINE READING ONLY

Program Example 5.7:

Java program to demonstrate relational operators

```
public class RelationalOperators
{
 public static void main(String[ ] args)
 {
                int a = 300, b = 200;
        boolean condition = true;
        //These codes shows various conditional operators
        System.out.println("a == b :" + (a == b));
        System.out.println("a < b :" + (a < b));
        System.out.println("a <= b :" + (a <= b));
        System.out.println("a > b :" + (a > b));
        System.out.println("a >= b :" + (a >= b));
        System.out.println("a != b :" + (a != b));
        System.out.println("condition==true :" + (condition == true));
        }
          }
```

Output:

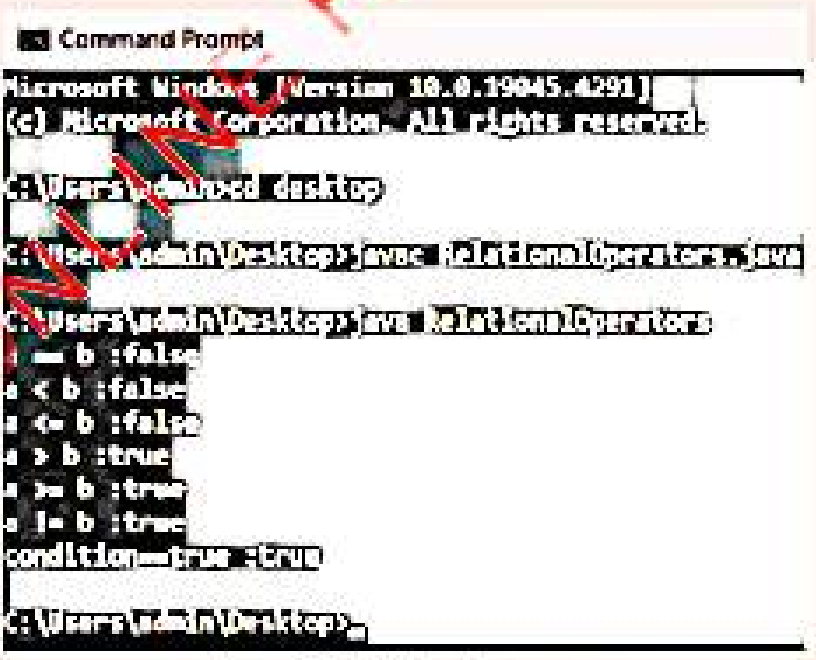


Figure 5.9: *Relational operators*

(c) Bitwise operators

This include Bitwise AND: &, Bitwise exclusive OR: ^, Bitwise inclusive OR: | and Shift operators: <<, >>, >>>. Table 5.9 shows the relational operators that are supported by Java language.

FOR ONLINE READING ONLY

Table 5.9: *Bitwise operators*

Operator Type	Symbol	Description	Syntax Example
left shift	<<	This operation moves the first operand a certain number of bits to the left, as determined by the second operand. Zeros are filled in from the right as the shift occurs.	A<<n
right shift	>>	This operation moves the first operand a specified number of bits to the right, as determined by the second operand. Any bits that are shifted out of the right end are discarded.	A>>n
Unsigned right shift (Zero fill right shift)	>>>	The value of the left operand is shifted to the right by the number of bits specified by the right operand. Any empty spaces created by the shift are filled with zeros	A>>>n
Bitwise AND	&	This operation copies a bit to the result if it is present in both operands. The result will be 1 only if both bits are 1	A&B
Bitwise OR	|	This operation transfers a bit to the result if it is present in either operand. The result will be 1 if either of the bits is 1	A|B
Bitwise XOR	^	In the result, each bit is set to 1 if the corresponding bits in both operands are not the same	A^B
Bitwise complement	~	This operation it changes zeros to ones and ones to zeros	~A

FOR ONLINE READING ONLY

Table 5.10 shows the the summary of the truth table for bitwise operators:

Table 5. 10: *Truth table*

A	B	A&B	A|B	A^B	~A
0	0	0	0	0	1
0	1	0	1	1	1
1	0	0	1	1	0
1	1	1	1	0	0

To calculate the bitwise XOR operation between two integers, 12 and 25, you do not need a Java program. Here is how you can do it manually, take the binary representations of each number:

12 = 00001100
^ 25 = 00011001
_____00010101 . This is equal to 21, figure in decimal digits.

Workout the Program Example 5.8 in any editor then compile and run the program in JDK using cmd.exe to see how the bitwise operators works:

Program Example 5.8:

Java program to demonstrate bitwise operators

```
public class BitwiseOPR{
        public static void main(String[] args){
            int x = 15;
            int y = 33;
            System.out.println("x&y = " + (x & y));
            System.out.println("x|y = " + (x | y));
            System.out.println("x^y = " + (x ^ y));
            System.out.println("~x = " + ~x);
           }
}
```

FOR ONLINE READING ONLY

Output:



Figure 5.10: *Bitwise operators*

Example of Bitwise right shift and Bitwise left shift

(i) Perform the bitwise left shift operation on the integer 6. Left shift each bit by 1. Write the answer in binary and decimal.

(ii) Perform the right shift by two bits operation on the integer 7 without use of Java program.

(iii) Perform the right shift by two bits operation on integer 60 without use of Java program.

Solution:

(i) 6=0110

6<<1=1100 in Binary; 6<<1=12 in Decimal

Program Example 5.9:

Java program to perform left shift by one bit operations

```
class Bits {
   public static void main(String[] args) {
      int number = 6;
      // 1 bit left shift
   System.out.println("Number left shifted by 1 bit: " + (number <<1));    // prints 12
   }
}
```

FOR ONLINE READING ONLY

Output:

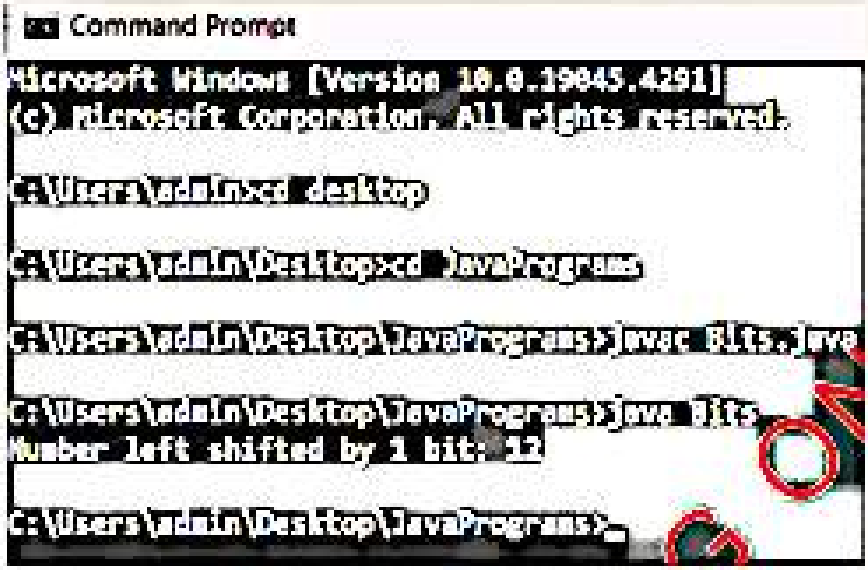


Figure 5.11: *Bit operations*

(ii) *7 = 0111*

7>>2= 1 in binary ; 7>>2=1 in Decimal

(iii) 60= 111100

60>>2 = 001111 in Binary: 60>>2 = 15 in Decimal

(d) Logical operators

This include logical AND : &&, logical OR : || and logical NOT : ~ !. Figure 5.12 and Table 5.11 shows the logical operators supported by Java.

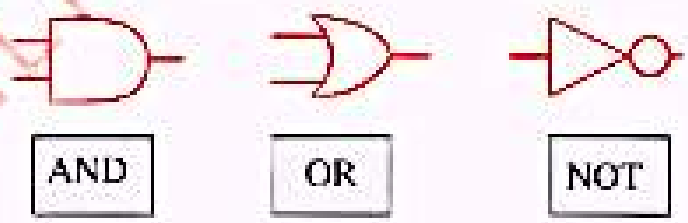


Figure 5.12: *Logical operators*

Table 5.11: *Logical operators*

Operator Type	Symbol	Description	Syntax example
Logical AND	&&	The condition becomes true when both the operands are non-zero	A&&B
Logical OR	||	The condition becomes true when any of the two operands are nonzero	A||B

FOR ONLINE READING ONLY

Operator type	Symbol	Description	Syntax example
Logical NOT	!	The logical NOT operator is used to invert or reverse the logical state of its operand. When a condition is true, applying the logical NOT operator will result in the condition becoming false	!(A&&B)

(e) Assignment operators " ="

Table 5.12: Assignment Operators

Operator	Description	Example
=	Assigns the value on the right to the variable on the left.	int a = 5;
+=	Adds the value on the right to the variable on the left and assigns the result to the variable on the left.	a += 3; // a = a + 3
-=	Subtracts the value on the right from the variable on the left and assigns the result to the variable on the left.	a -= 2; // a = a - 2
*=	Multiplies the variable on the left by the value on the right and assigns the result to the variable on the left.	a *= 4; // a = a * 4
/=	Divides the variable on the left by the value on the right and assigns the result to the variable on the left.	a /= 2; // a = a / 2
%=	Calculates the remainder of the division of the variable on the left by the value on the right and assigns the result to the variable on the left.	a %= 3; // a = a % 3
&=	Performs a bitwise AND operation on the variable on the left and the value on the right and assigns the result to the variable on the left.	a &= 1; // a = a & 1

FOR ONLINE READING ONLY

^=	Performs a bitwise XOR operation on the variable on the left and the value on the right and assigns the result to the variable on the left.	a ^= 3; // a = a ^ 3
<<=	Performs a left bitwise shift on the variable on the left by the number of positions specified on the right and assigns the result to the variable on the left.	a <<= 2; // a = a << 2
>>=	Performs a right bitwise shift on the variable on the left by the number of positions specified on the right and assigns the result to the variable on the left.	a >>= 1; // a = a >> 1
>>>=	Performs an unsigned right bitwise shift on the variable on the left by the number of positions specified on the right and assigns the result to the variable on the left.	a >>>= 1; // a = a >>> 1

(f) Ternary operator

The ternary operator in Java is a shorthand for an if-else statement. It is used to evaluate a boolean expression and return one of two values based on the result of the evaluation.

Syntax

In Java, the ternary operator (also known as the conditional operator) has the following syntax:

variable = (condition) ? expression1 : expression2;

In this syntax;

condition: Represent a boolean expression that evaluates to either true or false.

expression 1: Represents "If the condition is true, the variable is assigned the value of expression1",

expression 2: Represent "If the condition is false, the variable is assigned the value of expression2".

The Program Example 5.10 demonstrates the ternary operators:

FOR ONLINE READING ONLY

Program Example 5.10:

Java program to implement the Ternary Operator

```
int a = 10;
int b = 20;
// Using the ternary operator to determine the larger of two numbers
int max = (a > b) ? a : b;
System.out.println("The maximum value is " + max);
```

Output: The maximum value is 20

(g) Conditional operator (? :)

Since the conditional operator has three operands, it is referred as the ternary operator.

This operator consists of three operands and is used to evaluate Boolean expressions. The goal of the operator is to decide, which value should be assigned to the variable. The operator is written as – variable x = (expression) ? value if true : value if false

Program Example 5.11:

Java program to implement the Conditional Operator

```
public class example{
public static void main(String args[])
{
        int a, b;
        a = 10;
        b = (a == 0) ? 20: 30;
    System.out.println( "b : " + b );
    }
 }
```

Output:

```
b: 30
```

FOR ONLINE READING ONLY

(h) Unary operator: This includes Postfix: expr++ expr-- and Prefix: ++expr, --expr, +expr, and -expr as shown in Table 5.13.

Table 5.13: *Unary operator*

Operator	Description	Example	Explanation
+	Unary plus operator (promotes the value)	+5	No effect on the value; it is typically used to explicitly indicate a positive number.
-	Unary minus operator (negates the value)	-5	Negates the value, changing positive numbers to negative and vice versa.
++	Increment operator (increases the value by 1)	++a or a++	Pre-increment (++a) increases value before use; post-increment (a++) increases value after use.
--	Decrement operator (decreases value by 1)	--a or a--	Pre-decrement (--a) decreases value before use; post-decrement (a--) decreases value after use.
!	Logical complement operator (inverts boolean value)	!true	Inverts the boolean value, changing true to false and vice versa.

Operator precedence in Java

Operator precedence in Java defines the rules that determine the order in which different operators in an expression are evaluated. When an expression contains multiple operators, operator precedence dictates which operations are performed first. Table 5.14 shows operator precedence as used in Java programs.

Table 5.14: *Operator precedence in Java*

Category	Operator	Associativity
Primary	() [] .	Left-to-right
Unary	++ -- + - ~ ! (type) new	Right-to-left
Multiplicative	* / %	Left-to-right
Additive	+ -	Left-to-right
Shift	<< >>	Left-to-right
Relational and type testing	< <= > >= instance of	Left-to-right
Equality	== !=	Left-to-right

FOR ONLINE READING ONLY

Category	Operator	Associativity
Bitwise AND	&	Left-to-right
Bitwise XOR	^	Left-to-right
Bitwise OR	|	Left-to-right
Logical AND	&&	Left-to-right
Logical OR	||	Left-to-right
Ternary	?:	Right-to-left
Assignment and Lambda	= += -= *= /= %=>>=<<= &= ^=| = =>	Right-to-left

Constants in Java

In Java, a constant is a value that cannot be changed once it is assigned. To declare a constant, the "final" keyword is used, indicating that the variable's value is unmodifiable. Think of a constant like a labeled box where you store something that cannot be changed unless you relabel it.

In Java, it is common practice to write constant names in capital letters, with multiple words separated by underscores (_). Constants can also be defined as static, which allows them to be accessed without creating an instance of the class.

One important aspect of constants is that they can hold any type of data, such as numbers or text. They are similar to recipes where a fixed amount, like 1 teaspoon of sugar, is always used in a standard tea cup, regardless of the number of cups you plan to drink (unless you want very sweet tea).

(a) Importance of using constants in Java

There are several reasons of using constant as variables in Java, some of the reasons includes:

(i) Enhancing readability: Employing descriptive names for constants enhances code comprehension.

(ii) Improving maintainability: Altering a single constant is simpler than hunting down and replacing scattered values in your codebase. It is similar to adjusting a formula once rather than revising each set individually!

(iii) Potential performance boost: In select scenarios, employing constants may yield marginally faster programs since values needn't be recalculated repeatedly. It's analogous to pre-measuring ingredients to streamline baking.

(b) Declaration of constant variable in Java

In Java, there is no specific "constant" keyword. Instead, two keywords are used together to declare constants:

Final: The "final" keyword ensures that the value of the constant cannot be changed after it is assigned. It is similar to following a guideline or rule on how something should be done.

FOR ONLINE READING ONLY

Static: The "static" keyword allows the constant to be accessed throughout the program, meaning it can be used in multiple parts of the code.

To declare a constant in Java, the "final" keyword is used. Constant names, or identifiers, should typically consist of nouns or noun phrases made up of multiple words.

For example:

(i) Final int MAX_VALUE = 359; declares a constant named MAX_VALUE with a value of 359.

(ii) Final int MONTHS_IN_YEAR = 12; declares a constant variable named MONTHS_IN_YEAR with a value of 12.

Program Example 5.12:

Java program to demonstrate constant variable

```
public class Max {
    public static final int MAX_VALUE = 359;
    public static final double PI = 3.14159;
    public static void main(String[] args) {
        System.out.println("The maximum value is: " + MAX_VALUE);
        System.out.println("The value of PI is: " + PI);
    }
}
```

Output:

```
C:\Users\admin\Desktop\JavaPrograms>javac Max.java

C:\Users\admin\Desktop\JavaPrograms>java Max
The maximum value is: 359
The value of PI is: 3.14159

C:\Users\admin\Desktop\JavaPrograms>
```

Figure 5.13: *Demonstrate constant variable*

FOR ONLINE READING ONLY

(c) Types of constants

Constants can encompass various types of data that your program may use, such as:

(i) Number (int, double) - For instance, you can declare a constant like final static int PI = 3.14159;

(ii) Text (String) - Another example is defining a constant such as final static String INSTITUTE_NAME = "Wonderful Department at Tanzania Institute of Education";

Functions, objects and classes in Java OOP

Functions

(a) Meaning of a function

Think of functions as small, specialised tools within your program. Each function handles a particular job, such as computing an area, verifying a user's age, or showing a message. These functions can be used repeatedly in writing code so as to enhance cleanliness and organisation.

(b) Importance of using functions

There are various benefits of using functions in Java. These benefits includes:

(i) *Code reusability:* Instead of writing the same code multiple times, you can create a function and call it whenever you need that functionality. This saves time and effort.

(ii) *Improved readability:* By breaking down complex tasks into smaller functions, your code becomes easier to understand for both you and others.

(iii) *Modular design:* Functions promote a modular approach, where your program is built from independent, reusable components.

(c) Declaration of functions in Java

In Java, methods (also called functions) are like tools within a toolbox (class). To create a method, you first decide who can use it (public or private), what kind of information it might give back (return type), and give it a clear name. You can also give it instructions to work with (parameters). Finally, you tell the method what to do (method body), and optionally what answer to provide (return statement).

Functions in java can be defined in several ways:

(i) *Public static* keyword: This keyword combination makes a function to be accessible from other parts of a program. This is like opening the door so that a function to be used anywhere in a code, as shown in Program Example 5.13.

Program Example 5.13:

Java program to demonstrate method call(public static function)

```
public class MathCalculations {
    // This function is declared as public static, making it accessible from other parts of
    //the program
    public static int add(int num1, int num2) {
        return num1 + num2;
    }
```

FOR ONLINE READING ONLY

```
    public static void main(String[] args) {
        // Calling the public static function add from the MathCalculations class
        int result = MathCalculations.add(230, 300);
        System.out.println("The sum of 230 and 300 is: " + result);
    }
}
```

Output:

```
C:\Users\admin\Desktop\JavaPrograms>java MathCalculations
The sum of 230 and 300 is: 530

C:\Users\admin\Desktop\JavaPrograms>
```

Figure 5.14: *Demonstrate method call*

(ii) *Return types:* Functions can optionally return a value (for example when calculating the area of such as rectangle, triangle). When you need a function to return value, the type of data must be specified (like 'int', 'String' or 'double'), as shown in the Program Example 5.14.

Program Example 5.14:

Java program to demonstrate return type function

```
public class AreaCalculator {
    // Here a function computes the area of a square and returns the result
    public static int computeSquareArea(int Length) {
        return Length * Length;
    }
    public static void main(String[] args) {
        // computeSquareArea function is called for printing answer
        int squareArea = AreaCalculator.computeSquareArea(25);
        System.out.println("Area of square: " + squareArea);
    }
}
```

Output:

```
C:\Users\admin\Desktop\JavaPrograms>java AreaCalculator
Area of square: 625
```

Figure 5.15: *Demonstrate return type function*

FOR ONLINE READING ONLY

(iii) *Parameters:* Functions can receive inputs, called parameters that provide data to work with. Parameters are declared within parentheses after the function name.

Recalling from Program Example 5.14, In the main method, the computeSquareArea function is called with an argument of 25, which represents the Length of the square.

(iv) *Calling functions:* A code can be executed within a function calling it by its name, followed by parentheses that may include parameters.

In this program example, the "calculateSquareArea(25)" function has been called and the returned result will be stored in the SquareArea to provide output

Java Classes and Objects

In Java, everything is tied to classes and objects, each carrying attributes and methods. For instance, imagine a car as an object with attributes such as weight and color, and actions like drive and brake which are the method as it was elaborated earlier in previous sections. In this case, a class serves as a blueprint for creating objects, similar to an object constructor.

(a) Creating a class

When you want to create a class, the keyword "class" is used. You have to declare a class using the class keyword, followed by a unique identifier to name it. The class body is then enclosed within a pair of opening and closing braces "{ }". This structure is shown in a syntax below:

```
class Identifier
    {
    //The code for class body will be written here
    }
```

The following example declares a class named Animal:

```
class Animal
{
// The code for class body will be written here
}
```

From the above example the of class "Animal", a class body should be occupied with fields, methods, and constructors. The class body can be filled with variety of elements. The following example shows how you can create a class named "Main" with variable "x":

```
public class Main {
  int x = 116;
}
```

FOR ONLINE READING ONLY

(b) Creating an object

An object is created from a class in Java. From the previous class "Main", objects can be created as it is shown in the following Program Example 5.15 by specifying the class name and object name. The object is created using the keyword "new":

Program Example 5.15:

Java program to create an object myDemo

```
public class Main2 {
      int x = 116;
   public static void main(String[] args) {
   Main2 myDemo = new Main2();
  System.out.println(myDemo.x); //This print the value of x, where the object created
                                 //access the attribute from the main class
  }
}
```

Output:

```
116
```

Figure 5.16: *Creating an object*

(c) Creating multiple objects

Multiple objects can be created in Main2 class as it is shown in Program Example 5.16.

Program Example 5.16:

Java program to create multiple objects

```
public class Main2 {
   int x = 116; int y = 556;
   public static void main(String[] args) {
      Main2 myDemo1= new Main2(); // This code creates the Object 1
      Main2 myDemo2= new Main2(); // This code creates the Object 2
      System.out.println(myDemo1.x);
      System.out.println(myDemo2.y);
      }
  }
```

Output:

```
C:\Users\admin\Desktop\JavaPrograms>java Main2
116
556
```

Figure 5. 17: *Creating multiple objects*

FOR ONLINE READING ONLY

(d) Creating multi-clases with main()

In Java, you can create an object of a class and utilise it in another class. In Java this is commonly employed for improved class organisation, where one class encompasses all attributes and methods, while another class contains the main() method that carries the code to be executed. Note that, in the real world, when you develop applications, small applications can be accommodated by a single class, but larger applications often require multiple classes. Very important to note is that the name of the Java file must corresponds to the class name. In Program Example 5.17, learn by establishing two classes: Book.java and Application1.java with the main method. These should be contained within the same directory.

Program Example 5.17:

Java program to demonstrate multiple class Book.java and Application1.java

```
public class Book {
  int x = 117;
}
class Application1 {
    public static void main(String[] args) {
    Book myApp = new Book();
    System.out.println(myApp.x);
    }
}
```

Output:

When two class files compiled, it will give you the output:

```
C:\Users\admin\Desktop\JavaPrograms>javac Book.java

C:\Users\admin\Desktop\JavaPrograms>javac Application1.java
```

Figure 5.18: *Demonstrating multiple class*

When you run the file that contain the main method, for this case, Application1.java you will get the output which is the object created from the main class Book.java:

```
C:\Users\admin\Desktop\JavaPrograms>java Application1
117
```

Figure 5.19: *Demonstrating multiple class*

FOR ONLINE READING ONLY

Exercise 5.2

1. Create a class named MyFamily utilising keywords "public" and "class".
2. Use the class MyFamily, create the object called "OurGarden" using the keyword "New".
3. Write codes that will use the object "OurGarden" to access and print the value of any variable declared with integer data type in the class MyFamily.

Constructors and destructors in Java

In Java, constructors and destructors play crucial roles in OOP. Constructors are special methods used to initialise objects and are called automatically upon object instantiation. Java does not have destructors like C++, relying instead on garbage collection to manage memory by cleaning up unused objects.

Constructors in Java

Constructors are special methods in a class that are automatically called when objects are created from that class. They initialise objects when they are created. They have the same name as the class and does not have a return type. They ensure objects are in a valid state from the beginning.

(a) Syntax for a constructor

Any constructor have a name and follows a syntax as shown below:

```
ClassName() {
// constructor body
}
```

(b) Importance of using constructors

(i) *Guaranteed initialisation*: Constructors guarantee that objects are created with their properties properly initialised, preventing potential errors or unexpected behavior caused by uninitialised properties.

(ii) *Flexibility*: By creating constructors with different arguments, you have the flexibility to customize the initial state of your objects according to specific requirements or preferences.

(c) Creating constructors

When you create a constructor in Java, adhere to the following steps as demonstrated in Program Example 5.18. Steps to develop constructor:

1st Step: Declare a method with the same name as the class.

2nd Step: Do not specify both return type and void for the constructor.

3rd Step: Provide parameters to the constructor to initialise the object with specific values and perform any required initialisation or setup for the object within a constructor.

FOR ONLINE READING ONLY

Program Example 5.18:

Java program to create a constructor

```
// This create a Application class
public class Application {
  int x;  // This declares a class attribute
  // Create a class constructor for the Application class, with method declared the same //
name as a class
  public Application() {
    x = 118;  // Set the initial value for the class attribute x
  }
  public static void main(String[] args) {
    Application myApp = new Application(); // Create an object of class Application (This
                                                    //will call the constructor)
    System.out.println(myObj.x); // Print the value of x
  }
}
```

Observe Program Example 5.19 that is declared with string and is set to print information of an object with string value of an attribute.

Program Example 5.19:

Java program to create a constructor

```
public class Car {
  private String brand;
  private String model;
     // Constructor with parameters
  public Car(String brandName, String CarModel) {
     this. brandName = brandName;
     this. CarModel = CarModel;
  }
     public void carDetails() {
     System.out.println("Brand Name: " + brandName);
     System.out.println("Car Model: " + CarModel);
  }
     public static void main(String[] args) {
     // Creating an object using the constructor
     Car schoolCar = new Car("Nissan", "X-Trail");
           // Calling a method to display car information
     schoolCar.carDetails();
  }
}
```

FOR ONLINE READING ONLY

(d) Types of constructors

In Java, there are two main types of constructors:

(i) ***Default constructor:*** Is a kind of constructor that has no arguments (empty parentheses). It is used to give basic default values for the properties of a particular object. Java tends to create a default constructor automatically that is not initialised if a constructor is not declared in your class. Program Example 5.20 demonstrates the default constructor.

Program Example 5.20:

Java program to create a class with default constructor

```
public class Car {
        String color;
        String make;
        String model;
        // Default constructor (no arguments)
    public Car() {
        color = "black"; // Default color
        make = "unknown";
        model = "unknown";
    }
        //other methods can be specified here
}
// Creating car objects using constructors
Car car1 = new Car(); // Uses default constructor
```

(ii) Parameterised constructor : This type of constructor enables you to specify arguments within the parentheses during object creation. These arguments are commonly utilised to initialise the object's properties with values that you provide. In a class, you can have multiple parameterized constructors, each accepting different arguments to accommodate various initialisation scenarios. Program Example 5.21 shows a sample parameterised constructor.

FOR ONLINE READING ONLY

Program Example 5.21:

Java program to create a class with default constructor

```
// This code shows a parameterized constructor
class MyClass {
   int x;
   MyClass(int I) { // The constructor here has a parameter
      x = I;
   }
}
class ConstraDemo {
   public static void main(String args[]) {
      MyClass t1 = new MyClass(10);
      MyClass t2 = new MyClass(88);
      System.out.println(t1.x + " " + t2.x);
   }
}
```

Output:

```
10 88
```

Figure 5.20: *Output of Java class print*

In this version of the program, the MyClass ( ) constructor defines one parameter called I, which is used to initialise the instatnce variable, x. Thus, when the line MyClass t1 =new MyClass (10) ; executes, the value 10 is passed to I, which is then assigned to x.

The choice of constructor relies on you as a programmer depending on specific requirements. If you prefer objects to always have default values, a default constructor could be adequate. Nevertheless, if you require greater control over initialisation or the flexibility to create objects with varying initial states, parameterised constructors become necessary.

(e) Constructor overloading

Constructor overloading refers to creating multiple constructors in a class with different parameter lists. This enables the creation of objects with different argument combinations, offering flexibility in initialising objects to suit various scenarios. In Java, constructor overloading is similar to method overloading. In method overloading, various method definitions are generated according to different parameters. Through constructor overloading, you can achieve diverse approaches to object initialisation, accommodating varying numbers of parameters. Following is a

FOR ONLINE READING ONLY

piece of code example of overloading constructor:

```
public class Book {
    String title;
    String author;
       // This code shows constructor with two parameters
    public Book(String t, String a) {
       title = t;
       author = a;
    }
       // This code shows overloaded constructor with three parameters
    public Book(String t, String a, int pages) {
       title = t;
       author = a;
       pageCount = pages;
    }
}
```

(f) Destructors in Java

Destructors in Java are effectively simulated through the finalize() method. This method serves a purpose similar to that of traditional destructors. Java works for a destructor with the help of the Garbage collection. When a program need to call the destructor in Java, the finalise method must be used. This method is not independent because it relies on Garbage Collection. The garbage collector is a thread that deletes or destroys the object which has no use in the heap area. For example, if an object is linked to file, database, or network connections, it must first close all associated connections before being deleted. This ensures that resources are properly released prior to garbage collection. The functions are closed by calling the finalize() method.

(i) Syntax of a finalize() method

The following is a syntax for the finalize() method:

```
protected void finalize() {
   //finalization code is put here
}
```

To incorporate a finalizer into a class, you simply define the finalize() method. Whenever the Java runtime system is ready to reclaim an object of that class, it invokes this method. Within the finalize() method, you specify the necessary actions that should be executed before the object is deleted.

In this context, the modifier "protected" restricts access to the finalize() method. It is important to note that finalize() is invoked right before garbage collection

FOR ONLINE READING ONLY

occurs. It is not invoked when an object loses scope, for instance. Consequently, it is unpredictable when, or even if, finalize() will be executed.

(ii) Finalize() method

Program Example 5.22:

Java program to demonstrate finalize()method

```
public class MyHome {
   private int value;
   // Constructor
   public MyHome(int value) {
      this.value = value;
      System.out.println("Object created with value: " + value);
   }
     // Destructor-like method
   protected void finalize() {
      // Cleanup actions
      System.out.println("Object with value " + value + " destroyed!");
   }
   public static void main(String[] args) {
      // Create objects
      MyHome obj1 = new MyHome(15);
      MyHome obj2 = new MyHome(25);
      // Several operations continues
      // This set objects 1 and 2 to be null to make it eligible for garbage collection
      obj1 = null;
      obj2 = null;
      // This line of code is going to set garbage collection
      System.gc(); // This cod explicitly request garbage collection
      System.out.println("End of program");
   }
}
```

Output:

```
Object created with value: 15
Object created with value: 25
End of program
Object with value 25 destroyed!
Object with value 15 destroyed!
```

Figure 5.21: *Demonstrating finalize()method*

FOR ONLINE READING ONLY

Exercise 5.3

1. Write a Java class 'Car' with a constructor that initialises the 'brand' and 'model' attributes. Demonstrate its usage in a 'Main' class.
2. Create a class 'Book' with attributes 'title', 'author', and 'price'. Implement a parameterised constructor that initialises these attributes. Write a 'Main' class to create and display details of multiple 'Book' objects.
3. Define a class 'Employee' with attributes 'name' and 'salary'. Include a default constructor that initialises name to "Unknown" and 'salary' to 0. Demonstrate creating an 'Employee' object using this default constructor in a 'Main' class.
4. Discuss the concept of destructors in Java. Explain why Java does not have a 'finalize' method like C++ and how garbage collection handles resource cleanup.
5. Create a class 'DatabaseConnection' that simulates opening a database connection in its constructor and closing it in a method 'closeConnection()'. Illustrate the importance of proper resource management using constructors and methods in Java.

Implementation of key OOP features in Java

The implementation of key OOP features in Java, including inheritance, encapsulation, polymorphism, and abstraction, enables developers to create modular and reusable code. These principles help build robust, scalable, and maintainable applications.

Inheritance in Java

Java allows to borrow (inherit) features from one class to create a new class. Using inheritance, you can create a general class that defines traits common to a set of related items. This class can then be inherited by other, more specific classes, each adding those things that are unique to it. This inheritance concept has two key players:

(a) The "Child" class (***Subclass***): This class inherits properties and methods from another class.

(b) The "Parent" class (***Superclass***): This is the class whose features are being inherited. The child class inherits all of the variables and methods defined by the superclass and adds its own, unique elements.

(a) Declaration in inheritance

To declare a superclass, the class keyword followed by the class name is used. Then the superclass with its fields and methods is declared. To declare a subclass, the class keywords followed by the subclass name, then, the extends keyword followed by the superclass name is used as shown in Figure 5.22. In order to inherit the features, that is, the new class to inherit from the existing class, the "extends" keyword is used.

FOR ONLINE READING ONLY

Program Example 5.23:

Java program to demonstrate inheritance

```
class Vehicle {
  protected String brand = "Nissan";       // This shows vehicle attribute
  public void horning () {                 // This shows vehicle method
     System.out.println("Poooo, Pooo!");
     }
  }
class Car extends Vehicle {
  private String modelName = "X-Trail";    // Car attribute
  public static void main(String[] args) {
    // Create a myCar object
    Car myCar = new Car();
    // This code call the horning() method (from the super class) on the myCar object
    myCar.horning();
    // This code display the value of the brand attribute (from the Vehicle class) and the
    //value of the modelName from the Car class
    System.out.println(myCar.brand + " " + myCar.modelName);
  }
}
```

Output:

```
Poooo, Pooo!
Nissan X-Trail
```

Figure 5.22: *Demonstration of inheritance*

(b) Visualisation of the program code in Java

In order to visualise and explore what is going on in the code execution, you can use an open source software called Jeliot3 as shown in Figure 5.23. You can visualise the code execution through Activity 5.8.

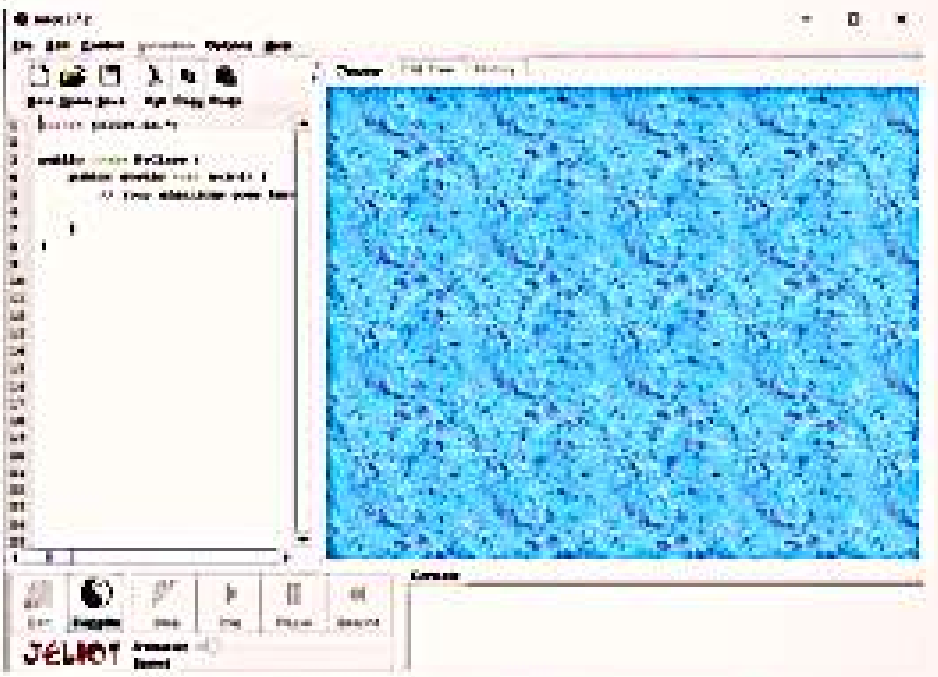


Figure 5.23: *Jeliot 3 Graphical User Interface*

FOR ONLINE READING ONLY

Activity 5.8:

Visualisation of execution code in inheritance of class Vehicle

Aim: Understand and visualise the execution of code involving inheritance in Java using the Vehicle class and its subclasses.

Materials: Computer with internet access, Jeliot 3 installed or access to an online Java compiler, Sample code

Instructions:

Step 1: Open the Jeliot compiler and orient with the interface, as shown in Figure 5.24.



Figure 5.24: *Features of Jeliot 3*

Step 2: Type or paste the code you need to visualise in code execution area as shown in Figure 5.23, then click the button compile to debug the program.

Step 3: Click the button play to run the codes step by step as shown in Figure 5.25.

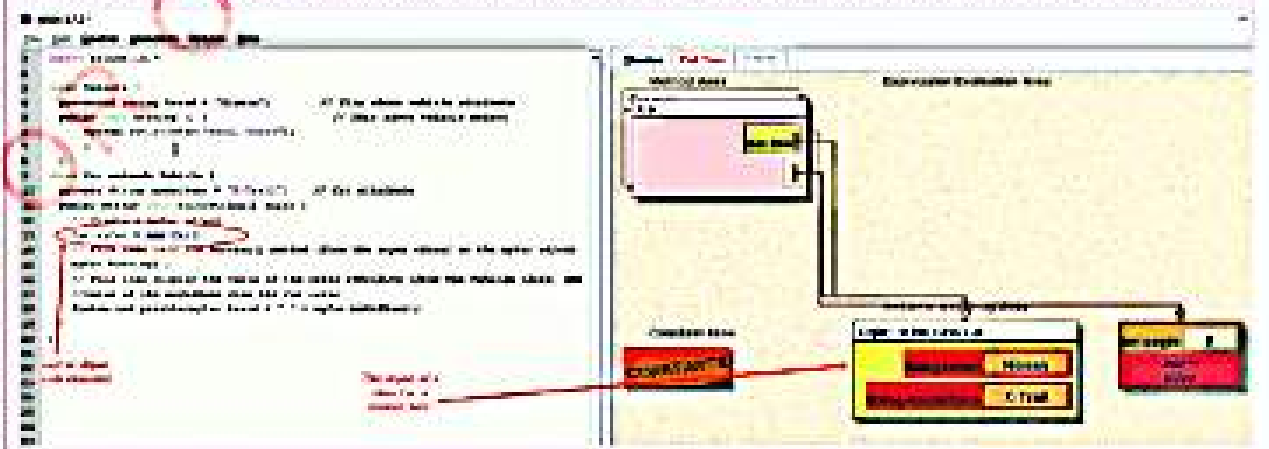


Figure 5.25: *Jeliot program running codes*

Step 4: Observe the output through the console and close the compiler, the out put is shown on previous pages in Figure 5.22.

FOR ONLINE READING ONLY

(c) Multiple inheritance in Java

Multiple inheritance is the ability of a programming language to enable a class to inherit characteristics from more than one parent class. However, in Java, direct support for multiple inheritance of classes is not provided. Instead, Java allows for multiple inheritance using an interfaces, which offers certain advantages of multiple inheritance without the associated complexities. In Java, a class can implement multiple interfaces, each of which specifies a set of methods that the class must implement. By implementing multiple interfaces, a class can inherit behaviors from each interface while still adhering to a single inheritance hierarchy.

Java Interfaces

An interface in Java is a blueprint for a group of related methods without providing implementations. In Java, interfaces establish an arrangement for classes by defining a series of method declarations without implementing them. Classes implement interfaces by providing actual implementations for the methods defined in the interface.

(a) Interface Declaration

An interface is declared using the "interface" keyword followed by the interface name. This can be presented as:

```
interface Animal {
   void eat();  //Interface method does not have a body
   void sleep(); // Interface method does not have a body
}
```

(b) Implementing an interface

To implement an interface, a class uses the "implements" keyword followed by the interface name. To utilise the methods defined in an interface, another class must "implement" the interface using the implements keyword instead of extends. The body of the interface method is provided by the "implement" class, as shown in Program Example 5.24.

Program Example 5.24:

Java program for interface

```
// Interface
interface Bird {
  public void birderies(); // interface method (does not have a body)
```

FOR ONLINE READING ONLY

```
  public void verticalmotion(); // interface method (does not have a body)
}
// Hen "implements" the Bird interface
class Hen implements Bird {
  public void birderies() {
    // The body of birderies() is provided here
    System.out.println("The hen speak: kooo..kokoko...kookoko");
  }
  public void verticalmotion() {
    // The body of verticalmotion() is provided here
    System.out.println("fyu!fyu!fyu!fyu!fyuuuuu");
  }
}
class Main {
  public static void main(String[] args) {
    Hen myHen = new Hen();  // This create a Hen object
    myHen.birderies();
    myHen.verticalmotion();
  }
}
```

Output:

```
The hen speak: kooo..kokoko...kookoko
fyu!fyu!fyu!fyu!fyuuuuu
```

Figure 5.26: *Output of interface program*

(e) Multiple inheritance using interface

In Java, multiple inheritance can be achieved using interfaces. Unlike classes, Java permits a class to implement multiple interfaces. This allows the class to inherit method signatures from multiple interfaces, enabling it to exhibit behavior from each interface it implements. This approach avoids the conflicts that can arise with multiple inheritance in class-based systems, known as the diamond problem. In order to implement multiple interfaces, use a comma to separate them. Program Example 5.25 shows a multiple interface.

FOR ONLINE READING ONLY

Program Example 5.25:

Java program to demonstrate multiple interface

```
Java program to demonstrate multiple interface
interface One {
  public void methodA(); // This is interface method
}
interface Two {
  public void methodB(); // This is interface method
}
class OnRoadTest implements One, Two {
  public void methodA() {
    System.out.println("We test the movie of interface..");
  }
  public void methodB() {
    System.out.println("We are to make object from this interface...");
  }
}
class Main {
  public static void main(String[] args) {
    OnRoadTest makeObj = new OnRoadTest ();
    makeObj.methodA();
    makeObj.methodB();
  }
}
```

Output:

```
We test the movie of interface..
We are to make object from this interface...
```

Figure 5.27: *Output of multiple interface program*

FOR ONLINE READING ONLY

Activity 5.9:

Visualisation of Java interface code execution

Use Jeliot 3 or later version, visualise the step by step execution of java program to explore about interface in Java. Use the code in Program Example 5.25.

Exercise 5.4:

1. Explain the concept of inheritance in Java and how it allows classes to inherit properties and behaviors from other classes. Provide an example of inheritance in action.
2. Describe the difference between single inheritance and multiple inheritance in Java. How does Java

handle multiple inheritance, and why is it achieved differently than in other programming languages?

3. How do interfaces provide a way to achieve multiple inheritance in Java? Explain the role of interfaces in defining contracts for classes and how classes implement interfaces to fulfill those contracts.
4. Discuss the diamond problem in the context of multiple inheritance and explain how Java's interface approach avoids this issue. Provide an example to illustrate how interfaces help resolve the diamond problem.
5. Describe the concept of abstract classes and how they relate to inheritance and interfaces in Java. Explain when it's appropriate to use abstract classes versus interfaces in your Java programs.
6. Discuss the importance of polymorphism in Object-Oriented programming and how inheritance and interfaces contribute to achieving polymorphic behavior in Java. Provide examples to demonstrate polymorphism in action using inheritance and interfaces.

Abstraction in Java

Abstract classes

Abstract classes are classes that cannot be instantiated and may contain abstract methods. They serve as blueprints for other classes and can define common behavior for subclasses. In other words, abstract class is a restricted class that cannot be instantiated directly, but can only be accessed by inheriting it from another class.

Abstract method

An abstract method is a method defined in an abstract class. It serves as a placeholder for methods that must be implemented by subclasses. It can only be used in an abstract class, and it does not have a body. The body is provided by the subclass. Abstract classes are declared using the "abstract" keyword. They can have both abstract and concrete methods. Abstract classes cannot be instantiated but can be subclassed. Subclasses of abstract classes must implement all abstract methods.

Program Example 5.26:

Java program to show abstract class

```
// Abstract class
abstract class Animal {
  // Abstract method (does not have a body)
  public abstract void animalSound();
  // Regular method
  public void sleep() {
    System.out.println("Zzz");
  }
}
// Subclass (inherit from Animal)
class Pig extends Animal {
  public void animalSound() {
```

FOR ONLINE READING ONLY

```
    // The body of animalSound() is provided here
    System.out.println("The pig says: wee wee");
  }
}
class Main {
  public static void main(String[] args) {
    Pig myPig = new Pig(); // Create a Pig object
    myPig.animalSound();
    myPig.sleep();
  }
}
```

Output:

```
The pig says: wee wee
Zzz.
```

Figure 5.28: *Output of abstract program*

Activity 5.10:

Visualisation of code execution in Java abstraction

Use Jeliot 3 or later version, visualise the step by step execution of java program to explore about abstraction in Java. Use the code in Program Example 5.27.

Exercise 5.5

1. Describe how abstraction is implemented in Java through abstract classes and interfaces. Provide examples of each.
2. Discuss the role of abstract methods in abstract classes and interfaces. How do they contribute to achieving abstraction?
3. Describe a real-world scenario where abstraction would be beneficial in Java programming. How would you implement abstraction to solve the problem?
4. Explain the concept of data hiding and how it relates to abstraction in Java. Provide examples to illustrate your explanation.
5. Design a Java program that utilizes abstraction to model a banking system. Include abstract classes or interfaces to represent different account types, such as savings and checking accounts, and demonstrate how abstraction simplifies the implementation and maintenance of the system.

Encapsulation in Java

Encapsulation is a fundamental concept in object-oriented programming (OOP) that involves bundling data (attributes) and methods (functions) that operate on the data into a single unit, called a class. The significance of encapsulation is that it facilitates data hiding, abstraction, and code organisation, using access modifiers, leading to modular and maintainable code.

Access modifiers and use of "this" and "final" keywords in Java

Access modifiers have a significant impact on Java programming as they govern the visibility and accessibility of class members. Their primary function is to ensure appropriate encapsulation and data

FOR ONLINE READING ONLY

hiding within classes. Below is an in-depth explanation of the frequently employed access modifiers.

(a) Public:

The public access modifier grants unrestricted access to a class member from any class, regardless of the package it belongs to. The following is a syntax for public access modifier:

```
public class MyClass {
    public int classAttributes;
}
```

The access modifier "public" is used for class members that need to be accessed from external classes or packages.

(b) Private:

The private access modifier limits access to a class member exclusively within its own class. It hides the implementation details of the class and guarantees the integrity of the data. The following is a syntax for private access modifier

```
public class MyClass {
    private int classAttributes;
}
```

The access modifier "private" is used for class members that are not supposed to be accessed directly from outside the class. They should be accessed through getter and setter methods for controlled access.

(c) Protected:

The protected access modifier provides access to a class member within its own package and subclasses, regardless of the package they belong to. The following is a syntax for protected access modifier:

```
public class MyClass {
    protected int classAttribute;
}
```

The "protected" access modifier is utilised for class members that require access within the same package or by subclasses. It allows for controlled sharing of data among interconnected classes.

Generally, access modifiers play a crucial role in upholding encapsulation principles in Java classes. They serve to prevent unauthorised access to class members and facilitate data hiding, thereby improving the security and maintainability of the codebase.

(d) This keyword

"This" keyword is used depending on various situations. The following are some situations where the keyword is used:

(i) Used in ambiguous variable name

In Java, it is not permissible to declare two or more variables with identical names within a scope, be it class scope or method scope. However, instance variables and parameters can share the same name. The following is a piece of Java code showing ambiguous variable names.

FOR ONLINE READING ONLY

```
class MyClass {
    // lines show instance variable
    int length;
    // lines show parameter
    MyClass(int length){
        length = length;
    }
}
```

In the piece of code provided, both the instance variable and the parameter are named "length." This creates ambiguity for the Java compiler. To resolve such ambiguity, you can use the "this" keyword. When you write and run the above codes without this keyword you may get ambiguous output such as zero or other. Program Example 5.27 shows the use of this keyword, the output is shown in Figure 5.29.

Programe Exampe 5.27:

Java code using this keyword for ambiguous variable names

```
class MyClass {
    // lines show instance variable
    int length;
    // lines show parameter
    MyClass(int length){
        this.length = length;
    }
    public static void main(String[] args) {
        MyClass obj = new MyClass(23);
        System.out.println("obj.length = " + obj.length);
    }
}
```

Output:

```
obj.length = 23
```

Figure 5.29: *Output of ambiguous variable*

(ii) Used in getters and setters methods

In these methods, this keyword is used to assign value inside the setter method and also to access value inside the getter method as shown in Program Example 5.28.

FOR ONLINE READING ONLY

Programe Example 5.28:

Java code using this keyword for getters and setters

```
class MyClass {
   String word;
   // setter method
   void setWord( String word ) {
      this.word = word;
   }
   // getter method
   String getWord(){
      return this.word;
   }
   public static void main( String[] args ) {
      MyClass obj = new MyClass();
      // calling the setter and the getter method
      obj.setWord("Usimbaji Mubashara");
      System.out.println("obj.word: "+obj.getWord());
   }
}
```

Output:

```
obj.word: Usimbaji Mubashara
```

Figure 5.30: *Output of "this" keyword used with getters and setters*

(iii) Used in constructor overloading

When dealing with constructor overloading, there may be instances where you need to call one constructor from another. However, it is important to note that you cannot directly invoke a constructor. Instead, you can employ the "this" keyword for this purpose. Program Example 5.29 elaborates.

Program Example 5.29:

Java program to use this keyword for constructors

```
class BigClass {
   //code shows instance variables
   int age;
   String name;
   // code shows parameterized constructor
```

FOR ONLINE READING ONLY

```
    BigClass(int age) {
        this.age = age;
    }
    // Code shows overloaded constructor
    BigClass(int age, String name) {
        // Code invokes the parameterised constructor using "this"
        this(age);
        this.name = name;
    }
    // method to show information
    void show() {
        System.out.println("Name: " + name + ", Age: " + age);
    }
    public static void main(String[] args) {
        // creating objects of BigClass
        BigClass objX = new BigClass(35600);
        BigClass objY= new BigClass(600, "Malaria");

        // calling show method to present information
        objX.show();
        objY.show();
    }
}
```

Output:

```
Name: null, Age: 35600
Name: Malaria, Age: 600
```

Figure 5.31: *Output of "this" keyword in constructors*

The first constructor accepts an age parameter, while the second constructor takes both age and name parameters. To avoid repeating the initialisation logic, the second constructor uses the "this" keyword to invoke the first constructor and pass the age parameter. By doing this, you can eliminate the need for redundant code duplication.

(iv) Used for passing current object as argument

By using the "this" keyword, it is possible to pass the current instance of an object as an argument to another method. This is demonstrated in Program Example 5.30.

FOR ONLINE READING ONLY

Program Example 5.30:

Java program to use this keyword for pasing current object

```
class MyClass {
    int age;
    //code shows  parameterised constructor
    MyClass(int age) {
        this.age = age;
    }
    //Code shows a method to display age
    void displayAge() {
        System.out.println("Age: " + this.age);
    }
    // method to pass a current object as argument
    void myObject() {
        methodTwo(this);
    }
    // method to receive the current object as an argument
    void methodTwo(MyClass obj) {
        System.out.print("See age from object: ");
        obj.displayAge();
    }
    public static void main(String[] args) {
        // creating an object of MyClass
        MyClass obj = new MyClass(5700);
        // code invoke myObject method
        obj. myObject();
    }
}
```

Output:

```
See age from object: Age: 5700
```

Figure 5.32: *Output showing "this" keyword passing objects*

The myObject() method invokes methodTwo() and provides the current object (referred to as "this") as an argument. Inside methodTwo(), you can receive the current object as an argument and proceed to call the displayAge() method, which prints the age of an object.

FOR ONLINE READING ONLY

(e) Final keyword

In the Java programming language, the "final" keyword is employed to indicate constants. It can be applied to variables, methods, and classes. When any of these entities is declared as final, it can only be assigned a value once. This means that a final variable cannot be reassigned with a different value. In this case, also a final method cannot be overridden by a subclass. As for variable and method, a final class cannot be extended by another class.

(i) Final variable

In Java the value of a final variable cannot be changed, as shown in Program Example 5.31.

Program Example 5.31:

Java program demonstrating final variable of a final keyword

```
class MyClass {
  public static void main(String[] args) {
    //this code show a final variable
    final int age = 93;

    // this code want to change the final variable
    age = 39;
    System.out.println("Age: " + age);
  }
}
```

Output:

```
Execution Error

This object cannot be modified. Its fields can be private or protected.
```

Figure 5.33: *Output "final" keyword program*

(ii) Final method

Program Example 5.32 demonstrate how the final method cannot be overridden. In this case, the final method in parent class cannot be overridden by a child class. In this example, a final method named show() is created inside the class FinalClass. The Main class inherits the 'FinalClass' class. When you try to override the final method in the Main class you might end up with compilation error.

FOR ONLINE READING ONLY

Program Example 5.32:

Java program to demonstrate final keyword to final method

```
class FinalClass {
    // code that creates a final method
    public final void show() {
        System.out.println("This is a final method.");
    }
}

class Main extends FinalClass {
    // Uncommenting the following method will cause a compilation error
    // because you cannot override a final method
    /*
    public final void show() {
        System.out.println("Trying to override final method???.");
    }
    */
    public static void main(String[] args) {
        Main myObj = new Main();
        myObj.show();  // This will call the show() method from FinalClass
    }
}
```

Output:

```
This is a final method.
```

Figure 5.34: *Output demonstrate final keyword to final method*

(iii) Final class

In java, you can declare the class as final in order to restrict other classes to inherit the parent class. The final class cannot be inherited by another class, as shown in Program Example 5.33.

Program Example 5.33:

Java program to demonstrate final class

```
// This code creates a final class
final class FinalClass {
  public void show() {
    System.out.println("This shows a final method.");
  }
}
```

FOR ONLINE READING ONLY

```
// This code is trying to extend the final class
class Main extends FinalClass {  // This will cause a compilation error
  public void show() {
    System.out.println("final method is overridden?!?!?.");
  }
  public static void main(String[] args) {
    Main newObj = new Main();
    newObj.show();
  }
}
```

Output:

```
Main.java:9: error: cannot inherit from final FinalClass
class Main extends FinalClass {  // This will cause a compilation error
                   ^
1 error
```

Figure 5.35: *Output for demonstrating final class*

Exercise 5.6

1. Explain the difference between public, private, and protected access modifiers in Java. Give examples of when each would be used.
2. How does the "this" keyword work in Java? Provide examples to illustrate its usage in different contexts.
3. Describe the purpose of the "final" keyword in Java. How does it affect variables, methods, and classes? Give examples to demonstrate its usage.
4. Can you use the "final" keyword with local variables in Java? Explain with examples.
5. What happens if you try to access a private member variable of a class directly from another class? How can you access it using the "this" keyword? Provide an example.

Encapsulation and Java classes

Encapsulation in Java is achieved by declaring class attributes as private and providing public getter and setter methods to access and modify the attributes.

Program Example 5.34:

Java program to demonstrate encapsulation

```
public class MyClass {
public class Car {
    private String model;
    private int year;
    // This is Getter method
    public String getModel() {
        return model;
    }
    // This is Setter method
    public void setModel(String model) {
        this.model = moxlel;
    }
```

FOR ONLINE READING ONLY

```
    // This is Getter method
    public int getYear() {
        return year;
    }
    // This is Setter method
    public void setYear(int year) {
        this.year = year;
    }
}
```

Getter and setter methods: Getter methods are responsible for fetching the values of private attributes, whereas setter methods are used to modify the attribute values. These methods facilitate controlled access to class data and uphold the principle of data encapsulation. As explained above, that private variables can only be accessed within the same class (an outside class cannot access it). However, it can be accessed if you provide the public "get" and "set" methods. In this case, the get method is used to return the value of the variable while the set method is used to set the value, as shown in Program Example 5.38.

Getter and Setter syntax: They start with get or set, followed by variable name. The first letter of the variable name should be written in upper case.

Activity 5.11:

Visualisation of encapsulation in Java

Use Jeliot 3 or later version, visualise the step by step execution of java program to explore about encapsulation in Java. Use the code in Program Example 5.37.

Program Example 5.35:

Java program to demonstrate get and set methods

```
public class Animal {
  private String name; // This shows restricted access

  // Getter
  public String getName() {
    return name;
  }

  // Setter
  public void setName(String newName) {
    this.name = newName;
  }
}
```

FOR ONLINE READING ONLY

From this program example, the "get" method returns the value of the variable name. The "set" method takes a parameter "newName" and assigns it to the name variable. The keyword "this" is used to refer to the current object. In this case, the name variable is declared as private, it does not grant permission to be accessed from outside this class.

Exercise 5.7

1. Explain the concept of encapsulation in Java and why it is important in software development.
2. Compare and contrast the public, private, and protected access modifiers in Java, highlighting their roles in encapsulation.
3. How does encapsulation contribute to the security and integrity of Java code? Provide examples to illustrate your answer.
4. Describe the steps involved in achieving encapsulation in a Java class. Include the declaration of attributes, access modifiers, and the use of getter and setter methods.
5. Discuss the advantages of using getter and setter methods to access private attributes in Java classes. Provide real-world scenarios where encapsulation through getter and setter methods would be beneficial.
6. Explain the potential risks of exposing class attributes as public fields without encapsulation. How does encapsulation mitigate these risks?
7. Design a Java class representing a bank account with private attributes for account number, balance, and customer name. Implement getter and setter methods for accessing and modifying these attributes, ensuring encapsulation principles are followed.

Polymorphism in Java

Polymorphism is a core principle in object-oriented programming, enabling objects to exhibit varying forms or behaviors depending on the situation. In Java, polymorphism is realized through inheritance, method overriding, and method overloading.

Polymorphism allows for the treatment of objects belonging to different classes as if they are objects of a shared superclass. This concept enhances flexibility and extensibility in software design. For instance, imagine a superclass called "Shape" with subclasses such as "Circle," "Rectangle," and "Triangle." While these subclasses inherit common characteristics from the "Shape" superclass, they also have their own unique implementations for calculating area and perimeter.

Inheritance as Polymorphism in Java

Inheritance involves a class acquiring the attributes and actions of another class. Polymorphism, closely associated with inheritance, permits subclasses to demonstrate distinct behaviors while still sharing core features with

FOR ONLINE READING ONLY

the superclass. Subclasses can redefine (override) inherited methods to offer their unique implementations, facilitating polymorphic behavior.

Program Example 5.36:

Java program to demonstrate polymorphism

```
class Animal {
    public void makeSound() {
        System.out.println("Animal makes different sounds");
    }
}
class Lion extends Animal {
    //This method override
    public void makeSound() {
        System.out.println(" Lion roars");
    }
}
class Cat extends Animal {
    //This method override
    public void makeSound() {
        System.out.println("Cat meows");
    }
}
public class Main {
    public static void main(String[] args) {
        Animal myFirstAnimal = new Lion();
        Animal mySecondAnimal = new Cat();

        myFirstAnimal.makeSound(); // This method will print "Lion roars"
        mySecondAnimal.makeSound(); // This method will print "Cat meows"
    }
}
```

In Java, polymorphism is achieved through inheritance, method overriding, and method overloading.

Activity 5.12:

Visualisation of polymorphism in Java

Use Jeliot 3 or later version, visualise the step by step execution of java program to explore about polymorphism in Java. Use the code in Program Example 5.39.

FOR ONLINE READING ONLY

Method overriding

Method overriding takes place when a subclass provides a different implementation of a method that is already present in its parent class. It allows a subclass to customize the behavior of inherited methods, as it is shown in Program Example 5.37.

Program Example 5.37:

Java program to demonstrate method overriding

```
class MotorMachine {
    public void startEngine() {
        System.out.println("MotorMachine starts");
    }
}
class Car extends MotorMachine {
    // This code try to override
    public void startEngine() {
        System.out.println("Car starts engine using a key");
    }
}
class Motorcycle extends MotorMachine {
    //This code try to override
    public void startEngine() {
        System.out.println("Motorcycle starts engine using a kick");
    }
}
public class Main {
    public static void main(String[] args) {
        MotorMachine machineA = new Car();
        MotorMachine machineB = new Motorcycle();

        machineA.startEngine(); // Output: Car starts engine using a key
        machineB.startEngine(); // Motorcycle starts engine using a kick
    }
}
```

In this particular illustration, there exists a parent class called MotorMachine that possesses a method named StartEngine(). The Car and Motorcycle classes are derived from the MotorMachine class, and they each provide their own versions of the StartEngine() method by overriding it. When we invoke the StartEngine() method

FOR ONLINE READING ONLY

on instances of the Car and Motorcycle classes, the overridden methods unique to each subclass are executed. This showcases the concept of method overriding, which is a form of run-time polymorphism.

Method overloading

Method overloading permits several methods to share the same name while possessing distinct parameter lists. It offers a means to define methods that execute similar tasks but with varying input types or counts of parameters. The compiler selects the appropriate method for invocation based on the arguments supplied to it. Program Example 5.38 illustrates the concept.

Program Example 5.38:

Java program to demonstrate method overloading

```
public class ArithmeticOperations {
    // Method to add two integers
    public int add(int a, int b) {
        return a + b;
    }
    // Method to add three integers
    public int add(int a, int b, int c) {
        return a + b + c;
    }
    // Method to add two doubles
    public double add(double a, double b) {
        return a + b;
    }
    // Method to concatenate two strings
    public String add(String a, String b) {
        return a + b;
    }
    // Method to multiply two integers
    public int multiply(int a, int b) {
        return a * b;
    }
```

FOR ONLINE READING ONLY

```
    // Method to multiply a double and an integer
    public double multiply(double a, int b) {
        return a * b;
    }
    public static void main(String[] args) {
        ArithmeticOperations arithmetic = new ArithmeticOperations();
        // Method overloading for addition
        System.out.println("Sum of 150 and 12: " + arithmetic.add(150, 12));
        System.out.println("Sum of 11, 12, and 23: " + arithmetic.add(11,12,23));
        System.out.println("Sum of 7.5 and 9.3: " + arithmetic.add(7.5, 9.3));
        System.out.println("Concatenated string: " + arithmetic.add("Habari rafiki",
"Tanzania"));
        // Method overloading for multiplication
        System.out.println("Product of 11 and 22: " + arithmetic.multiply(11, 22));
        System.out.println("Product of 10.2 and 8: " + arithmetic.multiply(10.2, 8));
    }
}
```

Output:

In this program, there are several variations of the add and multiply methods, each with different types of parameters. When these methods are called, Java determines the correct method to execute based on the number and types of arguments provided. This illustrates method overloading, which is a type of compile-time polymorphism.

FOR ONLINE READING ONLY

Exercise 5.8

1. Define polymorphism in Java and explain how it contributes to the flexibility and extensibility of object-oriented programs.
2. Describe the two types of polymorphism in Java: compile-time (method overloading) and runtime (method overriding). Provide examples of each.
3. Explain the concept of method overloading in Java. How does it enable polymorphism? Provide a scenario where method overloading would be useful.
4. Discuss the significance of method overriding in Java polymorphism. How does it allow subclasses to provide their own implementation of inherited methods?

5. Describe how polymorphism is achieved through inheritance and method overriding in Java. Provide a practical example to illustrate the concept.
6. Explain the role of dynamic method dispatch in achieving runtime polymorphism in Java. How does it enable method invocation based on the actual object type?
7. Design a Java program that demonstrates both compile-time and runtime polymorphism. Include examples of method overloading and method overriding to showcase the flexibility of polymorphic behavior.
8. How does the Java compiler determine which overloaded method to call during compile-time polymorphism?

Building an application with Python

When building any program in this section, there are several stages to undergo. The following are three major stages:

Stage One: ***Planning the project***

(1) Define objectives

(a) **Purpose:** Clearly articulate what the application aims to achieve.

(b) **Goals:** Specify the desired outcomes, such as improving efficiency or solving a particular problem.

(2) Identify requirements

(a) **Features:** List all functionalities the application should have.

i. **Example:** User authentication, data input, reporting.

(b) **Data inputs:** Identify the types of data the application will handle.

i. **Example:** User details, transaction data.

(c) **Outputs:** Define what the application will produce.

Example: Reports, notifications.

(3) Create a design document

(a) **Outline classes:** List all classes with a brief description.

(b) **Methods and relationships:**

(i) Detail methods for each class.

(ii) Describe how classes interact.

Stage Two: ***Designing classes and relationships***

(1) Class diagrams

(i) **Purpose:** Provide a visual representation of classes and their interactions.

(ii) **Tools:** Use diagramming software like UML or simple drawings.

(2) Key considerations

(a) **Encapsulation:**

(i) Keep data private within classes.

(ii) Provide public methods to access and modify data safely.

FOR ONLINE READING ONLY

(b) Inheritance:

(i) Use inheritance to avoid code duplication.

(ii) Example: classSavings Account(BankAccount).

(c) Composition:

(i) Combine simple objects to create complex ones.

(ii) Example: A User class containing multiple BankAccount objects.

Stage Three: ***Implementing the project***

1. Coding

Write code:

(i) Implement classes and methods as outlined in the design document.

(ii) Follow coding standards for readability and maintainability.

2. Modular development

(a) **Build and test:** Develop each module separately, ensuring it functions correctly before integrating.

(b) **Advantages:**

(i) Easier debugging and testing.

(ii) Enhances code reusability.

Stage four: ***Testing and debugging***

1. Testing

(a) Unit tests:

(i) Test individual functions or methods.

(ii) Use frameworks like unittest or pytest.

(b) **Integration tests:**

(i) Test the interaction between different modules.

(ii) Ensure that components work together as intended.

2. Debugging

(a) **Print statements:** Use for quick checks on variable values and program flow.

(b) **Debugging tools:**

(i) **pdb:** Python's built-in debugger for step-by-step execution.

(ii) **IDEs:** Use features in IDEs like PyCharm or VSCode for setting breakpoints.

3. Common techniques

(a) **Syntax errors:** Carefully review error messages and correct typos.

(b) **Logic and control flow:** Ensure the program logic is sound and flow is logical.

(c) **Exception handling:** Gracefully handle potential errors using try-except blocks.

Example: Imagine you are a programmer and want to develop an application Library management systems. Use the described stages to show how you can undergo.

FOR ONLINE READING ONLY

Solution:

Stage One: ***Planning the project***

1. Define objectives

(a) Purpose: Develop a system to manage library operations, including book checkouts, returns, and inventory.

(b) Goals:

(i) Allow users to search for books.

(ii) Manage book checkouts and returns.

(iii) Track inventory and user accounts.

2. Identify requirements

(a) Features:

(i) User authentication.

(ii) Search and browse books.

(iii) Checkout and return books.

(iv) Manage user accounts and book inventory.

(b) Data inputs:

(i) Book details (title, author, ISBN, availability).

(ii) User details (name, ID, checked-out books).

(c) Outputs:

(i) User-friendly interface for library staff and users.

(ii) Reports on book availability and user activities.

3. Create a design document

(a) Outline classes

(i) Class: Book

Attributes: title, author, ISBN, available_copies.

Methods: checkout(), return_book().

(ii) Class: User

Attributes: name, user_id, checked_out_books.

Methods: checkout_book(), return_book().

(iii) Class: Library

Attributes: books, users.

Methods: add_book(), remove_book(), search_book().

Stage Two: ***Designing classes and relationships***

1. Class diagrams

Visual representation:

(i) Library contains multiple Book and User objects.

(ii) User interacts with Book through checkout() and return_book() methods.

2. Key considerations

(a) Encapsulation:

(i) Keep book details private within the Book class.

(ii) Manage user details through the User class methods.

(b) Inheritance:

Not used in this simple scenario, but could be expanded (e.g., different types of users).

(c) Composition:

Library class composed of lists of "Book" and "User" objects.

FOR ONLINE READING ONLY

Stage Three: ***Implementing the project***

1. Coding

Book Class:

```
class Book:
    def __init__(self, title, author, isbn, available_copies):
        self.title = title
        self.author = author
        self.isbn = isbn
        self.available_copies = available_copies

    def checkout(self):
        if self.available_copies > 0:
            self.available_copies -= 1
            return True
        return False

    def return_book(self):
        self.available_copies += 1
```

User Class:

```
class User:
    def __init__(self, name, user_id):
        self.name = name
        self.user_id = user_id
        self.checked_out_books = []

    def checkout_book(self, book):
        if book.checkout():
            self.checked_out_books.append(book)
            print(f"{book.title} checked out successfully.")
        else:
            print(f"{book.title} is not available.")

    def return_book(self, book):
        if book in self.checked_out_books:
            book.return_book()
```

FOR ONLINE READING ONLY

```
            self.checked_out_books.remove(book)
            print(f"{book.title} returned successfully.")
        else:
            print("Book not found in user's checked out list.")
```

Library Class:

```
class Library:
    def __init__(self):
        self.books = []
        self.users = []

    def add_book(self, book):
        self.books.append(book)

    def remove_book(self, book):
        self.books.remove(book)

    def search_book(self, title):
        return [book for book in self.books if title.lower() in book.title.lower()]

    def add_user(self, user):
        self.users.append(user)
```

2. Modular development

(a) Develop each class separately and test its methods.

(b) Example of testing the Book class:

```
book1 = Book("Python Programming", "Sarai Mlavangwa", "1234567890", 3)
print(book1.checkout())  # Output: True
print(book1.available_copies)  # Output: 2
```

Stage four: ***Testing and debugging***

1. Testing

(a) Unit tests:

(i) Test individual methods for expected functionality.

FOR ONLINE READING ONLY

(ii) Example using 'unittest':

```
import unittest
class TestLibrarySystem(unittest.TestCase):
    def test_book_checkout(self):
        book = Book("Test Book", "Author", "12345", 2)
        self.assertTrue(book.checkout())
        self.assertEqual(book.available_copies, 1)

    def test_user_checkout_book(self):
        book = Book("Test Book", "Author", "12345", 1)
        user = User("Alice", "001")
        user.checkout_book(book)
        self.assertIn(book, user.checked_out_books)

if __name__ == '__main__':
    unittest.main()
```

(b) Integration tests:

Ensure the 'Library', 'Book', and 'User' classes work together smoothly.

```
library = Library()
book1 = Book("Python Programming", "Sanzi Mkuwangwa", "1234567890", 3)
user1 = User("Alice", "001")

library.add_book(book1)
library.add_user(user1)

user1.checkout_book(book1)
user1.return_book(book1)
```

2. Debugging

(a) **Print statements:** Add print statements within methods to track flow and variable states.

(b) **Debugging Tools:**

(i) Use pdb for step-by-step debugging.

(ii) Set breakpoints in your IDE to monitor execution.

FOR ONLINE READING ONLY

3. Common techniques

(a) **Syntax errors:** Correctly read error messages for quick fixes.

(b) **Logic and control flow:** Ensure program logic aligns with the project's goals.

(c) **Exception handling:** Use try-except blocks to manage potential errors, such as book not available.

```
try:
    user1.checkout_book(book1)
except Exception as e:
    print(f"An error occurred: {e}")
```

Question: Assume you are the programmer. Design any system relevant to your environment, that will use the stages for building an application

Chapter Summary

Reflect on what you have learnt from this chapter, summarise and keep it in your portfolio.

Revision Exercise 5

1. Create a Java class named "Car" with attributes such as make, model, and year. Define methods to set and retrieve these attributes. Instantiate multiple Car objects and display their details.
2. Implement a Java program to calculate the area of different shapes (circle, rectangle, triangle) using methods for each shape. Use abstraction to hide the implementation details of the area calculation.
3. Develop a Java application for a banking system with classes like Account, SavingsAccount, and CheckingAccount. Implement inheritance to create specialized account types and encapsulation to protect sensitive account information.
4. Write a Java program to demonstrate method overloading by creating multiple methods with the same name but different parameter lists, such as calculating the area of a square, rectangle, and circle.
5. Create a Java program to model a simple library system with classes like Book, Author, and Library. Use inheritance to represent relationships between different

FOR ONLINE READING ONLY

entities and encapsulation to protect book information.

6. Implement a Java program to demonstrate polymorphism by creating a base class "Shape" with methods like draw() and calculateArea(). Then, create subclasses like Circle, Rectangle, and Triangle to override these methods with specific implementations.
7. Develop a Java application to simulate a shopping cart system with classes like Product, ShoppingCart, and Customer. Use encapsulation to protect the cart's contents and inheritance to represent different types of customers.
8. Write a Java program to perform basic arithmetic operations (addition, subtraction, multiplication, division) using methods and operators. Use variables to store operands and results.
9. Implement a Java program to define and use constants for mathematical constants like Pi (π) and Euler's number (e). Use these constants in calculations throughout the program.
10. Create a Java application to model a school system with classes like Student, Teacher, and Course. Use constructors to initialize object states and demonstrate method overriding in subclasses.
11. Develop a Java program to manage employee information with classes like Employee, Manager, and Intern. Use inheritance to represent different employee roles and encapsulation to protect sensitive employee data.
12. Write a Java application to model a simple game with classes like Player, Enemy, and Weapon. Demonstrate polymorphism by allowing different types of players and enemies to interact with each other.
13. Implement a Java program to handle exceptions in file I/O operations. Use try-catch blocks to handle potential errors like file not found or permission denied.
14. Create a Java program to demonstrate operator precedence by performing complex arithmetic expressions involving addition, subtraction, multiplication, and division.
15. Develop a Java application to implement a basic calculator with methods for arithmetic operations like addition, subtraction, multiplication, and division. Use method overloading to handle different parameter types and numbers of operands.
16. Explore the following plain text of OOP codes, then answer the questions that follows:

FOR ONLINE READING ONLY

```
class Car:
    private String make;
    private String model;
    private int year;
    private boolean running;

    public Car (String make, String model, int year) {
        this.make = make;  // Encapsulated attribute
        this.model = model;  // Encapsulated attribute
        this.year = year;  // Encapsulated attribute
        this.running = false;  // Encapsulated attribute
    }

    public void startEngine() {
        this.running = true;  // Setter method
    }
    public void stopEngine() {
        this.running = false;  // Setter method
    }

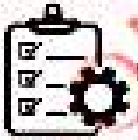
    public String getMake() {
        return this.make;  // Getter method
    }
```

i. What do you understand by Setter method and Getter method from codes above.

ii. Explain why are the attributes in the class labelled private.

iii. In the provided code examples, how is encapsulation implemented in the Car class?

iv. What are the advantages of encapsulating data and methods within a class, as demonstrated in the code?



Project: *Exploring Garbage Collection in Java*

Project overview: Use this project, investigate the inner workings of Java's garbage collection mechanism. By creating and destroying a significant number of objects, observe how the finalize() method is invoked just before an object is garbage collected.

Materials required:

Computer with Java Development Kit (JDK) installed, Integrated Development Environment (IDE) such as IntelliJ IDEA, Eclipse, or any

Deliverables:

A detailed report including screenshots of the running java garbage collection mechanism program.

FOR ONLINE READING ONLY



Object Oriented Programming with Python

Introduction

Python is a dynamic and versatile programming language widely used for developing various software applications. In this chapter, you will delve into the world of Python programming language. You will explore its syntax and learn the fundamental constructs required to create object-oriented programs. Additionally, you will learn about debugging object-oriented programs using Python. The competencies developed will enable you to solve complex problems and enhance your ability to collaborate effectively within coding teams. Furthermore, you will be able to perform code maintenance and debugging.

Think:

How Python's OOP powers can transform challenging tasks

Concepts of Object Oriented Programming with Python

Scenario 6.1: *Python-powered camera*

Imagine you love taking pictures of wild animals in their natural homes. To capture the best shots, you use special cameras that are powered by Python. These cameras can track and recognise even the rarest animals using smart technology. With Python's help, you can document these incredible creatures, learn more about their behavior, and help protect them. By using Python, you can take amazing photos that show the beauty of nature and make people care about conserving our wildlife.

Question:

How does Python-driven camera technology aid in tracking and identifying rare animals in wildlife photography? Provide a concise explanation.

FOR ONLINE READING ONLY

Overview of Python

Python is an interpreted, high-level programming language known for its simplicity, readability, and versatility. It prioritises code readability and expressiveness, using a clean and concise syntax. Python supports multiple programming paradigms, including procedural, object-oriented, and functional programming.

Features:

(a) *Readability:* Python has a clear and intuitive syntax, making it easy to learn and understand.

(b) *Dynamically typed*: Python allows variables to be dynamically typed, enabling flexible and concise code.

(c) *Extensive standard library*: Python comes with a large collection of modules and packages for various tasks, reducing the need for external dependencies.

(d) *Interpreted language:* Python code is executed line by line by the Python interpreter, facilitating rapid development and prototyping.

Applications:

(a) *Web development:* Python offers frameworks like Django and Flask for building web applications and APIs.

(b) *Data science and machine learning:* Python is used with libraries such as NumPy, Pandas, and TensorFlow for data analysis and machine learning.

(c) *Scripting and automation:* Python is commonly used for writing scripts to automate tasks, perform system administration, and conduct software testing.

(d) *Education:* Python is widely used in academic institutions to teach programming and computer science concepts.

Introduction to python OOP

Python is a programming language that is highly regarded for its simplicity and readability. It is an interpreted language, meaning that it does not require compilation before running. Created by Guido van Rossum and introduced in 1991, Python has gained popularity in multiple fields including web development, data analysis, artificial intelligence, scientific computing, and more.

Setting up Python environment: Installation

Activity 6.1:

Checking if python is installed on Windows PC

Aim:

To guide in checking if Python is already installed on Windows PC

FOR ONLINE READING ONLY

Materials: Computer

Instructions:

Step 1: In the start bar, type cmd.exe

Step 2: Open the command prompt

Step 3: Type " C:\Users\Your Name>python –version " in the case of computer used for this demonstration : C:\Users\Admin>python –version

Deliverable: The window as shown in Figure 6.1 will appear.

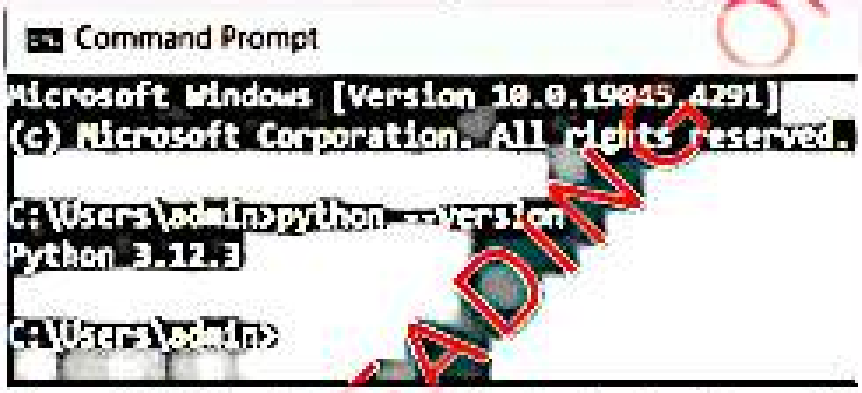


Figure 6.1: *Window showing a version of python installed on a PC*

Activity 6.2:

Setting up python environment

Aim:
To guide on the process of setting up Python on the computer systems.

Materials:
Python installer, Computer, Internet

Instructions:

Step 1: Obtain Python file from a teacher or download the installer from the official Python website (python.org).

Step 2: Run the installer and follow the installation instructions provided.

Step 3: During the installation, make sure to select the option to add Python to the system PATH. This allows you to run Python from any command prompt.

Deliverable: Python program installed in a computer

Integrated Development Environment (IDE): While Python code can be written and executed in any text editor, using an IDE can greatly enhance your development experience. Some popular Python IDEs include:

(i) *PyCharm:* A powerful and feature-rich IDE developed specifically for Python development.

FOR ONLINE READING ONLY

(ii) *Visual Studio Code (VSCode):* A lightweight and versatile code editor with excellent Python support through extensions.

(iii) *IDLE:* The default Python IDE that comes bundled with the Python installation.

Other IDEs includes Thonny, Netbeans or Eclipse. The IDEs are preferable when using large collections of python files. Using an IDE provides features such as code completion, debugging tools, integrated terminal, and project management capabilities, making it easier to write and manage your Python code. In some examples in this book, text editors like Notepad will be used.

Writing a simple Python program

Activity 6.3:

Writing the first Python program: "Hello fellow Africans"

Aim:
To understand the basic syntax and structure of a Python program

Materials:
Text editor (Such as Notepad) or any IDE, Computer installed with Python

Instructions:

Step 1: Open a text editor, Notepad++

Step 2: Type the following code:
print("Hello fellow Africans!")

Step 3: Save your first program in a Notepad as "HelloAfrica.py" at any location. In this case file is stored in Python folder on desktop. Note that, (.py is an extension for python files)

Step 4: Navigate to the directory where you saved your file, and run on the command line as in Figure 6.2



Figure 6.2: *Navigating the directory*

Deliverable: The output will be as in Figure 6.3



Figure 6.3: *Output of first Python program*

FOR ONLINE READING ONLY

Alternatively, you can open the IDLE on Python and open the HelloAfrica.py file to run it starting on the start bar with steps:

In search field, type IDLE→open→File→open→Desktop→Python→HelloAfrica→Run→Run module.

The output is shown in Figure 6.4:

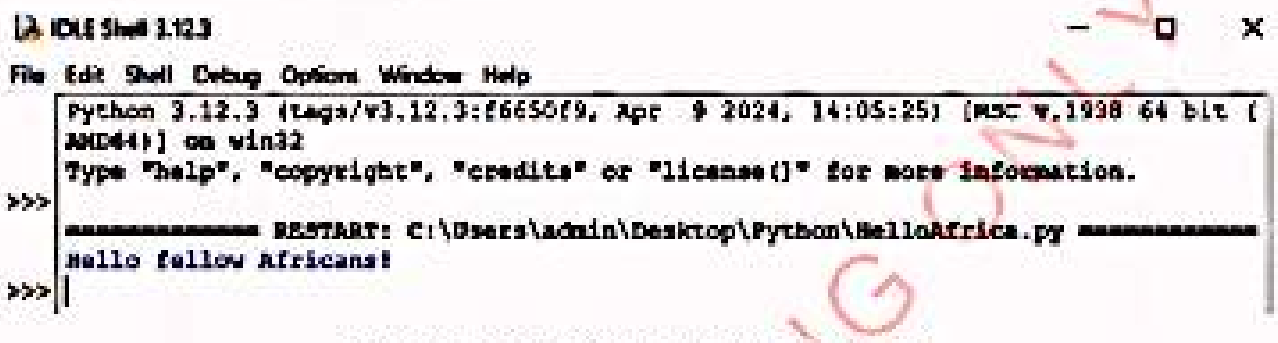


Figure 6.4: *First Python program using IDLE*

Virtual environments

It is recommended to use virtual environments to manage dependencies and isolate project environments. Virtual environments allow you to have separate Python installations and package dependencies for different projects. This helps avoid conflicts between different versions of packages used in different projects. To create a virtual environment, you can use tools like virtualenv or venv, which come bundled with Python. Here is how you can create a virtual environment using venv:

(i) Open a command prompt or terminal.

(ii) Navigate to your project directory.

(iii) Run the following command to create a virtual environment:

(iv) python-m venv myenv; This replace myenv with the name you want to give to your virtual environment.

(v) Activate the virtual environment. On Windows, run:

(vi) myenv\Scripts\activate

Once activated, any Python packages you install using pip will be installed within the virtual environment, keeping them separate from your system-wide Python installation.

Python fundamentals

Python fundamentals involves defining basic syntax, variables, operators, expressions and data types.

Basic syntax

The following explain important guideline for syntax in Python:

(a) *Indentation*

In Python, indentation is used to signify blocks of code. It is important to maintain consistent indentation, usually

FOR CLASSROOM READING ONLY

consisting of four spaces, to define code blocks within functions, loops, and conditional statements. By adhering to consistent indentation practices, the structure and hierarchy of the code become clear and easily understandable. For example,

```
if x > 100:
    print("x is greater than 100")
else:
    print("x is not greater than 100")
```

In this example, the two prints has gone inside to the right by four space leaving if and else at the left. If you skip indentation, Python will give you an error message, For example:

```
if x > 100:
print("x is greater than 100")
else:
print("x is not greater than 100")
```

This program will give you an error. It is important to use the same number of spaces in the same block of code so that Python will not give you an error

(b) Comments

In Python, comments are denoted by the # symbol and serve the purpose of explaining code or adding notes. Comments are not interpreted by the Python interpreter and are solely meant for human readers. They provide additional context, improve code readability, and can serve as reminders or explanations for future reference.
For example,

```
#This is a comment for this code
```

Variables

(a) Variable declaration

Python does not require a specific command for declaring a variable. In Python, a variable is automatically created when you assign a value to it for the first time. There is no need to explicitly declare the variable beforehand. Once a value is assigned to a variable, it is stored in memory and can be referenced by its assigned name throughout the program. This dynamic nature of variable creation simplifies the coding process and allows for more flexibility in Python programming. For example,

```
x = 256
y = "Mwanamwera"
print(x)
print(y)
```

(b) Variable naming

In Python, variables can have both short and descriptive names. However, there are certain rules and guidelines to follow. These include:

(i) A variable name must begin with a letter or an underscore character.

(ii) It cannot start with a number.

(iii) It can only consist of alphanumeric characters (letters and numbers) and underscores.

(iv) Variable names are case-sensitive, meaning that variables with different capitalization are considered distinct (such as age, Age, and AGE are treated as separate variables).

(v) Python keywords cannot be used as variable names, as they have

FOR ONLINE READING ONLY

special meanings and are reserved for specific purposes within the language.

The following are examples of writing variable names:

(i) myname = "Buricheka"

(ii) my_name = " Buricheka "

(iii) _my_name = " Buricheka "

(iv) myName = " Buricheka "

(v) MYNAME = " Buricheka "

(vi) myname3 = " Buricheka "

Operators

Operators are special symbols that perform operations on variables and values. While many operators are shared between Python and Java in previous chapter Five, there are notable differences in logical, ternary, increment/decrement, and some arithmetic operations. Understanding these differences is crucial when transitioning between these two languages. Table 6.1 shows the differences of perators as used in Java and Python:

Table 6.1: *Differences in operators between Java and Python*

Category	Operator	Java	Python
Logical operators	AND	&&	and
	OR	||	or
	NOT	!	not
Ternary operator		condition ? if_true : if_false	a if condition else b
Increment/decrement operators	Increment	++	N/A
	Decrement	--	N/A
Exponentiation		Math.pow(a, b)	a ** b
Floor Division		/ (when both operands are integers)	a // b

Expressions in Python

An expression is a combination of variables, operators, and values that yields a result value.

For examples,

$x = 10$

$y = 5$

$result = x + y * 2$ # Here the result is 20

By understanding variables, operators, and expressions, you can create more complex and powerful programs in Python. These fundamental concepts are the building blocks for writing efficient and effective python code.

FOR ONLINE READING ONLY

Data types

Python supports several built-in data types including numeric types(such as int, float, complex), sequence types (such as string, lists, turples and dictionary), Boolean type (such as bool) and other type (Such as null)

Control flow and decision making in Python

(a) Conditional Statements: if, elif, else

In Python, conditional statements are used to execute different blocks of code based on specific conditions. The syntax for the "if" statement is as follows:

```
if condition:
    # Block of code to execute if condition is True
```

The "elif" statement, short for "else if," enables you to evaluate multiple conditions consecutively. It allows you to check for additional conditions if the conditions in the preceding "if" statement(s) are not met.

On the other hand, the "else" statement is used to execute a block of code when the conditions of the "if" or "elif" statements are evaluated as False. It provides an alternative code path to be executed when the previous conditions are not satisfied. Here is an example that demonstrates the usage of these statements:

```
x = 10
if x > 0:
    print("x is positive")
elif x < 0:
    print("x is negative")
else:
    print("x is zero")
```

(b) Looping Constructs: for and while

Python offers two primary looping constructs: the "for" loop and the "while" loop. The "for" loop is utilised to iterate over a sequence, which can be a list, tuple, string, or any other iterable object. It allows you to perform a specific set of actions for each item in the sequence or execute a fixed number of iterations. This loop is particularly useful when you know the number of iterations in advance or when you want to process each element in a given sequence. For example,

```
animals = ['lion', 'leopard', 'buffalo']
for animal in animals:
    print(animal)
```

The "while" loop is employed to execute a block of code repeatedly as long as a specified condition remains true. It allows you to create a loop that continues until the condition evaluates to false. This type of loop is useful when the number of iterations is not known in advance, and the loop should continue until a certain condition is met or becomes false. For example,

```
count = 0
while count < 7:
    print(count)
    count += 1
```

FOR ONLINE READING ONLY

(c) Control statements: break, continue, and pass

Python offers control statements that allow you to modify the flow of a loop according to specific conditions.

The "break" statement is utilised to abruptly exit the loop before its normal completion. It provides a means to terminate the loop prematurely based on a certain condition.

The "continue" statement, on the other hand, is used to skip the remaining code within a loop for the current iteration. It allows you to jump to the next iteration without executing the subsequent statements in the loop.

Lastly, the "pass" statement is a placeholder that does nothing. It is used when syntactically required but no specific action is needed. It is commonly used as a temporary placeholder until the code is implemented. For example,

break :

```
# Search for a number in a list and stop when found
numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10,11,12,13]
search_number = 11
for number in numbers:
    if number == search_number:
        print("Number found!")
        break
    else:
        print("Searching...")
```

Continue:

```
# Print odd numbers from 1 to 13 skipping even numbers
for i in range(1, 14):
    if i % 2 == 0:
        continue
    print(i)
```

Pass:

Here are a couple of examples showcasing the usage of the "pass" statement:

FOR ONLINE READING ONLY

Visualisation of the program code in Python

In order to visualise and explore what is going on in the code execution, the open source software Thonny can be used. Figurekk illustrate the interface of this application . You can visualise the code execution through Activity 6.3.

Thonny main interface functions:

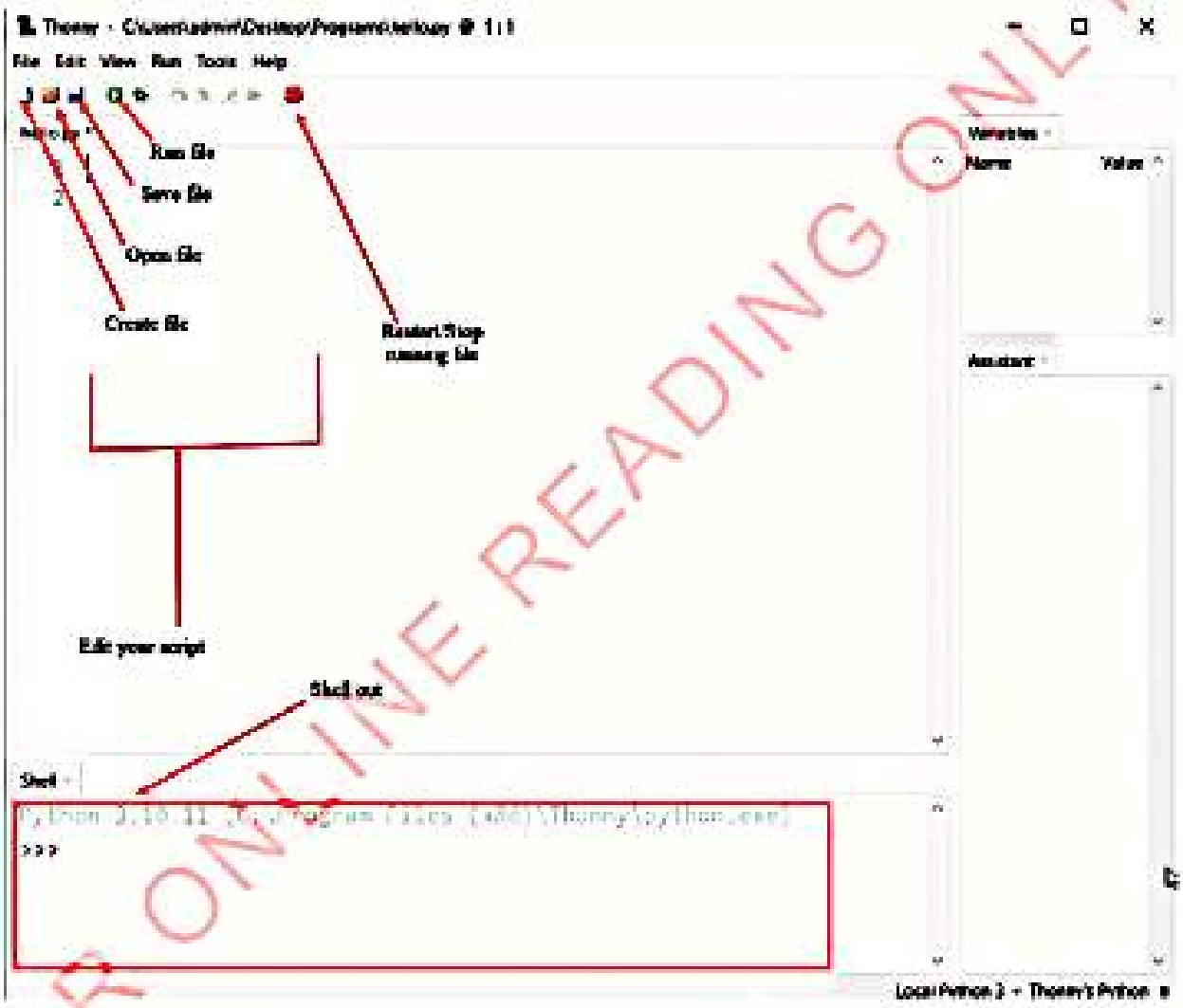


Figure 6.5: *Main interface functions of Thonny software*

FOR ONLINE READING ONLY

Activity 6.4:

Visualisation of program codes in Thonny software

Aim:

To understand Python code execution, improve debugging skills, track variable changes,

Materials: Computer, Thonny visualisation software or any tool

Instructions:

Use the following codes to visualise its execution in Thonny software:

```
for i in range(1, 14):
    if i % 2 == 0:
        continue
    print(i)
```

Step 1: Open Thonny software

Step 2: Open the file you saved in any location using Thonny interface. The file will be shown as in Figure 6.3.

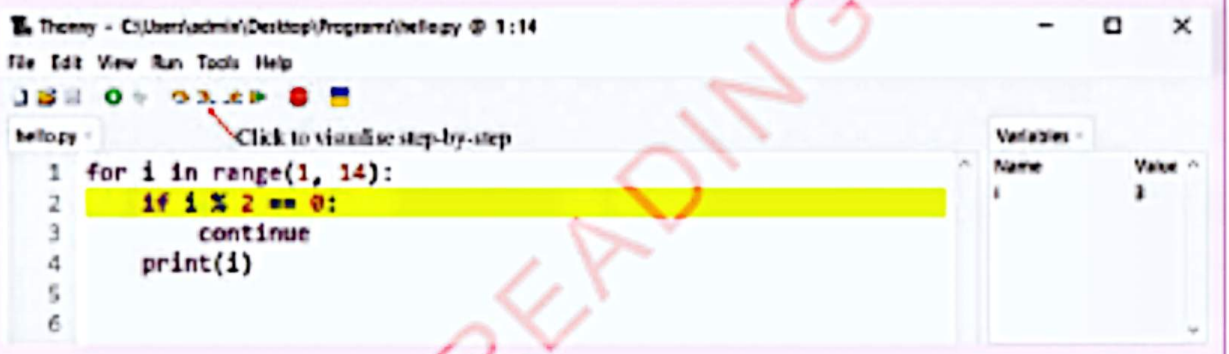


Figure 6.6: *Visualising step by step execution of program*

Example 6.1:

Placeholder function

```
# Placeholder for a function that does nothing
def placeholder_function():
    pass  # No implementation yet
```

In this example, you have a function called "placeholder_function()" that is defined but does not have any implementation yet. The "pass" statement is used as a placeholder to indicate that there is no code inside the function body at the moment.

Example 6.2:

Class definition without members

```
# Using pass in a class definition
class MyClass:
    pass  # Class definition without any members yet
```

FOR ONLINE READING ONLY

In this example, we have a class named "MyClass" that is defined but does not have any members (attributes or methods) defined yet. The "pass" statement is used as a placeholder to indicate that the class definition is empty for now.

In both cases, the "pass" statement serves as a null operation that does nothing. It is used when the syntax requires a statement, but no specific action or implementation is needed at that point.

Python functions and modules

Python functions and modules are powerful tools that help in organizing and structuring code. Functions allow us to break down complex tasks into smaller, reusable blocks of code. Modules are collections of functions and variables that can be imported and used in other programs.

Meaning of function

A function denotes a labeled segment of code designed to accomplish a specific task. It accepts inputs, termed parameters or arguments, and generates outputs, referred to as return values.

Function structure

Functions are declared using the "def" keyword, followed by the function name and parentheses.

Parameters, if present, are listed within the parentheses. The function's body is indented and encompasses the code executed upon function invocation. There is advantages in using functions in Python. The following are advantages of function:

(a) *Code reusability:* Functions can be invoked multiple times, curtailing code redundancy and fostering efficiency.

(b) *Abstraction:* Functions shield the implementation details, enabling users to concentrate on functionality without requiring knowledge of internal workings.

The following are some examples of how to write functions:

(i) Basic Function:

```
def invite(name):
    print("Welcome, " + name + "!")
welcome("Pepe Kale")  # Output: Welcome, Pepe Kale!
```

(ii) Function with return Value:

```
def cube(number):
    return number * number*number
result = cube(3)
print(result)  # Output: 9
```

FOR ONLINE READING ONLY

Modules in Python

Modules are files that contain Python code and define functions, classes, and variables. They enable code organisation into reusable components and help manage namespaces. Python includes a standard library with modules like math, random, and datetime that offer various functionalities. The following are some important points about module:

(a) Importing modules: To use functions or variables from a module, you import it into your Python script. You can import an entire module using the "import" statement followed by the module name. Alternatively, you can import specific functions or variables using the "from" keyword.

(b) Creating and using custom modules: You can create your own modules by writing Python code in a separate .py file. To use functions or classes defined in a custom module, you import the module into your script. Once imported, you can access the functions and variables using dot notation (module_name.function_name).

(c) Built-in functions: Python provides a set of built-in functions that are available without the need to import any modules. These functions cover a wide range of functionalities, including mathematical operations, string manipulation, type conversion, and more.

***Using modules*:** You can use modules depending on the context or situations you want.

(a) You can use the random module to generate a random number between a specified range where you can import the random module and employ the random.randint() function to create a random integer falling within the specified start and end values. Subsequently, the outcome is stored in the variable random_number and showcased through console printing. For example,

```
import random
start = 1
end = 10
random_number = random.randint(start, end)
print("Random number between", start, "and", end, "is:", random_number)
```

(b) You can use the datetime module to display the current date and time. Where you can import the datetime module and utilize the datetime.datetime.now() function to fetch the present date and time. The outcome is then captured in the variable current_datetime and exhibited through console printing. For example,

```
import datetime
current_datetime = datetime.datetime.now()
print("Current Date and Time:", current_datetime)
```

FOR ONLINE READING ONLY

Activity 6.5:

Creating and using a Python module

Aim:

Students will learn how to create a Python module and use it in a separate script. This activity will help students understand the concepts of modularity and code reuse in Python.

Materials Required:

(a) Computer with Python installed

(b) Text editor or Integrated Development Environment (IDE) such as PyCharm, Visual Studio Code, IDLE or Thonny.

(c) Sample Python codes.

Instructions:

Step 1: Create the module

(a) Open your text editor or IDE.

(b) Create a new file and name it mymodule.py.

(c) Write the following code in mymodule.py:

```
# Example code for mymodule.py
CONSTANT_PI = 3.14159
def add(a, b):
    return a + b
def subtract(a, b):
    return a - b
# More functions can be added as needed
```

(d) Save the file.

Step 2: Use the Module

(a) Create another new file in your text editor or IDE and name it main.py.

(b) Write the code in main.py:

```
# Example code for main.py
import mymodule

result_add = mymodule.add(500, 1300)
result_subtract = mymodule.subtract(101, 17)

print("Addition result:", result_add)
print("Subtraction result:", result_subtract)
```

(c) Save the file.

Step 3: Run the Script

(a) Open a terminal or command prompt.

(b) Navigate to the directory where you saved mymodule.py and main.py.

(c) Run the main.py script by typing *python main.py* command and pressing Enter:

(d) Observe the output in the terminal. It should display the results of the mathematical operations and the values of the constants.

Extension Activity:

(a) Add more functions to mymodule.py, such as modulus(a, b) and power(a, b), and use them in main.py.

(b) Create another module named geometry.py with functions like area_of_circle(radius) and circumference_of_circle(radius). Use these functions in a new script named geometry_test.py.

Deliverable:

Python module file, mainscript and screenshots of your python code.

FOR ONLINE READING ONLY

Document them in your portfolio.

Questions:

1. Reflect on how using modules can help in organising and reusing code.
2. Discuss how you can apply the concept of modules in larger projects.

Exercise 6.1

1. Create a Python script that defines variables of different data types (integer, float, string, and boolean). Print the type of each variable.
2. Write a Python program that takes two numbers as input from the user and performs addition, subtraction, multiplication, and division. Display the results for each operation.
3. Develop a Python script that accepts a string input from the user, converts it to uppercase, and prints the length of the string.
4. Create a Python program that initialises a list with five integer elements. Use a for loop to iterate through the list and print each element doubled.
5. Write a Python script that takes a number as input from the user and prints whether the number is positive, negative, or zero.
6. Define a Python function that takes two parameters and returns their sum. Call the function with two numbers and print the result.

OOP concepts in Python

The fundamental OOP concepts in Python include class and objects, Polymorphism, encapsulation, inheritance, and data abstraction

Class and objects

Python class

In Python, a class serves as a grouping of objects. It acts as a blueprint or template for creating individual objects. A class represents a logical entity that encompasses certain attributes (data) and methods (functions). It provides the structure and definition for creating objects with specific characteristics and behaviors..

To understand the need of creating a class, let us examine an illustration. Suppose you wish to keep track of numerous dogs with distinct characteristics, such as breed and age. If a list is employed, the first element might indicate the dog's breed, while the second element represents its age. However, when dealing with a collection of 100 different dogs, it becomes challenging to determine which element corresponds to which attribute. Moreover, if you want to incorporate additional properties for these dogs, such an approach lacks organisation and becomes unwieldy. This is precisely where classes come into play, as they provide a structured solution to effectively manage and organise data by defining attributes and behaviors for each individual object.

FOR ONLINE READING ONLY

In Python, the class keyword is used to define and create classes. Attributes refer to the variables that are associated with a particular class. In this scenario, attributes are always public and can be accessed using the dot (.) operator. For instance, if we have a class called MyClass, and it possesses an attribute named MyAttribute, we can access it by using the syntax MyClass.MyAttribute.

You can create an empty class, Program Example 6.1 shows how you can create empty class of Dog.

Program Example 6.1:

Creating an empty class in Python

```
class Dog:
    pass
```

In this example, Dog is a class with no attributes or methods. It serves as a blueprint that can be used to create objects with no initial state or behavior.

Python objects

In Python's object-oriented programming paradigm, an object represents an entity that possesses both state and behavior. These objects can be any real-world item such as a mouse, keyboard, chair, table, pen, and so on. Additionally, objects can be data types like integers, strings, floating-point numbers, arrays, and dictionaries. In fact, even simple entities like a single integer or a single string are considered objects. For example, the number 12 and the string "Hello, world" are both objects.

An object consists of:

(a) State: It is represented by the attributes of an object. It also reflects the properties of an object.

(b) Behavior: It is represented by the methods of an object. It also reflects the response of an object to other objects.

(c) Identity: It gives a unique name to an object and enables one object to interact with other objects.

To understand the state, behavior, and identity take the example of the class dog that was provided before. In the context of a dog, the identity can be thought of as the dog's name. The state or attributes of a dog refer to characteristics such as its breed, age, or color. The behavior of a dog encompasses actions like eating or sleeping.

Creating class and object in Python

Classes serve as blueprints for creating objects, encapsulating both data (attributes) and behavior (methods). To define a class, use the "class" keyword, followed by the class name and a colon to start the class block.

Creating objects (Instances)

Objects are instances of classes, representing individual entities based on the class blueprint. To create an object, instantiate the class by calling the class name followed by parentheses.

FOR ONLINE READING ONLY

(a) Class attributes and methods

Attributes are variables associated with the class or its instances, storing data related to the class. Methods are functions defined within a class, performing actions or providing functionality associated with the class. Instance attributes are specific to each object, while class attributes are shared among all instances.

(b) Constructor method (init)

The init method is a special method called when an object is created. It initializes the objects attributes and performs any necessary setup during object creation. Constructor parameters are used to initialise instance attributes with specific values.

(c) Instance and class methods:

Instance methods are defined with the "self" parameter and operate on individual object instances. Class methods are defined with the @classmethod decorator and operate on the class itself rather than instances.

The following program example 6.2 demonstrate how a class and object are created.

Program Example 6.2:

Program codes to demonstrate creating class and object

```
class Dog:
    # Class attribute
    species = "Canine"
    # Constructor method
    def __init__(self, name, age):
        self.name = name  # Instance attribute
        self.age = age
    # Instance method
    def description(self):
        return f"{self.name} is {self.age} years old"
    # Class method
    @classmethod
    def bark(cls):
        return "Woof!"
# Creating objects of the Dog class
dog1 = Dog("Buddy", 3)
dog2 = Dog("Max", 5)

# Accessing object attributes and methods
print(dog1.description())  # Output: Buddy is 3 years old
print(dog2.description())  # Output: Max is 5 years old
print(Dog.bark())          # Output: Woof!
```

FOR ONLINE READING ONLY

Implementation of key OOP features in Python

Polymorphism in Python

Polymorphism in Python enables objects from different classes to be treated as objects of a shared superclass. As explained earlier in other languages, polymorphism is achieved through method overloading and method overriding.

(a) *Method overloading*: Method overloading in Python involves defining multiple methods with the same name but different parameter lists. By default, Python does not support method overloading, but it can be simulated using default parameter values or variable-length argument lists.

(b) *Method overriding*: Method overriding happens when a subclass provides its own implementation of a method that is already defined in its superclass. It enables subclasses to customize the behavior of inherited methods. The following Program example 6.3 demonstrate polymorphism.

Program example 6.3:

Program code demonstrating polymorphism

```
class Animal:
    def speak(self):
        return "Animal speaks"
class Dog(Animal):
    def speak(self):
        return "Dog barks"
class Cat(Animal):
    def speak(self):
        return "Cat meows"
# Polymorphic behavior
animals = [Animal(), Dog(), Cat()]
for animal in animals:
    print(animal.speak())
```

Output:

```
Shell
>>> %Run animal.py
  Animal speaks
  Dog barks
  Cat meows
```

Figure 6.7: *Output of polymorphism program code*

FOR ONLINE READING ONLY

Encapsulation in Python

Encapsulation is a core concept in Python Object-Oriented Programming. It involves grouping together data and the methods that work on that data into a single unit called a class. Encapsulation involves ideas on access modifiers and 'getters and setters' as seen in other languages.

Access modifiers:

Unlike Java or C++, Python does not have built-in access modifiers. Instead, it relies on conventions to indicate the accessibility of attributes and methods. By convention, attributes and methods starting with a single underscore (_) are considered as protected, implying that they should not be accessed directly from outside the class. Attributes and methods starting with double underscores (__) are considered private and cannot be accessed from outside the class.

Getter and setter methods:

Encapsulation often involves the use of getter and setter methods to access and modify the attributes of a class. Getter methods are used to retrieve attribute values, while setter methods are used to modify them. This approach provides controlled access to attributes, enabling data validation and maintaining data integrity. The following Program Example6.4demonstratesencapsulation.

Program Example 6.4:

Program codes demonstrating encapsulation

```
class Person:
    def __init__(self, name, age):
        self._name = name
        self._age = age

    def get_name(self):
        return self._name
    def set_age(self, age):
        if age >= 0:
            self._age = age
    def display_info(self):
        print(f"Name: {self._name}, Age: {self._age}")
# Creating an object of the Person class
person = Person("John", 30)
```

FOR ONLINE READING ONLY

```
# Accessing attributes using getter methods
print(person.get_name())  # Output: John
# Modifying attribute using setter method
person.set_age(35)
# Displaying information
person.display_info()
```

Output:

```
Shell
>>> %Run person.py
  John
  Name: John, Age: 35
>>>
```

Figure 6.8: *Output of encapsulation program code*

Inheritance in Python

Inheritance is the process by which a new class (subclass or derived class) is created from an existing class (superclass or base class).

Syntax: In Python, inheritance is declared by mentioning the superclass name in parentheses after the subclass name. Subclasses have the ability to override superclass methods, allowing them to provide their own implementations or enhance the functionality of inherited methods. The following Program Example 6.5 demonstrate inheritance in Python.

Program Example 6.5:

Program codes demonstrating inheritance

```
class Animal:
    def speak(self):
        print("Animal speaks")
# Dog class inherits from Animal class
class Dog(Animal):
    def bark(self):
        print("Dog barks")
# Creating an object of the Dog class
```

FOR ONLINE READING ONLY

```
dog = Dog()
# Calling methods of the Dog class
dog.speak()  # Output: Animal speaks
dog.bark()   # Output: Dog barks
```

Output:

```
Shell
>>> %Run animal_2.py
  Animal speaks
  Dog barks
>>>
```

Figure 6.9: *Output of inheritance program code*

Abstraction in Python

Abstraction in Python involves hiding the complex implementation details of a class and showing only the necessary features of an object. It allows programmers to focus on the essential aspects of an object while hiding the irrelevant details. The following is an example of defining the syntax for abstraction.

```
from abc import ABC, abstractmethod
class MyAbstractClass(ABC):
    #abstractmethod
    def my_abstract_method(self):
        pass
```

The following Program Example 6.6 demonstrates abstraction in Python.

Program Example 6.6:

Program codes to demonstrate abstraction

```
from abc import ABC, abstractmethod
# Abstract class
class Animal(ABC):
    @abstractmethod
    def make_sound(self):
        pass
# Concrete classes
class Dog(Animal):
```

FOR ONLINE READING ONLY

```
    def make_sound(self):
        print("Woof!")
class Cat(Animal):
    def make_sound(self):
        print("Meow!")
# Create instances of concrete classes
dog = Dog()
cat = Cat()
# Call the abstract method
dog.make_sound()  # Output: Woof!
cat.make_sound()  # Output: Meow!
```

Output:

```
Shell
>>> %Run 'abstract class1.py'
  Woof!
  Meow!
>>>
```

Figure 6.10: *Output of abstraction program code*

FOR ONLINE READING ONLY

Exercise 6.2

1. Explain the concept of encapsulation in Python and why it is important in Object-Oriented Programming.
2. How does encapsulation help in achieving data hiding and preventing direct access to class attributes in Python?
3. Discuss the role of access modifiers such as public, private, and protected in implementing encapsulation in Python.
4. Provide an example of a Python class with private attributes and demonstrate how encapsulation is used to access and modify these attributes.
5. What is abstraction in Python and how does it differ from encapsulation?
6. Explain the concept of abstract classes and methods in Python. Provide an example to illustrate their usage.
7. How does abstraction help in simplifying complex systems and promoting code reusability in Python?
8. Discuss the benefits of using abstract classes and methods in Python programming, especially in large-scale projects.
9. Define inheritance in Python and describe its role in Object-Oriented Programming.

Chapter Summary

Reflect on what you have learnt from this chapter, summarise and keep it in your portfolio.

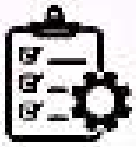
Revision Exercise 6

1. Write Python code to calculate the area of a rectangle with length 5 units and width 3 units. Print the result.
2. Write a Python program that asks the user to input their age. If the age is greater than or equal to 18, print "You are an adult". Otherwise, print "You are a minor".
3. Create a list of your favorite fruits. Write a Python program that uses a for loop to iterate through the list and print each fruit name on a new line.
4. Define a Python function called calculate_cube that takes an integer parameter side_length and returns the volume of a cube with that side length. Test the function with a side length of 4 units.
5. Write a Python program that prompts the user to enter their full name. Print the initials of their first name and last name in uppercase letters. For example, if the user inputs "Mtukula Jamaasoni", the output should be "MJ".
6. Write a Python program that reads a text file named "grades.txt". Each line in the file contains a student's name followed by their grade (for example, "Mponjoli 97"). Your program should calculate and print the average grade for all students. Handle any possible exceptions that may occur during file reading or grade calculation.
7. Explain the concept of polymorphism in Python OOP and provide a real-world example.
8. How does encapsulation enhance data security and privacy in Python classes? Give an example.
9. Describe the process of inheritance in Python and discuss its advantages in code reusability.
10. What is the purpose of an abstract class in Python? Provide an example of defining and implementing an abstract class.
11. How can you achieve method overriding in Python classes? Provide a scenario where method overriding is useful.
12. Discuss the difference between class attributes and instance

FOR ONLINE READING ONLY

attributes in Python classes with suitable examples.

13. Explain the role of constructors in Python classes. How do you define and use constructors in your code?
14. What is the significance of the super() function in Python inheritance? Give an example demonstrating its usage.
15. How do you implement encapsulation using private and protected attributes in Python classes? Provide syntax examples.
16. Describe the concept of multiple inheritance in Python. What precautions should you take when using multiple inheritance?
17. Discuss the benefits of using abstraction in Python programming. Provide an example of abstracting a complex system into classes.
18. How does method overloading differ from method overriding in Python? Provide examples to illustrate both concepts.
19. What is the purpose of the __init__() method in Python classes? How does it differ from other methods?
20. Describe the process of method resolution order (MRO) in Python inheritance. How does Python determine the order of method execution in subclasses?



Project: *Developing school Examination Grading System*

Project Description: The Mukuja Secondary School is initiating a project to develop an Examination Grading System using the Python programming language. The primary aim of this project is to create a comprehensive system for grading examination scores, aiding high schools in managing student details such as names, classes, total subjects, term, year of study, exam scores, and more. The system will facilitate the calculation of average marks attained by students and rank them based on their performance. Additionally, it will store student data in a database, offering features like record management, integration of examination records, information display, and tracking capabilities.

Project Objectives:

(a) To design and implement a Python program for managing examination scores and student details.

FOR ONLINE READING ONLY

(b) To calculate average marks and rank students based on their performance.

(c) To integrate a database for storing student records securely.

(d) To provide features for managing, displaying, and tracking examination information.

Project Tasks:

1. Design a user-friendly interface for inputting and displaying student details and examination scores.
2. Develop Python classes and functions for storing and managing student records, including details like names, classes, terms, and exam scores.
3. Implement algorithms for calculating average marks and ranking students based on their scores.
4. Integrate a database system for secure storage and retrieval of student data.
5. Test the system thoroughly to ensure accuracy and reliability.
6. Document each step of the development process, including code snippets, explanations, and screenshots.

Project Deliverables:

1. Python program source code files.
2. Documentation outlining the design, implementation, and testing of the Examination Grading System.
3. Test reports and results demonstrating the functionality and accuracy of the system.

FOR ONLINE READING ONLY



Chapter Seven

Operating Systems

Introduction

You use Operating Systems(OS) regular basis on computers, tablets, smartphones, and even gaming consoles. In this chapter, you will learn the concept of operating systems and how application system performance is affected by operating system design. Also, you will learn about operating system design and performance. The competencies developed will enable you to solve challenges related to Operating Systems design and performance.

Think

Traffic light controller of a busy intersection for flow of cars, pedestrians, and signals

7.1 Concept of Operating Systems

Case study 7.1: *Photo sharing among club members*

Members of an active School Photography Club have recently returned from an amazing field trip to Serengeti National Park, where they took amazing pictures of the environment and wildlife. They now face a challenge of distributing their pictures to every student in the school. Members of the club use different devices such as laptops, tablets, and smartphones with various operating systems such as Android, MacOS and Windows to share photos of different events. Hundreds of photos were taken, in different formats (JPEG, RAW), and saved across individual devices. The club wants to share the photos online in a way that is easily accessible to everyone. They also want to ensure that the photos are organised, captioned, and properly credited to the photographers.

FOR ONLINE READING ONLY

Questions:

1. How can the club members leverage the functionalities of their individual Operating Systems to share photos across different devices?
2. Which OS features can help organise the large number of photos efficiently?
3. How does using different image formats (JPEG vs RAW) affect the sharing process?
4. Which security measures the operating systems offer can help ensure that the photos are securely protected from unauthorised access?
5. How can the club leverage features within their chosen platform (online gallery, social media) to present the photos in an engaging way?

Meaning of Operating Systems

An operating system (OS) is a type of software that stands between a computer system's hardware and the several application software. An OS sits between a computer user and the computer's hardware to make things work smoothly for a user as shown in Figure 7.1.

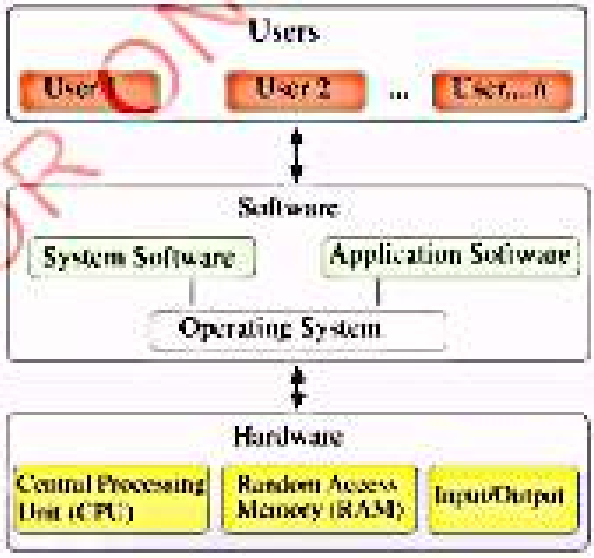


Figure 7.1: *Interaction of OS with user, software and hardware*

An OS is like an orchestra's, conductor, arranging the various components of the computer system so that they function as a unit. The OS manages all the resources of your computer, including memory, storage space, and processing power. It allocates these resources to different programs running simultaneously, enabling everything to run smoothly. An OS further allows computer users to install and run various software applications such as word processors, games, and web browsers. A computer operating system is basically a gatekeeper responsible for ensuring that programs run properly and interact with the hardware correctly.

Operating systems have evolved from simple software that was used to support the operations of physically big computers such as those with punch cards and flashing lights, to the most

FOR ONLINE READING ONLY

sophisticated OS versions that support unnecessarily big but more complex computers existing in the world today. Thus, early operating systems were very basic, mainly focusing on making the hardware work. However, they gained strength and usability over time, introducing capabilities like windows, multitasking, and internet connectivity.

Evolution of Operating Systems

The evolution of operating systems has been marked by significant milestones, from early mainframe systems to modern cloud-based platforms. Advancements in user interfaces, networking, and virtualisation have driven innovation and adaptation to changing computing paradigms. Looking ahead, operating systems will continue to evolve to meet the demands of emerging technologies and user needs.

Activity 7.1:

Evolution of Operating Systems exploration

Objective:

To understand the evolution of operating systems from their inception to the present day and explore their impact on computing and technology.

Materials Required:

1. Computers with internet access
2. Notebooks or digital devices for note-taking

Activity Steps:

Step 1: Discuss the basic concept of an operating system and its role in managing computer hardware and software

Step 2: Research how operating systems have evolved over time to meet the changing needs of users and advancements in technology

Step 3: Present your research findings in a comprehensive report summarising:

(a) Notable operating systems developed, including their key features and innovations during their assigned era

(b) Impact of these operating systems on computing and technology during that time.

Deliverable:

Detailed written report including table or chart comparing differentOSs

Document your research report in a portfolio.

Questions:

1. Reflect on how operating systems have evolved over time and the significance of these changes.
2. Discuss the role of operating systems in shaping the modern computing landscape and their influence on user experience, software development, and technological innovation.

FOR ONLINE READING ONLY

Types of Operating Systems

There are many types of operating systems in use today. Such operating systems may be classified based on various criteria including number of users and tasks to be handled. While many types of OS exist, this book describes general types of operating systems as follows:

(a) Desktop Operating Systems

These are Operating Systems that are specifically designed for desktop and laptop computers. Desktop Operating systems act as the control centre for personal computers. Through a graphical user interface (GUI) that includes elements like windows, icons, and menus, a desktop OS allow users to interact with the computer visually using a mouse and keyboard. Examples of desktop OS include Microsoft Windows, Apple macOS, Linux, and Chrome OS. Microsoft Windows is a widely used desktop OS globally, known for its user-friendly interface and vast software library. Apple macOS is an OS powering Apple Macintosh computers. It is known for its shiny design and focuses on creative workflows. Linux is a free and open-source OS that is popular among some ordinary computer users and software developers, offering flexibility and customisation of its various features. Chrome OS is a lightweight operating system designed for Chromebooks. It focuses on web applications and cloud storage.

(b) Mobile Operating Systems

A mobile operating system (OS) controls handheld devices like smartphones and tablets. Designed for portability, mobile OSs prioritise efficient battery usage and touch-friendly interfaces. They focus on apps, which are small programs for specific tasks like gaming or social media, and offer essential features such as calling, texting, and internet browsing, along with connectivity options like Wi-Fi and Bluetooth. Security is also a key focus, with measures like password protection and app permissions to safeguard personal data.

Mobile Operating Systems are tailored to meet these specific needs, with notable examples being Android, Apple iOS, and Windows mobile. Android, an open-source OS, is widely used for its customization options and extensive app library. Apple iOS powers Apple's handheld devices, known for its user-friendly interface, strong security, and smooth integration with Apple hardware and services. Microsoft's Windows Mobile OS is also used in some smartphones.

Overall, mobile Operating Systems are designed to support the unique requirements of handheld devices, ensuring mobility, app functionality, essential features, connectivity, and security.

(c) Server Operating Systems

A server Operating System (OS) is software designed to support server

FOR ONLINE READING ONLY

machines, which are powerful computers that store large amounts of data and run critical applications for multiple users on a network. Server OS ensures smooth, secure, and continuous operations, enabling collaboration and handling heavy-duty tasks with stability and reliability.

Server operating systems come in various types, each focusing on specific tasks. File servers manage and share files, web servers make web applications accessible, database servers organise and manage large data sets, and mail servers handle email communications. Each type is optimised for its specific function to ensure efficient performance. Examples of server Operating Systems include Microsoft Windows Server, Linux Server Distributions, and Apple macOS Server. Microsoft Windows Server is known for its powerful features and integration with Microsoft products. Linux offers flexible and customisable options like Ubuntu Server and Red Hat Enterprise Linux. Apple macOS Server adds server-specific features to the macOS platform, enhancing file sharing and centralised management tools.

Architectures of Operating Systems

The architecture of an Operating System provides the overall blueprint for how the OS interacts with the hardware. It defines how different parts of the Operating System communicate and work together. There are two main architectural choices: monolithic kernel architecture and Microkernel architecture. In a Monolithic kernel OS architecture, as shown in Figure 7.2, the core of the OS, called the kernel, is a single large program that handles everything from managing memory to running applications. It is simpler and faster but can be less flexible and secure. It is like a one-person orchestra leader who does everything themselves.

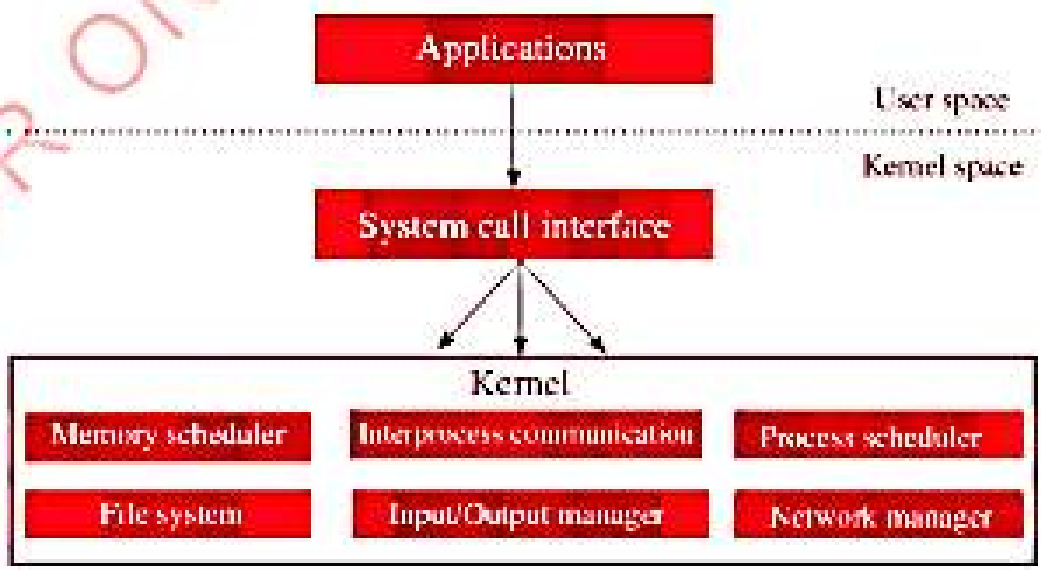


Figure 7.2: *monolithic kernel OS architecture (To be redrawn)*

FOR ONLINE READING ONLY

A Microkernel OS architecture, on the other hand, as shown in Figure 7.3 breaks down the kernel into smaller, more specialised components. Only the most essential tasks like memory management are done by the core kernel. Other services, like device drivers, run as separate programs. This makes the system more modular, secure, and easier to update. It is like a conductor with a dedicated team for each instrument section.

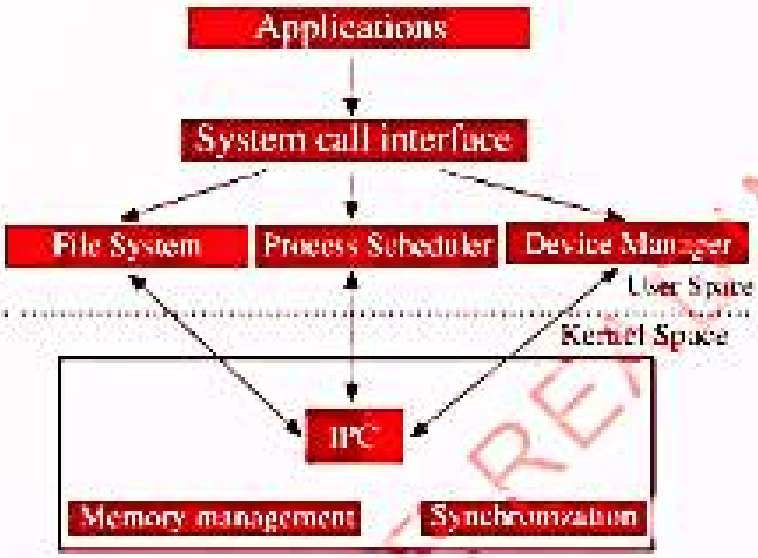


Figure 7.3: *Microkernel OS architecture*

Advantages of Microkernel architecture

The advantages of Microkernel architecture are:

(i) A highly modular design makes it easier to develop and maintain individual components.

(ii) It can support a wide range of operating systems and hardware platforms.

Disadvantages of Microkernel architecture

(i) The complexity grows as there is a necessity to oversee various components and facilitate communication among them.

(ii) Performance diminishes owing to the overhead involved in component communication.

(iii) Extensive testing and validation are imperative to guarantee the accurate functioning of the system.

Structure of an Operating System

The structure of an Operating System represents its various building blocks. It refers to how different core components of an OS are organised and interact. Some of the key components in the structure of an OS include the following:

(a) **Kernel:** This is a core component of an OS, which acts as the bridge between the hardware and software. It manages memory, storage, processes, and device drivers.

(b) **User interface:** This is what you see on your screen, like windows, icons, and menus. The user interface allows you to interact with the OS and other programs that run on top of it.

(c) **Device drivers:** These act as translators, allowing the OS to communicate with specific hardware components like printers or keyboards.

(d) **System utilities:** These are built-in programs that perform essential tasks like managing files, configuring settings, or troubleshooting issues.

FOR ONLINE READING ONLY

(e) **Application Programming Interfaces (APIs):** These are sets of instructions that allow software developers to create programs that interact with the OS and hardware.

Design of an Operating System

This provides the guiding principles of how an OS is built to achieve specific goals. The following are some key OS design principles:

(a) **Security:** The OS should protect the computer from unauthorised access, malware, and other threats.

(b) **Stability:** The OS should run smoothly and reliably, avoiding crashes or errors.

(c) **Performance:** The OS should efficiently utilize system resources to provide a responsive experience.

(d) **Usability:** The OS should be user-friendly and easy to learn for people with varying technical skills.

Impact of operating system design on application system design and performance

An OS's design significantly impacts how software application systems are designed and how they perform. An operating system (OS) is similar to a house's foundation. A strong foundation permits the construction of many structures (application software) on top of it. However, a weak foundation may restrict what may be constructed and how effectively it performs. Here is how OS design influences the development of software applications:

(a) **Resource Management:** The OS dictates how apps access resources like memory, processing power, and storage. An efficient OS design allows software applications to run smoothly without competing for limited resources.

(b) **Security:** The OS provides security features that can be leveraged by software applications. A secure OS helps developers build software applications that are less vulnerable to attacks.

(c) **Application Programming Interfaces:** The OS offers a set of instructions that can be used by software application developers to interact with the computer hardware and other software components. A well-designed API set makes it easier for developers to create powerful and efficient software applications.

(d) **Performance bottlenecks:** An OS with limitations in memory management, processing power allocation, or inefficient device drivers can create bottlenecks that slow down software applications. For example, modern operating systems are designed for multitasking, allowing users to run multiple software applications simultaneously. This requires efficient resource management to ensure smooth performance for all running applications.

(e) **Compatibility:** A well-designed OS ensures compatibility with various

FOR ONLINE READING ONLY

hardware components and software standards. This allows software applications to run smoothly on different systems without needing major modifications. For example, a well-designed OS uses industry-standard interfaces like USB (Universal Serial Bus) and drivers, which are pre-written software programs that act as translators, allowing the OS to recognize and communicate with a new hard drive, regardless of its manufacturer. You can access and use the hard drive without needing to install additional software specific to that manufacturer. With respect to software compatibility, a well-designed OS uses published software standards like APIs that act as a common language between programs and the OS. For example, a photo editing program, likely designed to work across different Operating Systems, uses these APIs to interact with the OS and access functionalities like opening files or using the graphics card, allowing you to run the program smoothly on your specific computer system.

(f) **Hardware utilization:** The OS plays a role in how efficiently hardware resources are used. A good design optimises hardware utilisation, leading to better performance of software applications. For example, in mobile gaming, mobile operating systems with features like hardware acceleration for graphics processing can significantly improve the performance of high-end games.

Activity 7.2:

Investigating key features of an Operating system

Aim:

To examine and understanding its functionality, perfomance, security, and user interface.

Use the internet to research and write a report of no more than two pages on the following aspects of the operating system installed on a personal computer, smartphone, or tablet owned by your parent, relative, or your school.

1: Investigate and summarize the features, strengths, and weaknesses of all OS versions prior to the currently installed version.

2: Identify and describe the key components of the currently installed OS version. Explain the function(s) of each component.

3: Provide two examples of application software that are negatively impacted by limitations in resource management, inefficient device drivers, or security measures within the currently installed OS version.

4: Explain how these limitations affect the performance or usability of the software.

Deliverables:

Produce a two-page report covering all the aspects mentioned above. Ensure your report is well-organised, clear, and concise. Document your report in a portfolio.

FOR ONLINE READING ONLY

Exercise 7.1

1. Suppose that you are a technical support specialist assisting someone unfamiliar with computers. Use simple terms and diagrams to explain the basic functions of an operating system and how it interacts with hardware and software applications.
2. Imagine that you have been given the task of creating a user guide for a new mobile app that needs to be user-friendly for people across different cultures and languages. How can you leverage the various features of operating systems to create a clear and universally understandable user interface?
3. Conduct research and write a two-page report on different types of malware that can target operating systems. How can an understanding of OS architecture (such as, kernel, security features) help users protect their devices from such threats?
4. Compare and contrast the design considerations for a desktop operating system and a mobile Operating System. How do these considerations relate to user needs and resource limitations of each device?
5. Imagine that you and your classmates are developing a new open-source Operating System that focuses on education. How can you leverage online collaboration tools and communication platforms to involve a global community of software developers and testers in the development process? What benefits might this approach have?

Operating System design and performance

Process management

A process is a program in execution, requiring resources like memory and processing power. The OS acts like a headmaster, managing processes (teachers) to ensure smooth operation. Threads, smaller units within processes, share resources and enable efficient multitasking. For example, a web browser may use separate threads for downloading content, displaying pages, and handling user interactions simultaneously. The OS allocates resources and schedules CPU time to processes and threads, preventing conflicts and ensuring smooth multitasking. Understanding processes and threads highlights the intricate work behind your computer's efficient operation.

The OS creates new processes when you launch software applications, allocating resources like memory, storage, and CPU time, similar to a headmaster assigning teaching materials. Multiple processes may compete for CPU time,

FOR ONLINE READING ONLY

so the OS uses scheduling algorithms to manage the queue and ensure smooth multitasking. This prevents resource hogging and prioritises demanding tasks, like gaming. The OS also facilitates communication between processes and terminates them when not needed, freeing resources and maintaining system stability.

Process management in an OS provides multitasking, allowing multiple programs to run without slowdowns. It ensures efficiency by managing resources effectively, preventing any single program from monopolising them. System stability is maintained, so if a program crashes, only its process is affected, not the entire system. Good process management ensures your computer can run multiple programs smoothly, maintaining efficiency and stability.

Activity 7.3:

Exploring process management on a computer

Aim:

To explore and understand how process management works in an operating system by observing and analyzing processes on a computer.

Materials:

A computer (Windows OS, macOS, or Linux distribution) or a smartphone/tablet

Instructions:

Step 1: Identify running processes:

(a) Open Task Manager (Windows), Activity Monitor (macOS), or System Monitor (Linux).

(b) Identify different processes running on the computer.

(c) Observe how various processes utilize CPU resources.

(d) Note how much memory each process consumes.

(e) Identify tools that show which processes are accessing the hard drive.

Step 2: Observe resource utilisation:

(a) While monitoring the Task Manager/Activity Monitor/System Monitor, perform the following tasks:

(i) Open a web browser and visit a few websites.

(ii) Open a music player and play a song.

(iii) Open a document editing program and start typing.

(iv) Open a heavy game or video editing software.

(b) Observe and note how CPU usage and memory consumption change as you open and use these programs.

Step 3: Analyse processes:

(a) Identify processes associated with specific applications you have opened.

FOR ONLINE READING ONLY

(b) Identify any background processes running even when not actively using them.

(c) Describe how demanding applications (Such as, games or video editing software) affect overall system performance.

Step 4: Research and documentation:

(a) Use the internet to:

(i) Research different types of processes (system processes, background processes) and their functions.

(ii) Explore online resources or documentation related to Task Manager, Activity Monitor, or System Monitor for more advanced information.

Deliverables:

Produce a one-page report covering all the aspects mentioned above about process management and its role in monitoring system performance. Relate this to the previous discussion on process creation and scheduling within the OS. Ensure your report is well-organised, clear, and concise. Keep your report in a portfolio

Note: If a personal computer is not available, perform the above activity using a smartphone or tablet and the appropriate system monitoring tools available on those devices.

Exercise 7.2

1. Suppose you are a tech support specialist for a school computer lab. A student reports that their computer is slow and unresponsive. Use your understanding of process management, to explain in simple terms to the student what might be causing the issue and how they can identify resource-heavy processes.
2. Research and compare different process scheduling algorithms used by operating systems, and explain how these algorithms impact the performance of modern computer games and the overall user experience.
3. Imagine that you are part of a team developing a new mobile operating system that focuses on energy efficiency. Explain how understanding process management can help you to optimise resource allocation and battery life on mobile devices.
4. Security threats can exploit vulnerabilities in processes. Research and discuss different types of malware that target processes. How can users be vigilant and identify potential threats related to process management?
5. Open-source operating systems like Linux are known for their customisation options. Investigate

FOR ONLINE READING ONLY

how advanced users can leverage the command line to manage processes on Linux. Explain the benefits and potential challenges of this approach.

CPU/Processor scheduling

CPU scheduling is about ensuring keeping the computer operations running smoothly. Imagine a busy teacher's office. Students (processes) need to be served by the teacher (use the CPU) efficiently. CPU scheduling is like the teacher, ensuring everyone gets their fair share of CPU time without chaos. But what happens if students get stuck waiting for each other's teacher? Specifically, suppose one student is being served by the first teacher while waiting for the services from the second teacher, and another student is being served by Teacher Two while waiting for the services of the first osne. This creates a deadlock. In the context of an operating system, a deadlock happens when multiple processes are waiting for resources held by each other. For example, a deadlock might happen when one process happens to be holding one computer resource (such as, CPU time) while waiting for a resource (like, memory) held by another process to continue execution. If two or more processes find themselves in such a situation, then the computer will likely stop working smoothly. Thus, because programs need CPU resources to execute, CPU scheduling helps to decide which process gets the CPU next, impacting program responsiveness and overall system performance.

There are several CPU scheduling algorithms, including the following three:

(a) **First Come First Served (FCFS):** Like a waiting line, processes are served in the order they arrive. It's fair but can be slow if short processes get stuck behind long ones.

(b) **Shortest Job First (SJF):** Prioritises the shortest processes, leading to faster completion times on average. However, it requires knowing the process lengths beforehand, which isn't always possible.

(c) **Round Robin (RR):** Acts like a fair organiser, ensuring that each process gets a turn for the CPU usage for a short period before switching to others. The RR algorithm has several benefits, including the following three. The first benefit is fairness, because each process gets a chance to run for a short time, even if it is a long process overall. The second benefit of RR is responsiveness, in a sense that short processes can finish quickly, and users experience better responsiveness. The third benefit of RR is that it prevents starvation. Unlike FCFS, in RR shorter processes do not get stuck behind long ones. Every process eventually gets the CPU time, even if it is for short bursts.

FOR ONLINE READING ONLY

Activity 7.4:

Visualizing CPU/Process scheduling using simulation tool

Objective:

Create executable programs, load them into the CPU memory, and run them under the control of the operating system.

Instructions:

Step 1: Start the CPU simulator (refer to previous experiments using the CPU simulator).

Step 2: In the compiler window, select the **COMPILER** button, as shown in Figure 7.4.

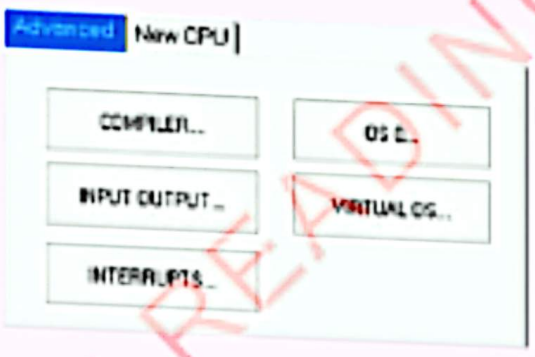


Figure 7.4: *Compiler button of the CPU simulator*

Step 3: Enter the following code in the compiler source editor area as shown in Figure 7.5 and click **COMPILE**.



Figure 7.5: *Compiler source code editor*

FOR ONLINE READING ONLY

Step 4: Load the compiled code into the CPU simulator's memory:

(a) Click the **LOAD IN MEMORY** button in the compiler window, as shown in Figure 7.6.

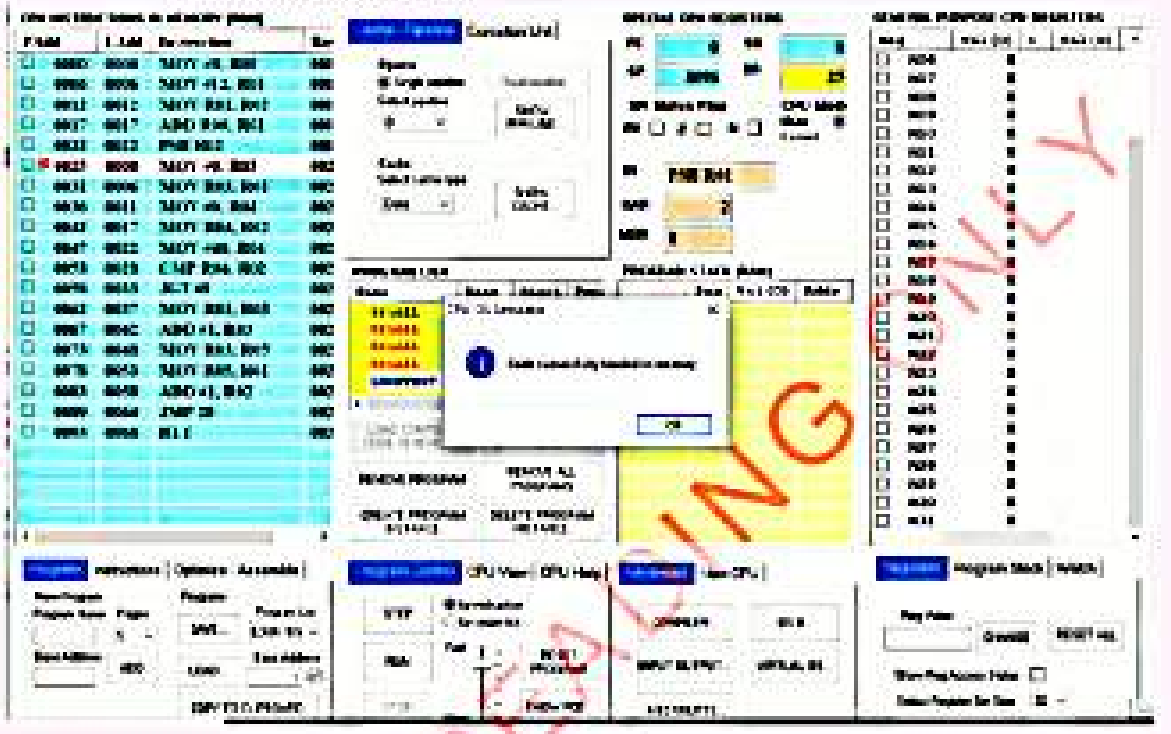


Figure 7.6: *Compiler window*

(b) The loaded code should appear in the **PROGRAM LIST** as shown in Figure 7.7 (LOOKTEST).

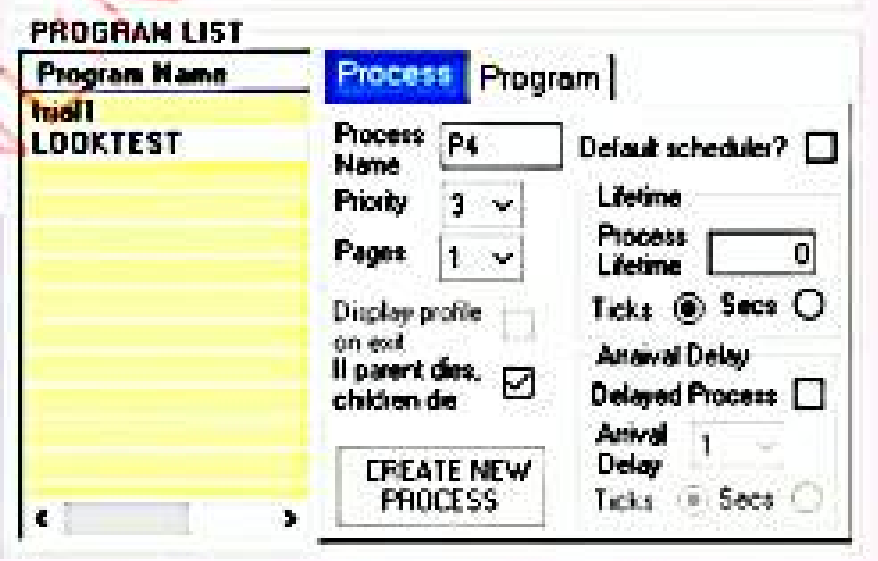


Figure 7.7: *Program list window*

Step 5: Create processes from the programs in the OS simulator, as shown in Figure 7.8:

(a) Click on the OS button and select **CREATE NEW PROCESS.**

FOR ONLINE READING ONLY

(b) Repeat this four times.

(c) Check the First-Come, First-Served (FCFS) scheduling policy in the **SCHEDULER/Policies** view.

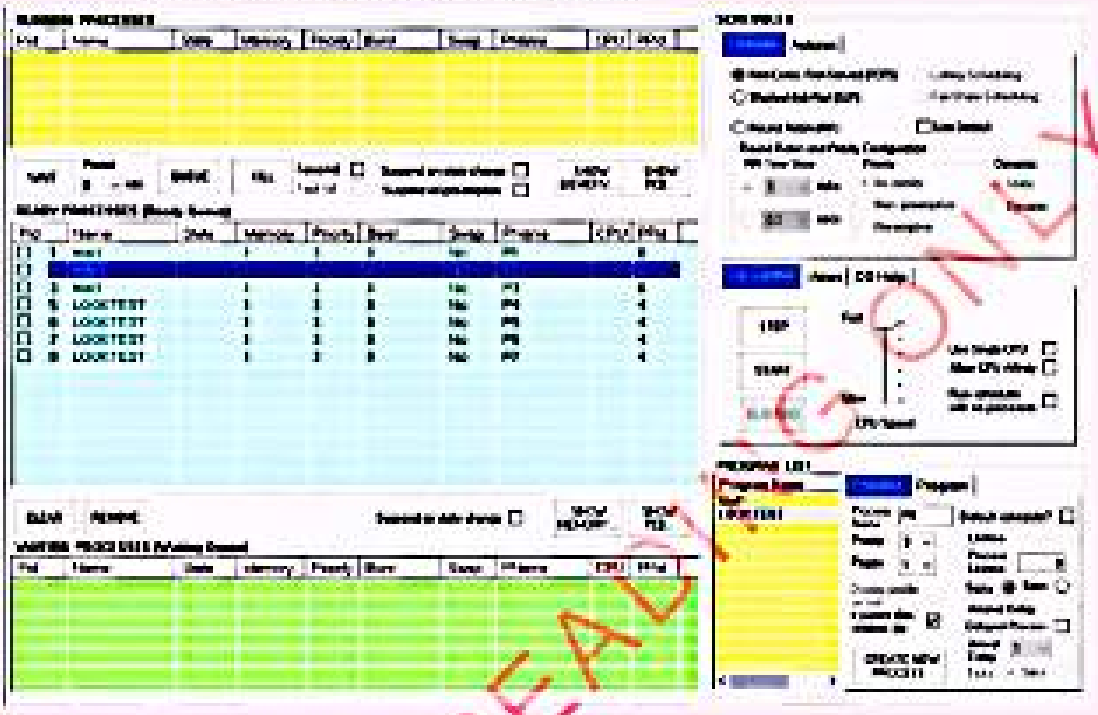


Figure 7.8: *Scheduling policy*

Step 6: Click **START** in the **OS** simulator to run the processes and observe the results, focusing on the **RUNNING PROCESSES** and **READY PROCESSES** views as shown in Figure 7.9.

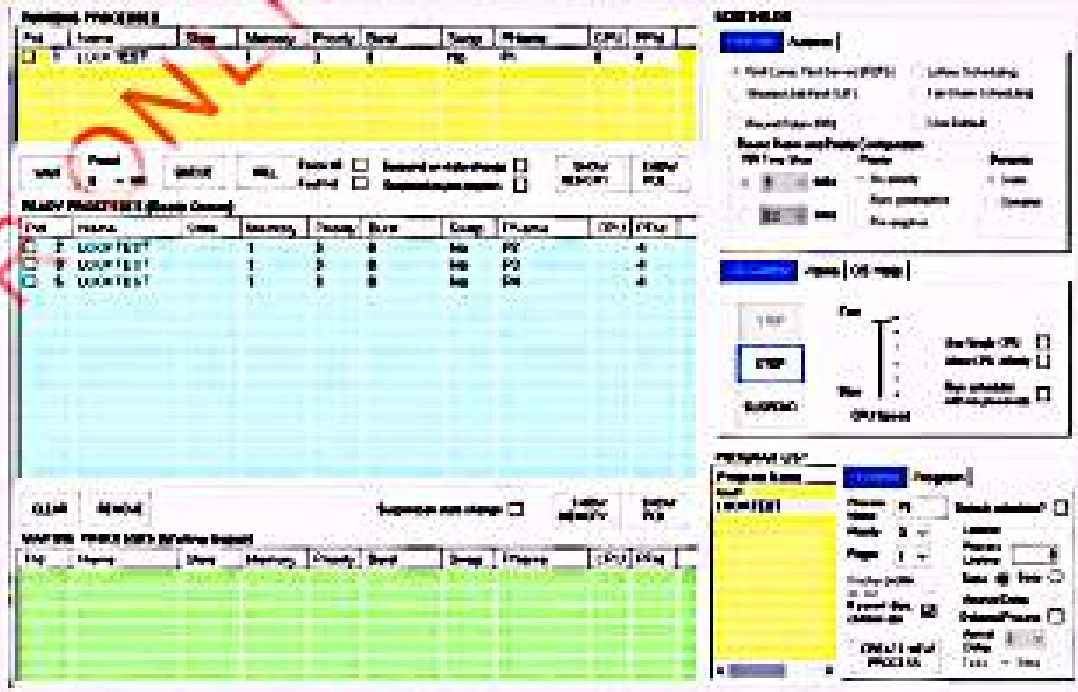


Figure 7.9: *OS simulator- running processes*

FOR ONLINE READING ONLY

Step 7: Use different scheduling policies (a) Shortest Job First (SJF), and (b) Round Robbin (RR), repeat steps 1 to 6.

Deliverable:

Write a brief report that document the key experimental procedures, including the screenshots of the setup in each step and the results. Document them in your portifolio.

FOR ONLINE READING ONLY

Deadlock

Deadlock refers to a specific condition when two or more processes are each waiting for each other to release a resource, or more than two processes are waiting for resources in a circular chain. Deadlock occur when each process needing what another process has. This results is blocking from sharing resources such as memory, devices, links.

Consider a bridge crossing scenario as in Figure 7.10.



Figure 7.10: *Causes of deadlock (The figure should be redrawn)*

Each segment of the road can be viewed as a resource. A car must own the segment it occupies and must acquire the segment it is moving into. For a bridge, a car must acquire both halves of the bridge. Traffic is allowed only in one direction at a time. A problem arises when two cars traveling in opposite directions meet on the bridge, with each car acquiring one segment and needing the next. If a deadlock occurs, it can be resolved by having one car back up, preempting resources and rolling back.

This may require several cars to back up. A deadlock in an operating system occurs when a set of processes become stuck, waiting for each other to release resources. Figure 7.11 shows a simplified diagram to illustrate the deadlock situation in an OS with processes and resources.

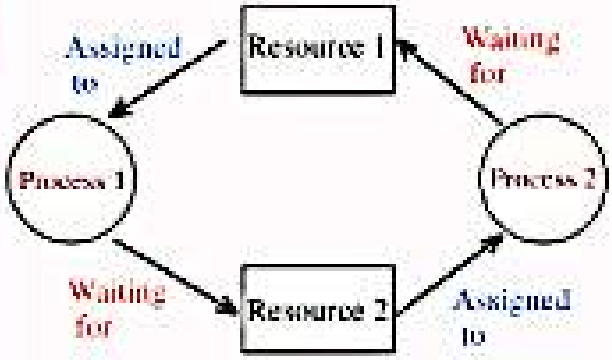


Figure 7.11: *Deadlock situation (to be redrawn)*

Process 1 holds Resource 1 and requests Resource 2. **Process 2** holds Resource 2 and requests Resource 1. In this case, Process 1 cannot proceed because it needs Resource 2, which is held by Process 2. Process 2 cannot proceed because it needs Resource 1, which is held by Process 1. Both processes are waiting for each other to release the resources, causing a deadlock.

Deadlock preventive strategies

No scheduling algorithm is perfect. The choice of an algorithm depends on the workload and required performance levels. To minimize the possibility of deadlock happening, operating system developers can use preventive or avoidance strategies. There are three key techniques to prevent deadlock:

(a) **Mutual exclusion:** Only one process can access a resource at a time. This is like accessing only one teacher at a time.

(b) **Hold and wait:** A process can only hold onto one resource at a time and must request others while not holding any. No hoarding.

(c) **No preemption:** Once a process has a resource, it cannot be forcibly taken away. No pushing in line.

(d) **Semaphores:** These are special flags within the operating system that act like signals to control access to shared resources. Semaphores work as follows:

(i) **Available resource:** When a resource like a file or printer is free, the semaphore is set to "available" and so the process is allowed to use the shared resource.

(ii) **Requesting access:** When a process needs to use a resource, it checks the semaphore. If it is available, the process gets access, and the semaphore is switched to "in use" to prevent others from accessing the resource.

(iii) **Releasing the resource:** When the process finishes using the shared resource (e.g., finishing file reading or using a printer), it releases it, and the semaphore is switched back to "available", allowing the next waiting process to proceed.

(e) **Critical sections:** This is the actual part of the process code where the resource is used. It is similar to the time a student spends printing their document. The operating system ensures only one process is in its *critical section* using the resource at a time.

FOR ONLINE READING ONLY

Although it tricky to completely avoid deadlock situations, operating system developers can analyse processes and resource requests in advance to prevent deadlock situations. Imagine assigning "service tickers" (resource claims) to students before entering the line for accessing teachers. This ensures everyone can get the resources they need. However, despite all preventive and avoidance efforts, deadlocks can still occur. Thus, if deadlock occurs, you can sometimes restart some processes or forcibly take away resources to break the cycle. It is like the teacher stepping in to rearrange the line. Note that deadlock prevention is mainly ideal, but avoidance can be more practical, while recovery from deadlock is disruptive and best avoided.

Managing shared resources prevents data corruption by ensuring that only one process can access a file at a time. Conflicts such as those resulting from having two users editing the same word processing file simultaneously. It also facilitates fairness by ensuring that processes take turns using shared resources, preventing one process from hogging them. Finally, because operating systems use a combination of approaches to manage shared resources efficiently, understanding these concepts will help you to appreciate the complexity of running multiple programs on a computer and ensuring smooth operation for everyone.

Activity 7.5:

Online interactive CPU Scheduling Simulation

Objective:

This activity will help you explore CPU scheduling algorithms in a fun and interactive way. You will learn how First Come First Served (FCFS) and Shortest Job First (SJF) impact process turnaround time.

Materials:

A computer with internet access to perform the activity.

Instructions:

Step 1: Head to the Simulation

Visit https://visualgo.net/, an online simulator that allows you to visualise CPU scheduling algorithms with a user-friendly interface.

Step 2: Choose an Algorithm

Click on "Algorithms" and select "First Come First Served (FCFS)."

Step 3: Create Processes

Click on "Processes" to define process names and their burst times (CPU time needed) in milliseconds. Start with a simple scenario: create 4-5 processes with varying burst times (e.g., P1: 500ms, P2: 300ms, P3: 800ms).

Step 4: Run the Simulation

Click "Run" and observe. The simulator will display the execution order,

FOR ONLINE READING ONLY

Gantt chart (visual representation of process execution), and calculate the turnaround time for each process.

Step 5: Analyse FCFS

(a) How do processes with shorter burst times fare in this algorithm?

(b) Observe the waiting time for each process. Does it seem fair?

Step 6: Experiment with SJF

Click on "Algorithms" and select "Shortest Job First (SJF)."

Step 7: Run the simulation again

Use the same processes you created earlier. Observe the execution order, Gantt chart, and turnaround times.

Step 8: Compare and analyse

(a) How does the turnaround time for each process differ between FCFS and SJF?

(b) Which algorithm seems to favor shorter processes?

(c) Discuss the advantages and disadvantages of each algorithm based on the simulation results.

Step 9: Change the burst times

Modify the burst times of some processes and rerun both simulations. Does it alter the scheduling outcomes?

Step 10: Explore more algorithms

The website offers other scheduling algorithms like Round Robin (RR) and Priority Scheduling. Experiment with them and see how they impact process execution and turnaround time.

Deliverable:

Write a report that document the key experimental procedures, including the screenshots of the setup in each step and the results. Document them in your portifolio.

Exercise 7.3:

1. How does CPU scheduling impact a digital workflow of a computer user?
2. Research and compare different CPU scheduling algorithms used by three operating systems of your choice. Work with a colleague to create a table or infographic summarizing the strengths and weaknesses of each algorithm.
3. Imagine you are part of a development team creating a new operating system for a self-driving car. Explain how CPU scheduling algorithms play a critical role in ensuring real-time response for critical tasks like braking or sensor data processing.

Memory management

In addition to CPU scheduling, the OS is also responsible for allocating memory space to different processes when needed. Main memory is the physical RAM available in a computer. The OS performs main memory management through different memory allocation strategies, including the following:

(a) *First-Fit:* A memory block is allocated to the first process that fits

FOR ONLINE READING ONLY

(b) *Best-Fit:* Tries to find the smallest available block of memory that best matches the needs of the process.

However, with time, as processes come and go, unused memory fragments can appear, reducing efficient usage of the computer memory. This is the concept of memory fragmentation, which the operating system much also address, to ensure that all the available computer memory is used in the best possible way. Moreover, virtual memory might sometimes be required to extend the RAM using the hard drive. When becomes full, the operating system can temporarily move less-used data to the hard drive. In so doing, an illusion that the computer has more RAM than is physically available is created.

File system structure and implementation

The computer's file system is like a well-organised filing cabinet within the computer's hard drive. Directories and files organises data into directories (folders) and files (documents). The File Allocation Table (FAT) or inodes are similar to filing indexes that keep track of where data for each file is physically located on the hard drive.

Input/Output systems and device management

Input/Output systems manage communication between the computer and various devices.

(a) ***Device drivers:*** These facilitate translation for each device, allowing the OS to understand and communicate with different hardware components such as printers and keyboards. The operating system provides specific instructions for using the printer, scanner, internet, and several other devices.

(b) ***Interrupt handling:*** Devices can interrupt the CPU to request attention. The operating system manages these interrupts efficiently, ensuring that all devices get serviced without causing delays.

Activity 7.6:

Monitoring the usage of computer memory

Aim:

This activity allows you to explore memory management concepts using your own computer. You will learn how to monitor memory usage and see how programs affect system resources.

Materials:

A computer with internet access to perform the activity.

Instructions:

Step 1: Finding the Task Manager/ Activity monitor/System Monitor

(a) Windows: Press Ctrl + Shift + Esc to open Task Manager.

(b) Mac: Click on the Apple icon in the top left corner and select "System Preferences" > "Dock & Menu

FOR ONLINE READING ONLY

Bar." Enable "Activity Monitor" under "Other."

Step 2: Exploring memory usage

(a) Locate the "Memory" or "Memory Usage" section in Task Manager/ Activity Monitor.

(b) Observe the following details:

(i) Total memory: This is the total physical RAM installed on your computer.

(ii) Used memory: This shows how much memory is currently being used by running programs and system processes.

(iii) Available memory: This indicates the amount of free memory remaining.

Step 3: Monitoring memory usage in action

(a) Open a few different programs (e.g., web browser, word processor, music player). Observe how the "Used Memory" value increases as programs launch and consume memory resources.

(b) Close some programs and observe how "Used Memory" decreases, freeing up memory for other tasks.

Step 4: Observe memory pressure

Some systems might display a "memory pressure" indicator. This helps you to assess whether your system is experiencing memory limitations.

(a) Low pressure: Your system has enough memory to handle current tasks efficiently.

(b) High pressure: Memory usage is approaching available capacity, and performance might be impacted.

Step 5: Further experimentation

(a) Open multiple browser tabs or windows simultaneously, and observe how memory usage increases with more active content.

(b) Try running a resource-intensive program (e.g., video editing software). See how it significantly impacts memory usage compared to simpler applications.

Deliverable:

Write a two-page report on OS memory management, covering the following aspects

(a) Explain the importance of memory management for the smooth operation of a computer system.

(b) Discuss how memory usage impacts application performance. What might happen if a program requires more memory than available?

(c) Use the resources on the internet and the school library to study the concept of *virtual memory* and how it helps to extend the available memory by utilizing hard drive space

(d) Document your findings in a portifolio.

Chapter Summary

Reflect on what you have learnt from this chapter, summarise and keep it in your portfolio.

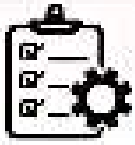
FOR ONLINE READING ONLY

Revision Exercise 7

1. Research the concept of context switching in CPU scheduling. How can operating systems minimize context switching to improve the gaming experience?
2. Operating systems use fairness and efficiency principles when scheduling CPU tasks. Discuss with your classmates the ethical considerations of prioritizing certain processes (e.g., emergency services applications) over others. How can we balance fairness with efficiency in CPU scheduling?
3. How does CPU scheduling affect app responsiveness and patience?
4. How can you optimise the performance of a smartphone using CPU scheduling knowledge?
5. How does CPU scheduling promote collaboration and sharing of computer resources?
6. How does CPU scheduling connect to coding and resource management?
7. How can the knowledge of CPU scheduling prepare you for emerging technologies?
8. Provide two real-world cases of deadlock. For each case briefly explain its causes and ways to handle the deadlock. How does the OS handle process synchronization to avoid race conditions?
9. How does the OS use CPU scheduling to improve efficiency?
10. What are the key differences between preemptive and non-preemptive scheduling?
11. How does the OS ensure data security and integrity on storage devices?
12. What mechanisms does the OS use to detect and recover from deadlocks?
13. Imagine that you are a team developing a new mobile operating system for educational tablets. Discuss and answer the following questions:
    (a) Who are the primary users of this tablet OS (students, teachers, both)? What are their specific needs and priorities?
    (b) How will you ensure the security and privacy of student data stored on the tablets?
    (c) How can the OS handle multiple apps running simultaneously while maintaining efficient usage of computer resources such as CPU and memory?
    (d) Design a user-friendly and intuitive interface suitable for users of different ages and tech experience levels. Sketch your ideas or create a simple mock-up.
    (e) How can the OS be designed to adapt to future technological advancements and integrate with emerging learning technologies?

FOR ONLINE READING ONLY

14. Choose one of the following statements and prepare arguments for or against it:
    (a) Virtual memory is always a better alternative to increasing physical RAM.
    (b) Round Robin scheduling is the most efficient CPU scheduling algorithm for all scenarios.
    (c) Operating systems should prioritize user convenience over system efficiency.
15. Conduct the research on emerging operating system trends related to artificial intelligence, cloud computing, Internet of Things (IoT) platforms or mobile technology. Create a short presentation or infographic explaining your findings and how these trends might impact future OS functionality.



Project: *Exploring and Comparing Operating Systems*

Project Objective

To explore and compare different operating systems across various devices. Focus on primary operating systems (such as Windows, macOS, Android, iOS, Linux) and compare two similar OS versions (such as Windows 10 vs. Windows 11). Research specific features and how different OS handle them.

Project Tasks:

Step 1: Research the OS of Your Choice

(a) **Target Devices:** Identify the types of devices that typically use this OS (desktop, mobile, embedded systems).

(b) **Core Functionalities:** Investigate how the OS handles memory management, CPU scheduling, process management, and security.

(c) **User Interface:** Examine how the OS presents information and allows users to interact with the system.

(d) **Strengths and Weaknesses:** Analyse the main advantages and disadvantages of this OS compared to others.

Step 2: Specific Feature Comparison

Choose a specific feature (such as security measures, virtual assistants) and compare how different OS handle it.

Project Deliverables

Write a five-page individual report summarizing what you have learned and the insights gained from the project.

FOR ONLINE READING ONLY

Databases and Database Management Systems

Introduction

Database systems are integral part of your daily living, impacting everyday tasks such as banking, travel reservations, e-commerce, and supermarket inventory control. They play a vital role in diverse fields like business, government, healthcare, and education. In this chapter, you will learn about the basic concepts of Databases and Database Management Systems; Database models, design and querying; and Database Management Systems. You will also learn about the emerging database models, technologies, and applications. The competencies developed will enable you to create databases.

Think:

Librarians and cataloguing systems in a large library

Basic Concepts of Databases and Database Management Systems

Scenario 8.1: *Traditional paper base for academic records management*

The new form five students have just reported to a school for high school admission. They are disturbed by the amount of paperwork the school uses to manage the students' academic records, including continuous assessments and certificates. In a move to modernize information management, the school has decided to implement a database system for managing academic records. Use the following questions to explore what this transition means and its impact on the new Form Five students and others at the school and beyond.

Questions:

1. What challenges might the new students face in managing their academic records using traditional paper-based methods?

FOR ONLINE READING ONLY

2. How can a database system streamline the process of managing academic records compared to manual methods?
3. What are the essential features and functionalities that a database designed for academic records management should possess?

Database system vs. file-based system

A database is a structured collection of related data, organised for easy retrieval, reflecting real-world information accurately. For example, names, phone numbers, and addresses stored in a mobile phone's database software or using software like Microsoft Access or MySQL. A database must be accurate and reliable, updating immediately as changes occur. This is true for complex databases like government payment systems, social media platforms, e-commerce sites, and search engines, which handle millions to billions of constantly changing records.

In contrast, a file-based system organises small groups of data and is defined in application manuals for specific functions. The following are the main differences, which are also shown in Figure 8.1(a) and (b)

(a) **File-based system**: Organises files on storage media like hard disks and flash drives. **Database System (DBMS)**: Software to access, create, and maintain databases.

(b) **Disaster recovery**: File-based systems lack this mechanism, while DBMS provides it.

(c) **Transaction transfer**: Complex in file-based systems but easy with DBMS using SQL.

(d) **Concurrency**: Not offered in file-based systems; provided by DBMS.

(e) **Data redundancy and inconsistency**: File-based systems suffer from redundancy and inconsistency as the same data is stored in different programs. DBMS reduces these issues by centralizing data storage, allowing consistent updates accessible by all applications.

The following summary highlights the key differences and advantages of using a DBMS over a file-based system.

FOR ONLINE READING ONLY

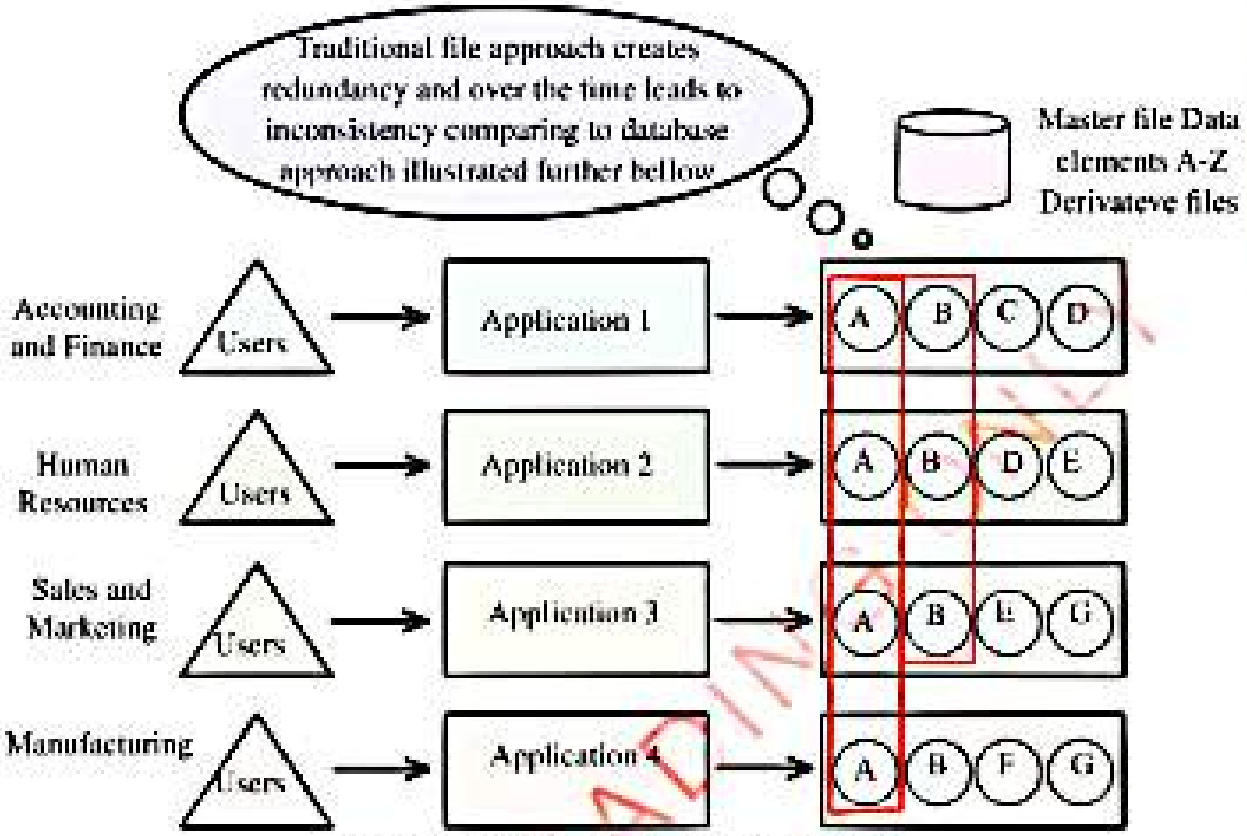


Figure 8. 1(a): *The traditional file approach*

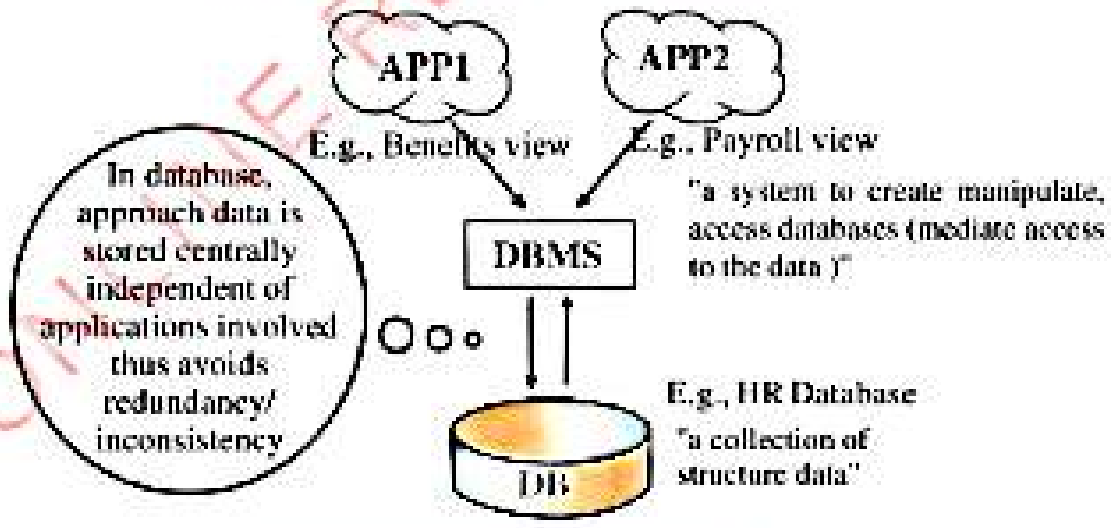


Figure 8.1 (b): *The database approach*

Activity 8.1:

Exploring information storage systems at your school

Aim:

To explore and compare different storage methods—physical files, database systems, and file-based systems—to understand their advantages and disadvantages.

FOR ONLINE READING

Materials: Physical files (such as student records, attendance sheets); access to the school's database system; and access to a file-based system

Instructions:

1. **Physical files:**
   (a) Observe how physical files are organised and stored.
   (b) Note the ease of access and retrieval of information.
   (c) Identify any potential issues, such as misplacement or damage.

2. **Database system:**
   (a) Explore the school's database system, focusing on its structure and organisation.
   (b) Examine how efficiently data is stored, retrieved, and managed.
   (c) Highlight key features such as search functionality, data integrity, and security measures.

3. **File-based system:**
   (a) Investigate the structure and organisation of a file-based system used in the school.
   (b) Note the limitations and challenges, such as data redundancy, consistency, and security issues.

Deliverables

Write a brief report comprising of the following:

(a) Summary of your observations and key findings for each storage method.

(b) Comparison of the pros and cons of physical files, database systems, and file-based systems.

(c) Real-world applications and examples where each system might be most effective.

Finally, document your findings and compiled report in a portifolio.

Exercise 8.1

1. Why is immediate and accurate data updating crucial for complex databases like government payment systems and social media platforms?
2. How does concurrency differ between file-based systems and DBMS, and why is it important?
3. Explain how data redundancy and inconsistency can be a problem in a file-based system compared to a DBMS. Provide examples.
4. How does a DBMS handle concurrent access to data, and why is this important? Contrast this with the concurrency control mechanisms in a file-based system.
5. Explain how a DBMS reduces data redundancy and inconsistency compared to file-based systems, citing real-world examples.

FOR ONLINE READING ONLY

(b) Network database model

The network database model is designed to flexibly represent objects and their relationships. Data is organised more like a network in this model, and many parent nodes are allowed as shown in Figure 8.3. The network model extends the hierarchical structure since it provides many-to-many relationships to be managed in a tree-like structure with multiple parents. Because more associations are developed in this database model as the data become increasingly connected, accessing the data becomes quicker and faster, and redundancy is managed more efficiently than in the hierarchical approach. Because of this, the network database model was the most extensively used database model before the introduction of the relational model.

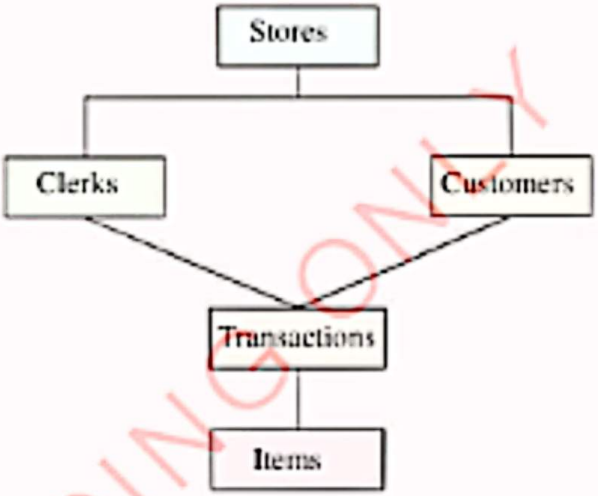
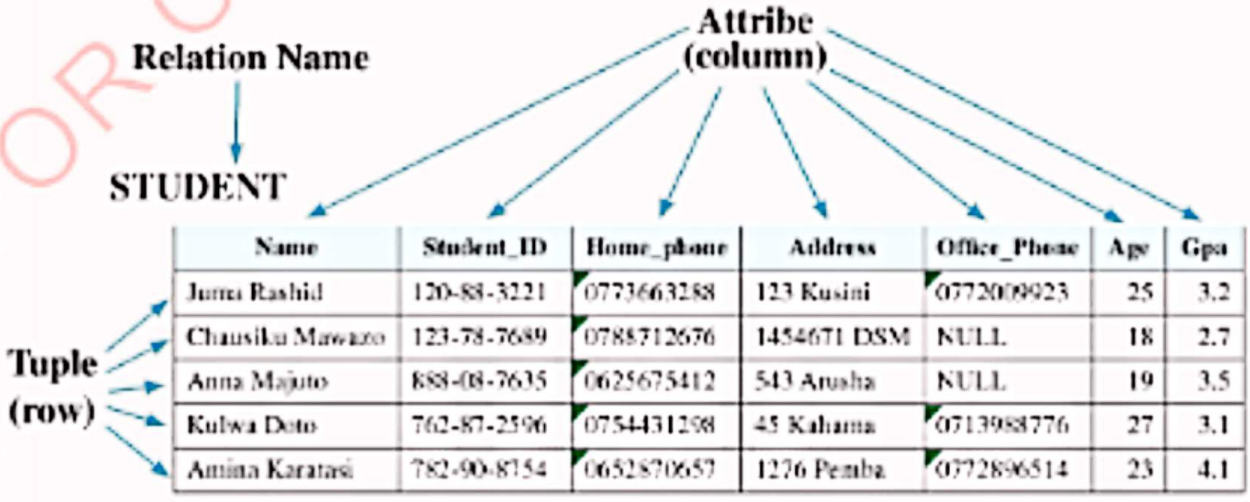


Figure 8.3: *An example of a network database model*

(c) Relational database model

A relational database model represents data in which all items are organised into two-dimensional tables known as relations. It is currently the most extensively used model. In this paradigm, all data are kept in rows and columns in a table. Thus, compared to network and hierarchical models, the basic structure of a relational database model is simple, and it is readily scaled up because we may add as many rows and columns as needed. For example, in the student table, shown in Figure 8.4 you can see columns (attributes or fields) and rows (tuples or records).



Name	Student_ID	Home_phone	Address	Office_Phone	Age	Gpa
Juma Rashid	120-88-3221	0773663288	123 Kusini	0772009923	25	3.2
Chausiku Mawazo	123-78-7689	0788712676	1454671 DSM	NULL	18	2.7
Anna Majuto	888-08-7635	0625675412	543 Arusha	NULL	19	3.5
Kulwa Doto	762-87-2596	0754431298	45 Kahama	0713988776	27	3.1
Amina Karatasi	782-90-8754	0652870657	1276 Pemba	0772896514	23	4.1

Figure 8.4: *The attributes and tuples of a student table*

FOR ONLINE READING ONLY

(d) Object-Oriented database model

In contrast to relational databases, which are table-oriented, an object-oriented database architecture represents both the data and the relationship in a single structure called an object. The object-oriented data model better represents real-world problems. It allows us to store music, video, photographs, and other data types in databases, which was not possible or efficient in the relational paradigm. Furthermore, in this object-oriented database architecture, two or more objects with various properties and methods are linked together, as in today's object-oriented databases and programming languages. These links are used to connect one thing to another. Figure 8.5 shows an example in which each object has its properties and methods. For example, consider the employee object, "Salary" is one of the employee object properties, while "Get hired" is one of the methods.

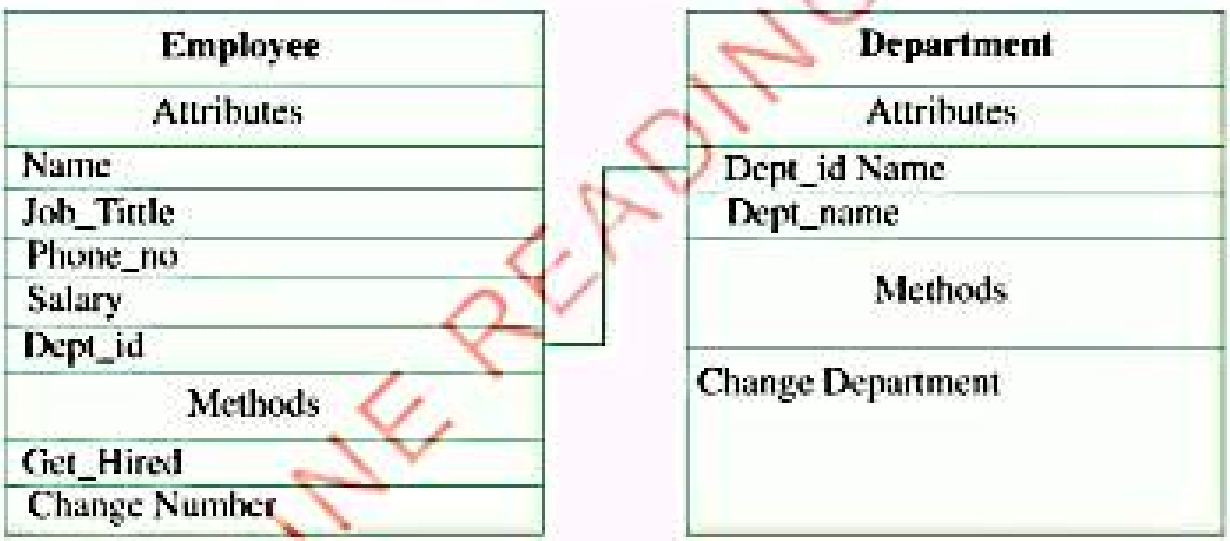


Figure 8.5: *An example of object-oriented database model implementation*

(e) Entity-Relationship database model

The Entity-Relationship (ER) database model is a conceptual framework used for database design, illustrating the database's structure through entities, relationships, and attributes. Here are key insights into the ER database model:

(i) *Entities:* Entities represent real-world objects or concepts stored in the database, such as people or products. They are organised into tables within the database.

(ii) *Relationships:* Relationships define how entities interact or associate with each other, including one-to-one, one-to-many, or many-to-many connections. For instance, a customer may have multiple orders, forming a one-to-many relationship.

(iii) *Attributes:* Attributes describe the properties or characteristics of entities, like names or ages. They are represented as columns within entity tables and can be simple or composite.

FOR ONLINE READING ONLY

(iv) *Entity-Relationship Diagram (ERD):* An ERD visually depicts the structure and connections of entities, using symbols like rectangles, diamonds, and ovals to represent entities, relationships, and attributes.

(v) *Normalization:* The ER model aids in normalization, a process that minimizes data redundancy and ensures data integrity by organizing data into multiple tables and establishing relationships.

(vi) *Database design:* Serving as a blueprint, the ER model guides the design of the database, enabling designers to understand entity structures and relationships before implementing tables, columns, and connections in the database.

(vii) *ER diagram tools:* Various tools facilitate the creation and visualisation of ER diagrams, offering graphical interfaces for designing, modifying, and documenting database structures. These tools assist in creating clear representations of the database design.

(f) Flat file database model

In the flat file model, all the data is stored in a single table, similar to a basic list. Think of a student's list with just student names. While relatively easy to create, limitations arise when dealing with complex relationships between data points. For example, a flat file list would not show which student belongs to which combination in the student's database example. While convenient for basic situations, flat files become cumbersome for managing intricate data structures.

Exercise 8.2

1. Explain the concept of a database model and explain its significance in contemporary information systems.
2. How do hierarchical and network models differ, and provide practical examples of their applications?
3. Compare and contrast relational and object-oriented models, and cite real-world examples.

Database design

Database design is a fundamental topic that involves creating a detailed data model of a database. This data model outlines how data is stored, organised, and manipulated, allowing for efficient data retrieval, business operations, decision-making, and data analytics. The data models are also crucial in the design and implementation of databases as they provide a clear framework for how data is connected and how it flows through systems, helping in designing databases that are efficient, reliable, and scalable, and ensuring data integrity and consistency. They are categorized into a conceptual data model, logical data model and physical data model. These detailed data models are data models that are specified data structures and business rules in a relational database model design.

FOR ONLINE READING ONLY

Relational database

A relational database is a collection of data elements linked by predefined connections, organised into tables with columns and rows. It divides data into tables that can be connected based on shared data, allowing the creation of new tables from existing ones through queries. This structure enhances understanding of data relationships and supports better decision-making. For example, a school database might include tables for pupils, family members, results, subjects, and teachers, connected by columns like Pupil_ID as shown in Figure 8.6. Queries can link these tables to generate reports, such as consolidated pupil results. Relational databases are widely used for managing inventories, sales, finances, treatments, and more. This section will cover key concepts such as entity-relationship models, functional dependencies, and normalization.

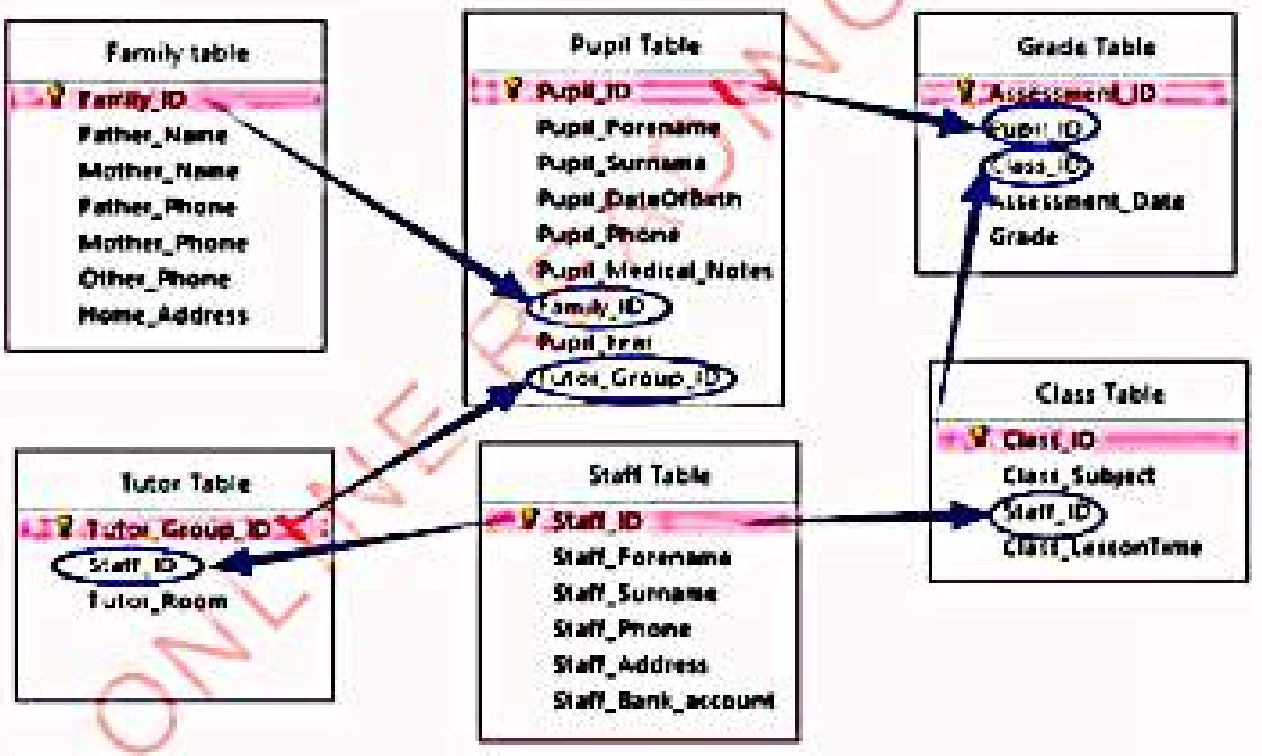


Figure 8.6: *Simple relational database example – Pupil Database*

Relational database model

A relational database model uses predefined relationships between data items, guided by structured data models that progress from conceptual to logical to physical stages. These models provide a framework for organizing and accessing data accurately and consistently, defining essential structures, rules, and naming conventions for database development. Visual representations illustrate how data elements connect, addressing questions such as who, what, where, and why. For instance, in retail transactions, the model includes customer details, product specifics, and purchase history, starting with conceptual models for user needs, advancing through logical models, and finalizing with detailed physical database.

FOR ONLINE READING ONLY

(a) *Conceptual data model*

The conceptual data model offers a high-level overview of organisational data, concentrating on business concepts, rules, and processes without addressing technical implementation details. It defines the structure of information needed for business operations, analytics, and reporting, identifying essential data and tracking business events and outcomes. Key characteristics include:

(i) Independence from hardware and software specifications

(ii) Comprehensive coverage of business concepts across the organisation

(iii) Design tailored for a business audience

(b) *Logical data model*

The logical data model explore deeper into the company's processes and their interconnections, defining data structures such as entities, tables, and relationships. Its goal is to define data independently of implementation, making it adaptable to various database systems and file structures. Key characteristics include:

(i) Independence from specific DBMS

(ii) Capability to integrate with other logical data models

(iii) Precise data attribute specifications

(iv) Adherence to normalization processes up to the Third Normal Form (3NF)

(c) *Physical Data Model*

The physical data model details the design and implementation of data in a database or file format for a specific system. It represents actual data storage within a DBMS, considering technical and performance factors. Derived from conceptual and logical models, it includes entities, relationships, data types, keys, and other DBMS features essential for application development. Key aspects include:

(i) Project-specific data representation

(ii) Dependence on a particular DBMS version or technology

(iii) Cardinality and nullability in table relationships

(iv) Precise data type assignments

(v) Definition of keys, views, indexes, and access profiles

The three data models, shown in Figure 8.7, use Entity Relationship (ER) modeling to design a robust relational database structure. Entities represent real-world objects with common attributes, while relationships describe data sharing among these entities.

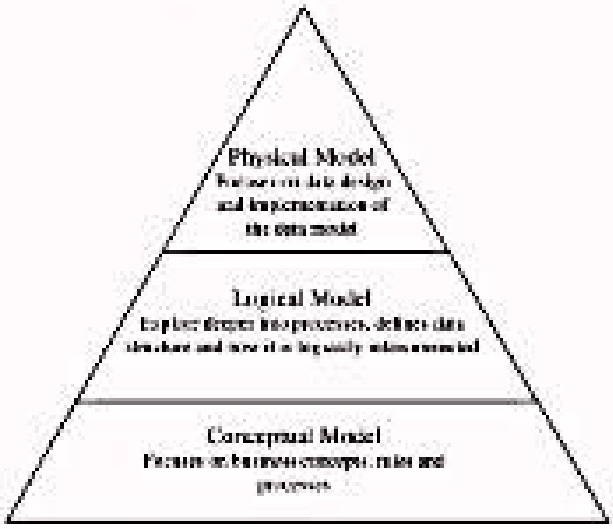


Figure 8.7: *Conceptual, logical and physical data models*

FOR ONLINE READING ONLY

Example 8.1

Create the three data models from a real-life scenario.

Scenario: A route taken by a particular bus company, whose buses travel according to a given schedule.

Step 1: *Creating a conceptual data model*

In this step, you have to identify entities and their relationships. There are three entities, which are: Route, Bus Company, and Schedule. Concerning relationships, one route can have one or many bus companies offering transportation services to passengers. Also, a route can have one or multiple transportation schedules.

Step 2: *Creating a logical data model*

Identification of characteristics, primary keys, and foreign keys are necessary for the establishment of a logical data model in addition to the names of entities and relationships that were formed between entities in step 1.

(a) Identification of attributes

(i) The entity Route may have the following attributes: route number, route name, origin, and destination.

(ii) The entity Bus Company may have the following attributes: company id, route number, owner name, and postcode.

(iii) The entity Schedule may have the following attributes: schedule number, route number, departure time, and arrival time.

(b) Primary keys

Under the current scenario, the primary keys are route number, company ID, and schedule number for entities Route, Bus Company, and Schedule, respectively.

(c) Foreign keys

According to the relationships established in Step 1, the route number from the entity Route will become a foreign key in the remaining entities, which are Bus Company and Schedule. This implies that there shall be an addition of one attribute (route number) in the Bus Company and Schedule entities.

Step 3: *Creating a physical data model*

The creation of the physical data model requires defining the names of the tables, names of the columns, and column data types in addition to the characteristics, primary key, and foreign keys described in the logical data model in step 2. Since the physical data model is also an implementation of the database, its specification may be dependent on a particular database management system such as MS

FOR ONLINE READING ONLY

Access, MySQL, and PostgreSQL.

(a) Names of tables

The names of the tables are *route*, *bus_company*, and *schedule*

(b) Names of columns

(i) The table *route* has the following columns: *route_number, route_ name, origin, destination.*

(ii) The table bus_company has the following columns: *company_ id, route_ number, owner_name, post_code.*

(iii) The table schedule has the following columns: *schedule_number, route_ number, departure_time, arrival_time.*

(c) Data types

(i) The data type for *route_number, company_id, schedule_number, and post_code* will be integer.

(ii) The data type for *route_name, origin, destination, and owner_name* will be characters of varying length (preferably 50 characters).

(iii) The data type for *departure_time* and *arrival_time* will be date time or time (depending on your preference).

Note: The physical data model and relationships between entities will be covered in more detail in this chapter's subsequent sections.

Activity 8.2:

Designing a Data Model

Following the real-life example provided, design a data model for a shop encompassing conceptual, logical, and physical data models. Focus on three key entities.

Relational database terminologies and their application

As mentioned earlier, the relational database is a type of database that focuses on the relationship between stored data elements. The fundamental terminologies applied to relational databases include attribute, tuple, relation, domain, primary key, candidate key, alternate key, foreign key, selection, and projection.

(a) Attribute or field

Attributes or fields describe characteristics or properties that define all items about a particular category. An attribute is also a named column in a database, e.g., first name, postal code, and date of birth in the employees' database.

(b) Tuple or record

A tuple or record is a data set representing a single item. It is a collection of related data fields or attributes. It is also known as a row in a database. An example of a record is, for instance, when you have an employee's database with attributes first_name, post_code, and date_of_

FOR ONLINE READING ONLY

birth whose values may be Juma, 11101 Kivukoni, and 2/3/1980, respectively.

(e) Relation

It is a set of tuples or records with the same attributes; a set of columns and rows as in Figure 8.8. Also, we refer to it as an entity that is a generalised class of people, places, concepts, or things for which data is collected, stored, and maintained. The entity is also named Table in a database. For example, a school database may include Students, Classes, and Courses.

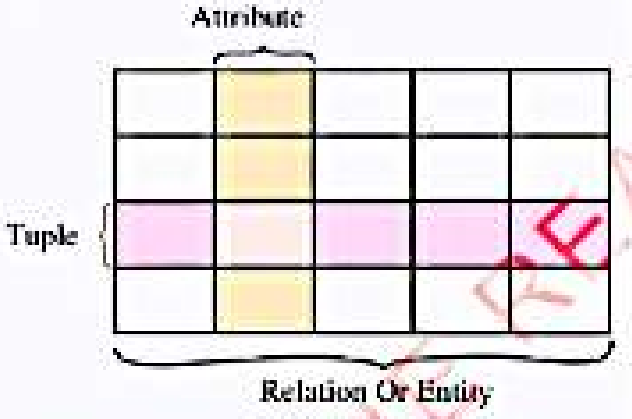


Figure 8.8: *Illustration of some relational database basic terminologies*

(d) Domain

The domain is a set of allowable values for an attribute. It refers to a group of values we can assign to an attribute and can be considered as a constraint on the attribute's value. Each attribute has a domain. For example, for an attribute "Grade" in a student database, its domain can be a positive number between 0 and 100. Another example could be an attribute "Regions" in the Tanzania census database in which by March 2022, there were 31 regions (26 from Tanzania mainland and 5 from Zanzibar); thus, the domain alphabetically is "Arusha, Dar es Salaam, Dodoma, Kaskazini Pemba, Kusini Unguja , Tabora, Tanga".

(e) Primary key

The primary key is a tuple, field, or set of tuples or fields that uniquely identify a record or tuple concerning a table. A table can have only one primary key, often denoted as PK. The primary key cannot be null due to the dependency of other fields concerning a table. A good example is a National Identification Number in the National Identification Authority (NIDA) database or a student registration number in a student database, as indicated in Figure 8.9 as "StudID."



Figure 8.9: *Example of the primary, candidate and alternate keys using a student*

FOR ONLINE READING ONLY

(f) Candidate key

A candidate key, denoted as K, is a minimal set of attributes that uniquely identify a row in a table. Each set of attributes has unique values per row, and removing any attribute can cause duplicate rows. Unlike the primary key, which uniquely identifies a record with only one key per table, a candidate key can have multiple keys per table. For example, in Figure 8.9, the columns StudID, Roll_Number, and email can all be candidate keys as they contain unique values and can be used to identify data. Any of these can be chosen as the primary key.

(g) Alternate key

The candidate keys that are not selected as the primary key are called alternate keys. In this kind of key, columns have a candidate key except for the primary key, which has an alternate key. For example, in Figure 8.9, the columns - Roll_No and Email are alternate keys, considering that Student ID is the primary key.

(h) Foreign key

A foreign key is an attribute or set of attributes in one table that matches the primary key of another table, creating a relationship between them. It uses values from the referenced table to restrict the domain of attributes in the referencing table, maintaining data integrity and enabling navigation between different entities. For example, in an Employee database, each employee works in a specific department. Instead of storing department information in the employee table, we link the two tables by adding the primary key of the DEPARTMENT table, Department_ID, as a foreign key in the EMPLOYEE table, establishing the relationship as shown in Figure 8.10.

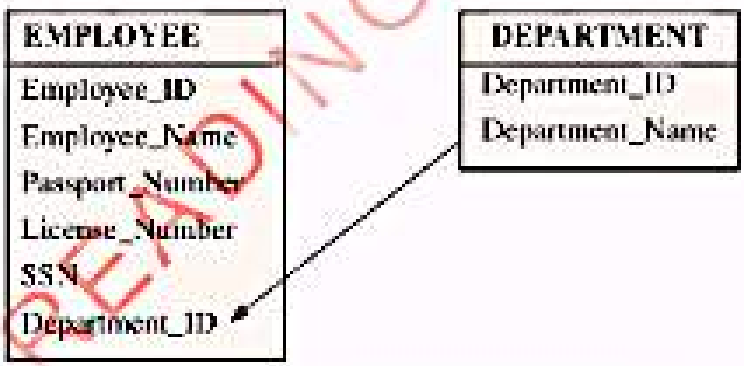


Figure 8.10: *Example of the foreign key using an employee database*

(i) Composite Key

A composite key in relational databases is a combination of two or more columns that uniquely identify each row in a table. Uniqueness is guaranteed only when the columns are combined, not individually. Composite keys are a type of candidate key formed by multiple columns. Note that a composite key can be made from multiple candidate keys and cannot be empty or null.

Activity 8.3:

Identify keys in the tables

Aim:

Identify and understand the different types of keys (such as Primary, Foreign, Candidate,

FOR ONLINE READING ONLY

and Alternate keys) in a relational database by examining the student registration and results tables in your school.

Materials:

Database Management System (DBMS) Software (Such as MySQL, PostgreSQL, or SQLite); Access to Student registration and results tables from your school's database or a sample dataset provided for the activity; Diagramming Tool (such as Lucidchart, Draw.io, or even a pen and paper to create visual representations of the tables and their relationships); Sample Data or Dataset (gather or simulate sample data for student registration and results tables).

Instructions:

Consider the related tables of student registration and results in your school. Identify and illustrate them with a diagram.

Step 1: Examine the tables

(a) *Student registration table:* This table contains information about the students.

(b) *Student results table:* This table contains information about the students' results.

Step 2: Identify the Keys

(a) *Primary key*: The unique identifier for each row in a table.

(b) *Foreign key:* An attribute that creates a relationship between two tables.

(c) *Candidate key*: A set of attributes that can uniquely identify a row in a table.

(d) *Alternate key*: Candidate keys that are not chosen as the primary key.

Step 3: Illustrate the Keys using diagram

(a) Draw a diagram to show the relationship between the Student Registration and Student Results tables.

(b) Indicate the Primary, Foreign, Candidate, and Alternate keys.

Deliverables:

Write a brief report on the relational database keys identified from your school databases, including the screenshots. Document your findings in a portfolio.

Selection and projection

Selection and projection operations are basic and most used operations specific to relational databases. The selection is defined as taking the horizontal subset of rows of a single table that satisfies a particular condition. It retrieves tuples from a relation, limiting the results to only those that meet a specific criterion. Selection answers the question: which rows shall the query return (where condition)? For example, `**Select * From emp Where deptno = 10**` as shown in Figure 8.11.

FOR ONLINE READING ONLY

EMPNO	FNAME	JOB	HIREDATE	SAL	DEPTNO
7782	Juma	Manager	09-june-2001	2,800,000	10
7839	Asha	Data Clerk	17-nov-2001	950,000	10
6289	Mariam	Analyst	23-jan-2002	2,250,000	10
5234	John	Data clerk	02-apr-2001	975,000	20
5235	Pima	Analyst	03-dec-2001	2,450,000	30
3255	Asha	Director	23-may-2007	4,000,000	40
2646	Zuhura	Manager	20-may-2010	3,000,000	20
5433	Peter	Ass. Director	30-jan-2009	3,800,000	20
2555	Jim	Analyst	06-sept-2011	2,100,000	30
2434	Shukran	Data clerk	02-oct-2010	950,000	30
9546	Hashim	Salesperson	23-may-2007	2,500,000	20
4356	Sura	Salesperson	06-sept-2011	1,500,000	30
4666	Husna	Manager	02-aug-2010	2,650,000	20
7677	Ansel	Ass. Director	23-may-2007	4,000,000	20

Figure 8.11: *Selection operation in a relational database*

The projection is defined as the set obtained when the components of the tuples are restricted to the set, it discards (or excludes) the other attributes. It extracts only the specified relation attributes from a tuple or set of tuples. It is a vertical subset of table columns, and it may include one column, two columns or all table columns. Projection answers the question: which columns shall the query return? For example, `Select empno, fname, job From emp`; returns columns as shown in Figure 8.12.

EMPNO	FNAME	JOB	HIREDATE	SAL	DEPTNO
7782	Juma	Manager	09-June-2001	2,800,000	10
7839	Asha	Data Clerk	17-nov-2001	950,000	10
6289	Mariam	Analyst	23-jan-2002	2,250,000	10
5234	John	Data clerk	02-apr-2001	975,000	20
5235	Pima	Analyst	03-dec-2001	2,450,000	30
3255	Asha	Director	23-may-2007	4,000,000	40
2646	Zuhura	Manager	20-may-2010	3,000,000	20
5433	Peter	Ass. Director	30-jan-2009	3,800,000	20
2555	Jim	Analyst	06-sept-2011	2,100,000	30
2434	Shukran	Data clerk	02-oct-2010	950,000	30
9546	Hashim	Salesperson	23-may-2007	2,500,000	20
4356	Sura	Salesperson	06-sept-2011	1,500,000	30
4666	Husna	Manager	02-aug-2010	2,650,000	20
7677	Ansel	Ass. Director	23-may-2007	4,000,000	20

Figure 8.12: *Projection operation in the relational database*

Projection differs from selection as SELECT is used to select all rows of a specific relation that meet a particular condition, whereas the project is used to select particular columns. However, one query may contain both: selection and projection: For example, `**select empno, fname, job From emp Where deptno=10**`; returns enclosed columns and rows as in Figure 8.13.

FOR ONLINE PREVIEW ONLY

EMPNO	FNAME	JOB	HIREDATE	SAL	DEPTNO
7782	Juma	Manager	09-june-2001	2,800,000	10
7839	Asha	Data Clerk	17-nov-2001	950,000	10
6289	Mariam	Analyst	23-jan-2002	2,250,000	10
5234	John	Data clerk	02-apr-2001	975,000	20
5233	Pima	Analyst	03-dec-2001	2,450,000	30
3255	Asha	Director	23-may-2007	4,000,000	40
2646	Zubura	Manager	20-may-2010	3,000,000	20
5433	Peter	Ass. Director	30-jan-2009	3,800,000	20
2535	Jim	Analyst	06-sept-2011	2,100,000	30
2434	Shukran	Data clerk	02-oct-2010	950,000	30
9546	Hashim	Salesperson	23-may-2007	2,500,000	20
4356	Sura	Salesperson	06-sept-2011	1,500,000	30
4666	Busara	Manager	02-aug-2010	2,650,000	20

Figure 8.13: *Selection and Projection operations in a relational database*

FOR ONLINE READING ONLY

Activity 8.4:

Applying selection and projection Operations

Aim:

To practice and understand database query operations such as selection and projection through practical application and analysis of SQL queries on a specified dataset.

Materials:

1. Computer with database management system (DBMS) software (e.g., MySQL, PostgreSQL)
2. Access to the "emp" table or relevant dataset used in Figure 8.13
3. Pen and paper or digital tools for documenting query results

Instructions:

Implement the concepts of selection and projection by analyzing the query "Select Fname, job, hiredate, sal From emp Where deptno=30" based on Figure 8.13. Visualise and interpret the resulting dataset in a structured format similar to the referenced figure.

Step 1: Understanding Selection and Projection

Familiarize yourself with the concepts of selection (filtering rows) and projection (selecting specific columns) in database queries.

Step 2: Analysing the query

Study the SQL query "Select Fname, job, hiredate, sal From emp Where deptno=30" and its components. Identify the columns being selected (projection) and the condition for filtering rows (selection).

Step 2: Executing the query

Use a DBMS software to execute the SQL query on the "emp" table or relevant dataset. Retrieve and record the results that match the criteria "deptno=30".

Step 3: Formatting results

Present the query results in a tabular format similar to Figure 8.13. Include columns for "Fname", "job", "hiredate", and "sal", with each row representing an employee meeting the selection criteria.

Deliverable:

Submit a comprehensive documentation of the process and results in your report or portfolio. Include a brief analysis of the selected dataset, highlighting any observations or insights gained from the exercise. Present the query results in a structured table format similar to Figure 8.13, demonstrating proficiency in selection and projection operations within a database context.

Advantages of relational database model

A relational database system has multiple advantages, as described in the next paragraphs.

(a) *Simple model:* Easy to structure and query without complex processes, unlike hierarchical and network models.

(b) *Data accuracy:* Utilizes primary and foreign keys to relate tables, reducing data duplication and enhancing accuracy.

(c) *Easy data access:* No fixed pattern for accessing data; users can query any table or combination of tables effortlessly.

(d) *Data integrity:* Ensures robust data entry and validation, maintaining relational reliability and preventing isolated or unrelated records.

(e) *Flexibility:* Can scale and adapt to changing needs with no limits on rows, columns, or tables, allowing efficient updates and deletions.

(f) *Normalisation:* Ensures database structure integrity by eliminating variances and achieving normalization, typically up to the third normal form.

(g) *High Security:* Allows selective confidentiality for tables, restricting access based on user roles.

(h) *Future modifications:* Easily accommodates new requirements, allowing straightforward insertion, deletion, or updating of records.

Examples of relational database systems

Relational database systems are used in many organisations today. There are several common and popular relational database systems based on: end-users, open source, and proprietary, as follows:

(a) *Popular DBMSs for end-users:* The popular and most used relational

FOR ONLINE READING ONLY

databases in this end user's utilisation category are Microsoft Access and File MakerPro.

(b) *Popular open-source DBMSs for organisations:* The popular and most used relational database in this category is MySQL and PostgreSQL.

(c) *Popular proprietary DBMSs for organisations:* The popular and most used relational databases in this category are Microsoft SQL Server, Oracle, IBM-DB2 and recently AWS - Amazon Relational Database Service (Amazon RDS), which makes it easy to set up, operate, and scale a relational database in the cloud.

Exercise 8.3

1. Identify two related entities in your school students' organisation and determine three possible attributes for each. Create both tables with three tuples for each and show how they relate.
2. In the tables you have created in question (2), identify the primary key, candidate key, alternate key and foreign key. Also, determine the domain for each attribute that was identified.
3. Differentiate between selection and projection in an RDBMS while giving examples.

Entity-Relationship model

Entity-relationship (ER) modelling is a best practice for producing well-designed databases. It depicts the structure of a relational database allowing you to understand the data and how it shares information.

Elements of the entity-relationship model

An ER data model's main elements involve entities, attributes, and relationships, as depicted in Figure 8.14, with the example from a student database involving how each element is diagrammatically represented.

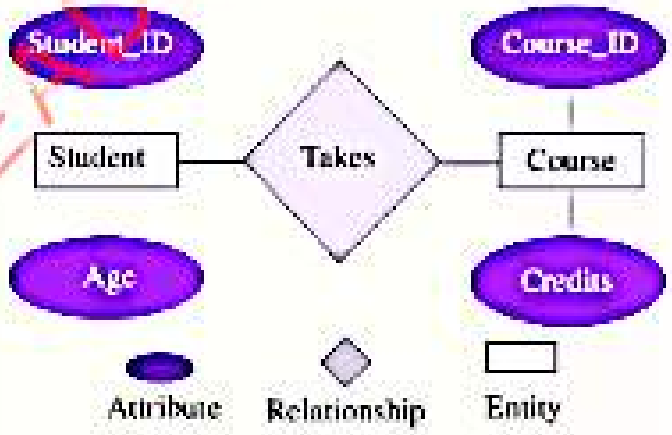


Figure 8.14: *A sample of entities, attributes, and relationships from a student*

Entity: An entity is an object representing things from the real world. An entity contains a set of attributes, each with values.

Attribute: The attribute is the characteristics or properties of an entity based on the data type. Each column represents an attribute that stores the value of an entity characteristic.

Relationships: Relationships define how entities are related to each other; and how they share information in the database. This is done

FOR ONLINE READING ONLY

through Entity-Relationship Diagrams (ERD). ERD uses basic graphical symbols to show the organisation of entities and their relationships. An important aspect of relationships is cardinality which defines how the entities' data or records are related. The common cardinalities include one-to-one, one-to-many, many-to-many, and many-to-one relationships. Details for each relationship are outlined as follows:

(a) *A one-to-one relationship* is when one entity from entity set A can be associated with at most one entity of entity set B and vice versa. For example, a school might decide to assign one class teacher to precisely one class; thus, one teacher manages only one class.

(b) *A one-to-many relationship* is when one entity from entity set A can be associated with multiple entities of entity set B. Still, an entity from entity set B can be associated with at least one entity in set A. For example, the relationship between a teacher and the subjects they teach, where one teacher can teach multiple subjects, but only one teacher teaches a given subject.

(c) *A many-to-one relationship* is when more than one entity from entity set A can be associated with at most one entity of entity set B. However, an entity from entity set B may or may not be associated with more than one entity from entity set A. For example, many students belong to the same class.

(d) *A many-to-many relationship* is when one entity from set A can be associated with more than one entity from set B and vice versa. For example, students are related to multiple teachers, and teachers can be associated with multiple students.

Cardinality notation symbols

In cardinality notation symbols, the lines connecting entities appear with extra marks at the end. These marks define the cardinality degree. The notations are summarised as shown in Figure 8.15.

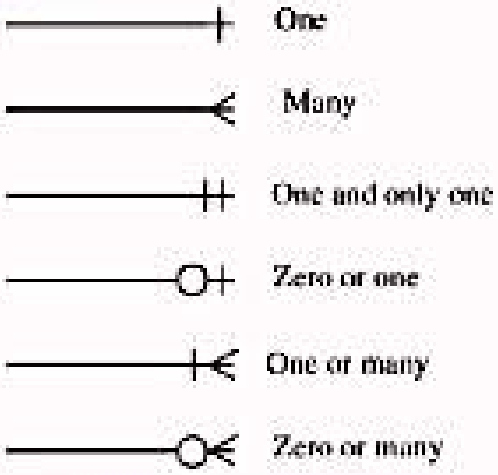


Figure 8.15: *Cardinality notation*

Generally, these cardinalities might apply in an ERD at the entity level. An example is the customer order database in Figure 8.16, in which a salesperson serves many customers and each can also place multiple orders. Each order can have only one invoice, but each of these orders may also include many line items, and each specifies the products that it contains.

FOR ONLINE READING ONLY

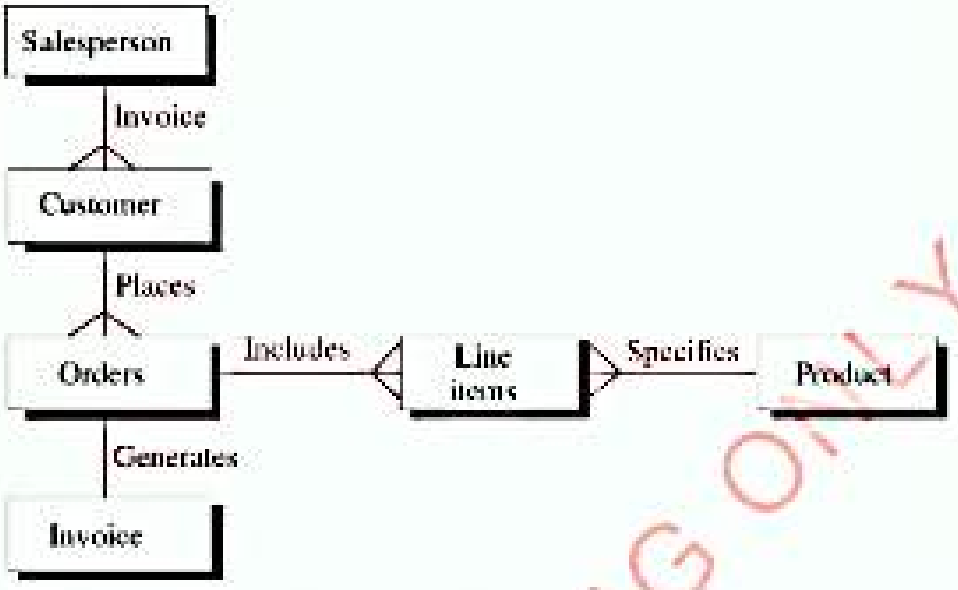


Figure 8.16: *An example Entity-Relationship Diagram (ERD) - customer order database*

Activity 8.5:

Construction of an E-R diagram to represent a real-life database

Aim:

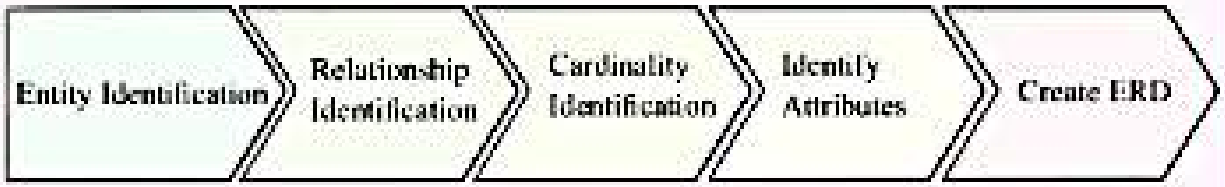
Develop an Entity-Relationship (E-R) diagram to model and visualise the relationships between students, courses, and professors in a university database scenario.

Materials Required:

1. Computer with ER diagramming software (such as Lucidchart, Microsoft Visio)
2. Access to a DBMS for conceptual modeling
3. Data or information about students, courses, and professors in a university setting

Instructions:

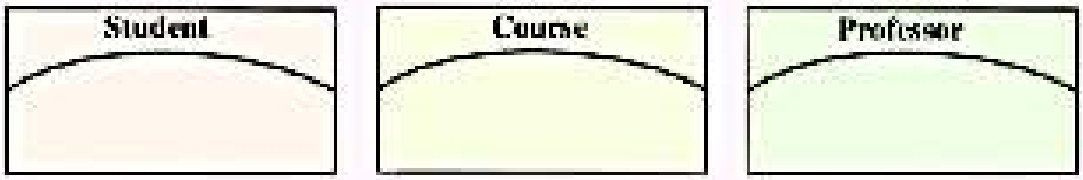
In a University, a student enrols in courses. A student must be assigned to at least one or more courses. To maintain instruction quality, a Professor can deliver only one course. Create an Entity Relationship Diagram (ERD) following the steps indicated in the diagram below and the steps that follow:



FOR ONLINE READING ONLY

Step 1: Entity identification

We have three entities: Student, Course and Professor.



Step 2: Relationship identification

Analyse the relationships between students, courses, and professors. Consider the constraints such as student enrollment in courses and professor assignment to courses. You have two relationships: The student is assigned a course, and the professor delivers a course.



Step 3: Cardinality identification

From the problem statement, we know that, a student can be assigned multiple courses, and at a time Professor can deliver only one course.



Step 4: Identify attributes

To identify attributes, you need to study the files, forms, reports, and data currently maintained by the organisation. You can also conduct interviews with various stakeholders to identify entities. Initially, identify the attributes without mapping them to a particular entity. Once you have a list of attributes; you need to map them to the identified entities. Ensure an attribute is to be paired with exactly one entity. If you think an attribute should belong to more than one entity, use a modifier to make it unique. Once the mapping is done, identify the primary keys. If a unique key is not readily available, create one.

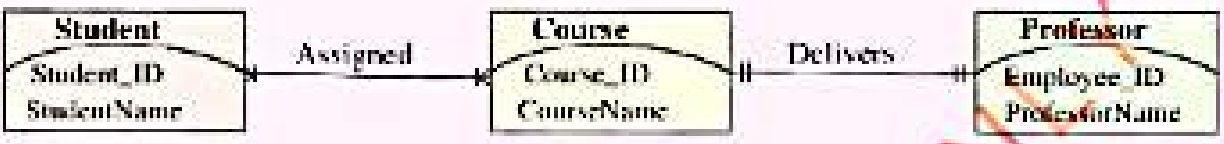
Entity	Primary Key	Attribute
Student	Student_ID	StudentName
Professor	Employee_ID	ProfessorName
Course	Course_ID	CourseName

FOR ONLINE READING ONLY

For the Course entity, more attributes could for example, be Duration, Credits, Assignments, etc. For ease, we have considered just one attribute for each entity.

Step 5: Create the ERD Diagram

A more modern representation of the ERD example.



Deliverables:

Submit the completed E-R diagram depicting the relationships between students, courses, and professors. Provide a brief description outlining the rationale behind your design decisions and how the diagram accurately represents the university database scenario. Document your work in a portifolio.

Functional dependency

(a) Meaning of functional dependency

Functional dependency is a constraint that determines how one attribute relates to another in a relational database. It means we can derive the value of one attribute from another. For example, knowing a student's roll number allows us to find their address and marks, meaning the address and marks are functionally dependent on the roll number. Similarly, knowing an employee's number lets us determine their name and salary, making the name and salary functionally dependent on the employee number. Table 8.1 illustrates this concept.

Table 8.1: *Functional dependency*

Employee Number	Employee Name	Employee Salary
EMP01	Diana	900,000
EMP02	Juma	950,000
EMP03	Andrew	850,000

(b) Importance of functional dependencies

(i) Functional dependency helps to maintain the quality of data in the database.

(ii) It helps to define databases' constraints, ensure data integrity, and avoid data anomalies.

(iii) It plays a vital role in finding the difference between good and bad database design, thus promoting robust and impactful databases.

(c) Types of functional dependencies

There are four main types of functional dependency which are: Multivalued dependency, Trivial functional dependency, Non-trivial functional dependency and Transitive functional dependency.

(i) *Multivalued dependency*

This type of functional dependency occurs when there are multiple independent

FOR ONLINE READING ONLY

multivalued attributes in a single table. It consists of at least two dependent attributes on its attributes; thus, it always requires at least three attributes. For example, consider a motorbike manufacturing company that produces two colours (black and red) in each model every year. Table 8.2 shows multivalued dependency.

Table 8.2: *Multivalued dependency*

Motorbike_ Model	Manuf_year	Colour
M1001	2017	Black
M1001	2017	Red
M2012	2018	Black
M2012	2018	Red
M2222	2019	Black
M2222	2019	Red

Here columns manuf_year and colour are independent of each other and dependent on motorbike_model. In this case, the two columns are said to be multivalued dependent on motorbike_model. These dependencies are represented like this: motorbike_model ->> manuf_year; and motorbike_model ->> colour.

(ii) Trivial dependency

This type of functional dependency occurs when a set of attributes called a trivial is included in that attribute. So, X -> Y is a trivial functional dependency if Y is a subset of X. For example, consider this Employee Table 8.3 with two columns, Emp_id and Emp_name.

Table 8. 3: *Trivial dependency table*

Emp_id	Emp_name
AS555	Harry
AS811	Mwajuma
AS999	Kelvin

In this case, {Emp_id, Emp_name} -> Emp_id is a trivial functional dependency as Emp_id is a subset of {Emp_id, Emp_name}.

(iii) Non-trivial dependency

This functional dependency occurs when A->B holds true where B is not a subset of A. In other words, if a functional dependency X->Y holds true, then Y is not a subset of X; thus, this dependency is called a non-trivial functional dependency. For example, an employee table with three attributes: emp_id, emp_name and emp_address. From this example, the following functional dependencies are non-trivial: emp_id -> emp_name (This means that if we know emp_id, we know emp_name, but emp_name is not a subset of emp_id) emp_id -> emp_address (This imply that, if we know emp_id, we know emp_address which is not a subset of emp_id).

(iv) A transitive dependency

It is a type of functional dependency that happens when it is indirectly formed by two functional dependencies, and can only occur with three or more attributes. Consider the following Table 8.4 containing the details of the company's Chief Executive Officers (CEOs).

FOR ONLINE PREVIEW ONLY

Table 8.4: Company chief executive officers' details

Company	CEO	BirthDate
Microsoft	Satya Nadella	01/2/1967
Google	Sundar Pichai	12/6/1972
Facebook	Mark Zuckerberg	07/9/1984

It follows that {Company} -> {CEO} (if we know the company, we know its CEO's name). {CEO} -> [Birthdate]; if we know the CEO, we know the Birthdate. Therefore, if the rule of transitive dependency: {Company} -> [Birthdate] should hold, it will make sense because if we know the company's name, we can know the CEO's birthdate.

Activity 8.6:

Exploring multivalued dependencies in databases

Aim:

Explore multivalued dependencies by creating and analyzing a table that exemplifies these characteristics.

Materials:

1. Computer with database software (Such as MySQL, PostgreSQL, or any DBMS tool)
2. Access to a DBMS
3. Data sets or sample data to create the table

Instructions:

Step 1: Study Multivalued Dependencies

Learn about multivalued dependencies where one attribute determines another, but the dependent attribute can have multiple independent values.

Step 2: Identify a Real-Life Scenario

Observe your environment to find an example where several independent attributes relate to a single entity.

Step 3: Create a Table

Based on your observation, design a single table that demonstrates multivalued dependency characteristics.

Deliverable

Submit the table demonstrating multivalued dependencies, including attributes and sample data. Provide a brief explanation of the scenario that inspired your table design and how multivalued dependencies are illustrated. Make sure you document your findings in a portifolio.

Normalisation

(a) Meaning of normalisation

Normalization organises data into cohesive groups and establishes relationships to minimize redundancy and prevent anomalies like insertion, update, and deletion issues. Its goal is to ensure data consistency, optimise data retrieval, and avoid the need for frequent database redesigns. By analyzing schemas based on functional dependencies and primary keys, normalization follows structured

FOR ONLINE READING ONLY

steps such as synthesis (creating new designs) or decomposition (improving existing designs). This approach reduces redundancy, anomalies, and inconsistencies while enforcing proper dependency constraints on attributes and tables to maintain data integrity.

For instance, normalization prevents anomalies like an employee with ID 426 having different addresses across multiple records, as seen in Table 8.5.

Table 8.5: *Data anomalies*

	Employees' Skills	
Employee ID	**Employee Address**	**Skill**
426	Box 137 MWANZA	Public speaking
426	Box 132 MWANZA	Typing
645	Box 8991 Dar es salaam	Carpentry
645	Box 8991 Dar es salaam	Public speaking

In this example, the same information can be expressed on multiple rows; therefore, updates to the relationship may result in logical inconsistencies. For example, each record in an "Employees' Skills" relation might contain an Employee ID, Employee Address, and Skill; thus, a change of address for a particular employee may need to be applied to multiple records (one for each skill). If the update is only partially successful, (the employee's address is updated on some records but not others), then the relation is left in an inconsistent state. Specifically, the relation provides conflicting answers to this particular employee's address, affecting the database's use and effectiveness.

(b) Database normalisation forms

Normalization is a methodical process that follows clear rules at each stage of data modeling. It involves various normal forms, each with specific criteria. The primary normal forms include First Normal Form (1NF), Second Normal Form (2NF), and Third Normal Form (3NF). These forms are designed to remove anomalies such as insertion, updating, and deletion issues once applied to a dataset.

Additionally, there is the Un-Normalized Form (UNF), which represents the starting point before applying normalization rules. It serves as the initial stage before detailed descriptions of the common normal forms (1NF, 2NF, and 3NF) are discussed, as illustrated in Figure 8.17.

FOR ONLINE READING ONLY

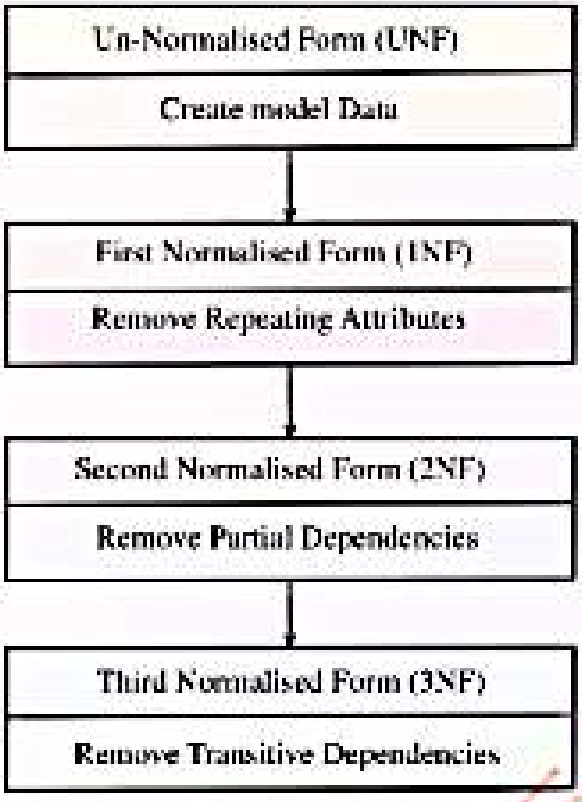


Figure 8.17: *Objective of UNF, and Three normal forms (1NF; 2NF; 3NF)*

(c) Un-Normalised Form (UNF)

Un- Normalization is a systematic process governed by clear rules throughout the data modeling stages. It encompasses several normal forms, each defined by specific criteria. Key normal forms include First Normal Form (1NF), Second Normal Form (2NF), and Third Normal Form (3NF), aimed at eliminating anomalies such as insertion, updating, and deletion issues once implemented on a dataset.

Additionally, the Un-Normalized Form (UNF) serves as the initial starting point prior to applying normalization rules. It represents the preliminary stage before delving into detailed descriptions of common normal forms (1NF, 2NF, and 3NF), as illustrated in Figure 8.17 and shown in Table 8.6.

Table 8.6: *An example of an unnormalised form table*

emp_id	emp_name	emp_address	emp_dept
101	Samson Madirisha	DSM	D001
101	Samson Madirisha	DSM	D002
123	Madina Wavu	Kagera	D890
166	Hashim Milango	Mbeya	D900
166	Hashim Milango	Mbeya	D004

In this unnormalized Table 8.6, there is an *update anomaly*. For instance, employee Samson appears in two rows because he belongs to two different departments. Updating Samson's address requires changing both rows to maintain consistency; otherwise, the data becomes incorrect. If one department's address is updated but not the other, the database would show conflicting addresses for Samson.

Additionally, this table may lead to an *insert anomaly*. If a new employee joins without being assigned to any department (emp_dept does not allow nulls), inserting their data into the table becomes impossible.

FOR ONLINE LEARNING ONLY

Lastly, there is a *delete anomaly* in this unnormalized table. If the company closes department D890, deleting rows with emp_dept as D890 would also remove employee Madina's information because she is only assigned to that department.

To resolve these issues, it's essential to normalize the data into the three normal forms introduced earlier. Another example of an Unnormarised Table is shown in Table 8.7.

Table 8.7: *Non-normalized table*

Full name	Address	Movie_industry	Sports
Ismail Makame	Kijichi	Hollywood and Bollywood	Football and Basketball
Rehema Kiongwe	Mnazi Moja	Swahiliwood and Nollywood	Netball
Muki Mohamed	Mbagala	Nollywood and Riverwood	Cricket

Advantages of normalization

(i) Normalization helps to reduce data redundancy
(ii) More incredible overall database organisation
(iii) Data consistency within the database
(iv) Much more flexible database design
(v) Enforces the concept of relational integrity

Disadvantages of normalization

(i) You cannot start building the database before knowing what the user needs.
(ii) The performance degrades when normalizing the relations to higher normal forms, that is, 4NF, and 5NF.
(iii) It is very time-consuming and challenging to normalize relations of a higher degree.
(iv) Careless decomposition may lead to a bad database design and severe problems.

First Normal Form (1NF)

As indicated in Figure 8.8, the objective of this 1NF is to remove repeating attributes. This is done because the relation is in 1NF if it contains an atomic value, and an attribute of a table cannot hold multiple values. Instead must have only single-valued attributes.

Also, 1NF disallows the multi-valued attribute, composite attribute, and their combinations. For a table to be in the First Normal Form, it should follow the following four rules:

(i) It should only have single (atomic) valued attributes/columns.
(ii) Values stored in a column should be of the same domain.
(iii) All the columns in a table should have unique names.
(iv) The order in which data are stored does not matter.

FOR ONLINE READING ONLY

For example: Suppose an agriculture-based company wants to store its employees' names and contact details. It creates a table as presented in Table 8.8.

Table 8.8: *An example of a table not conforming to 1NF*

emp_id	emp_name	emp_address	emp_mobile
101	Hamis Waya	Iringa	06580001770
102	Erick Misitu	Mbeya	0713876776; 0767000771
103	Mwantumu Chake	Coastal	0688000775
104	Namdori Mchicha	Arusha	028675432; 0736000671

As you can see from Table 8.9, employees (Erick and Namdori) have two mobile numbers, so the company stored them in the same field as you can see in emp-mobile. Therefore, this table is not in 1NF as a rule says, "each attribute of a table must have atomic (single) values"; the emp_mobile values for employees Erick and Namdori violate this rule. To make the table comply with 1NF, we should have those anomalies fixed in the data, as indicated in Table 8.9.

Table 8.9: *An example of a table conforming to the First Norm Form (1NF).*

emp_id	emp_name	emp_address	emp_mobile
101	Hamis Waya	Iringa	0658000770
102	Erick Misitu	Mbeya	0713876776
102	Erick Misitu	Mbeya	0767000771
103	Mwantumu Chake	Coastal	0688000775
104	Namdori Mchicha	Arusha	028675432
104	Namdori Mchicha	Arusha	0736000671

For example, if we want to put Table 8.9 in the 1N, it will look as in Table 8.10

Table 8.10: *The first normalized form*

Full name	Address	Movie_Industry	Sports
Ismail Makame	Kijichi	Hollywood	Football
Ismail Makame	Kijichi	Bollywood	Basketball
Ismail Makame	Kijichi	Hollywood	Basketball
Ismail Makame	Kijichi	Bollywood	Football
Rehema Kiongwe	Mnazi Moja	Swahiliwood	Netball

FOR ONLINE READING ONLY

Full name	Address	Movie_industry	Sports
Rehema Kiongwe	Muazi Moja	Nollywood	Netball
Mohamed Muki	Mbagala	Riverwood	Cricket
Mohamed Muki	Mbagala	Nollywood	Cricket

Second Normal Form (2NF)

A table is in 2NF, if the relation or table is in 1NF (First Normal Form) and all non-key attributes are fully functional dependent on the primary key (non-prime attribute). For example, suppose a school wants to store the data of teachers and the subjects they teach and, in a school, a teacher can teach more than one subject. Since a teacher can teach more than one subject, the table can have multiple rows for the same teacher, as indicated in Table 8.11.

Table 8.11: *An example of a table not conforming to 2NF*

Teacher_id	Subject	Teacher_age
25	Computer	40
25	Biology	40
47	English	35
83	Math	36
83	Chemistry	36

Based on Table 8.11, we have; candidate keys which are {Teacher_id, Subject} and the Non-prime attribute is: teacher_ age. Therefore, the non-prime attribute Teacher_age is dependent on Teacher_id, which is a proper subset of a candidate key; thus, violating the rule for 2NF. To convert the given table into 2NF, we decompose it into two tables: Teacher details in Table 8.12 and Teacher- subject in Table 8.13, which now comply with Second Normal Form (2NF). Resulting tables conforming to Second Norm Form (2NF).

Table 8.12: *Teacher details*

Teacher_id	Teacher_age
25	40
47	35
83	36

Table 8.13: *Teacher_subject table*

Teacher_id	Subject
25	Computer
25	Biology
47	English
83	Math
83	Chemistry

The characteristics of 2NF

(a) In the 2NF, relational must be in INF.

(b) In the second normal form, all non-key attributes are fully functional and dependent on the primary key.

(c) A single column Primary Key that is not functionally dependent on any subset of candidate key relation.

Third Normal Form (3NF)

A table design is said to be in 3NF if it is in 2NF and transitive functional dependency of the non-prime attribute

FOR ONLINE READING ONLY

(an attribute that is not part of any candidate key) on any super key (single key or a group of multiple keys that can uniquely identify tuples in a table) should be removed.

In other words, a table can be explained as in 3NF if it is in 2NF, and for each functional dependency X-> Y, at least X is a super key of the table or Y is a prime attribute of the table. For example, suppose a company wants to store the complete address of each employee; they create a table named employee_ details, as indicated in Table 8.14.

Table 8.14: *An example of a table not conforming to 3NF.*

emp_id	emp_name	emp_pc	emp_reg	emp_city	emp_district
1001	John Miti	282005	AR	Arusha	Arumeru
1002	Muhidini Kata	222008	DS	DSM	Kinondoni
1006	Lora Misitu	282007	DS	DSM	Temeke
1101	Lilly Nyavu	292008	TG	Tanga	Muheza
1201	Steve Milango	222999	MZ	Mwanza	Ilemela

In Table 8.14 you can see that Super keys are: {emp_id}, {emp_id, emp_name}, {emp_id, emp_name, emp_pc}and so on; Candidate Keys: {emp_id}, Non-prime attributes: all attributes except emp_id are non-prime as they are not part of any candidate keys.

Here, emp_reg, emp_city and emp_ district depend on emp_pc, and, emp_pc is dependent on emp_id that makes non-prime attributes (emp_reg, emp_city & emp_ district) transitively dependent on super key (emp_id). This violates the rule of 3NF. To make this table comply with 3NF, we have to break the table into two tables to remove the transitive dependency, as shown in Table 8.15 and Table 8.16, which conform to 3NF.

Table 8.15: *Employees' details*

emp_id	emp_name	emp_pc
1001	John Miti	282005
1002	Muhidini Kata	222008
1006	Lora Misitu	282007
1101	Lilly Nyavu	292008
1201	Steve Milango	222999

FOR ONLINE READING ONLY

Table 8.16: *Employee address*

emp_pc	emp_reg	emp_city	emp_district
282005	AR	Arusha	Arumeru
222008	DS	DSM	Kinondoni
282007	DS	DSM	Temeke
292008	TG	Tanga	Muheza
222999	MZ	Mwanza	Ilemela

Activity 8.7:

Normalising databases up to the 3rd Normal form

Aim:
The aim of this activity is to understand and apply the principles of database normalization up to the Third Normal Form (3NF).

Materials:
Computer with database management system (DBMS) software (such as, MySQL, PostgreSQL);
Access to Employee details.

Instructions:
Study Table 8.17 and then answer the questions that follow:

Table 8.17: *Employee details before normalization*

Staff No	Branch No	Branch address	Name	Position	Hr/week
S3244	B0021	Temeke, Pl 12G DSM	Asha Haji	Assistant	16
S3244	B0045	Mikocheni, Pl 32A, DSM	Asha Haji	Assistant	9
4545	B0021	Temeke, Pl 12G DSM	Mayalla Macho	Assistant	14
S4545	B0045	Mikocheni, Pl 32A, DSM	Mayalla Macho	Assistant	10

Questions:

1. Why is this table not in 3NF? Give two reasons.
2. Describe and illustrate the process of normalizing the data shown in Table 8.17 to the Third Normal form (3NF).
3. Identify the primary, alternate, and foreign keys in your 3NF relations.

FOR ONLINE READING ONLY

Activity 8.8:

Designing a database model for a school library.

Use the following steps to perform the activity:

Aim:
To enhance understanding of database design, schema implementation, and SQL querying skills among students

Materials:
Textbooks, online tutorials, Computer with database design software (Such as MySQL, Microsoft Access or any relational Database Management System-RDBMS); Sample data sets

Instructions:

Step 1: Choose a database model
Select an appropriate database model (relational or object-oriented).

Step 2: Understand Requirements
List the requirements for a school library system, including Book inventory, Student records and Borrowing transactions

Step 3: Design the Schema
Using the chosen database model, design a schema that includes:
(a) Entities: such as Books, Students, and Transactions
(b) Attributes: such as Books (BookID, Title, Author);
(c) Students (StudentID, Name);
(d) Transactions (TransactionID, BookID, StudentID, BorrowDate); and
(e) Relationships (such as students borrowing books)

Step 4: Create Sample Data
Create sample data for: Books (such as BookID: 1, Title: "Database Concepts", Author: "Kihambule Mtemi"), Students (such as, StudentID: 101, Name: "Juakali Matonge"), Transactions (such as, TransactionID: 201, BookID: 1, StudentID: 101, BorrowDate: "2024-06-01")

Step 5: Implement the Database
Use database design software to Implement the schema and enter the sample data

Step 6: Query the Database
Use basic SQL queries or relevant commands to Retrieve data (such as list all books, find books borrowed by a specific student) and Generate simple reports.

FOR ONLINE READING ONLY

Deliverables:
A report summarising your work and reflections including screenshots for detailed schema design, Sample data entries, Implemented database, and SQL queries or commands used. In a report explain the rationale behind your design choices and benefits and limitations of your chosen model. Document your work in a portifolio.

Exercise 8.4

1. Describe the steps you would take to convert unnormalised tables to the First Normal Form, Second Normal Form, and Third Normal Form.
2. Provide at least one example relevant to a school or college environment that illustrates reasons for converting database tables to the First, Second and Third Normal Forms.
3. Contrast relational and object-oriented models and give two examples of their uses in real life for each of them.
4. Why is the relational model the most used model today despite the continued inventions of even newer models?
5. Identify a real-world application for each of the following database models: hierarchical, network, and relational. Justify your choice based on the characteristics of each model.
6. What features would you find in an object-oriented database model that are not typically present in a relational database model?
7. How do data structures like trees relate to database models? Provide an example of a database model that uses this structure.

Database Management Systems (DBMS)

Database management is the process of organizing, storing, and retrieving data from a computer. It involves designing, implementing, and sustaining stored data to optimise its value, which database management represents. In designing a system, a programmer must have a database to store data. For example, if you want to store student results in an academic records system, you need to have a database so that all the data can be stored in it. A database enables you to perform many operations, such as inserting, updating, and deleting data. Some common database management systems include MySQL, PostgreSQL, SQLite, Microsoft Access, SQL Server, FileMaker, Oracle, RDBMS, dBASE, Clipper, and FoxPro.

Structured Query Language

Structured Query Language (SQL) is a database programming language designed to retrieve and manage data in a relational database.

FOR ONLINE READING ONLY

Importance of Structured Query Language (SQL)

(a) Allows users to access data in the relational database management systems.
(b) Enables users to describe the data.
(c) Allows users to define the data in a database and manipulate that data.
(d) Allows embedding within other languages using SQL modules, libraries, and pre-compilers.
(e) Enables users to create and drop databases and tables.
(f) Enables users to create, view, and store procedures and functions in a database.
(g) Allows users to set permissions on tables, procedures, and views.

Since there are so many database management systems, in this chapter you will learn about Structured Query Language using MySQL, a popular open-source relational database management system (RDBMS) that uses Structured Query Language (SQL) for managing and manipulating data. To use this DBMS, you need to install the XAMPP application, an open-source software stack that provides an easy-to-install Apache distribution for developers to create a local web server environment as a popular choice for developers who need a local development environment for PHP and MySQL-based applications.

To bring this environment in place download and install XAMPP, then start MySQL and Apache on the XAMPP control panel to access the database; open your browser and type localhost/phpMyAdmin, and the browser will open the dashboard as shown in Figure 8.18.

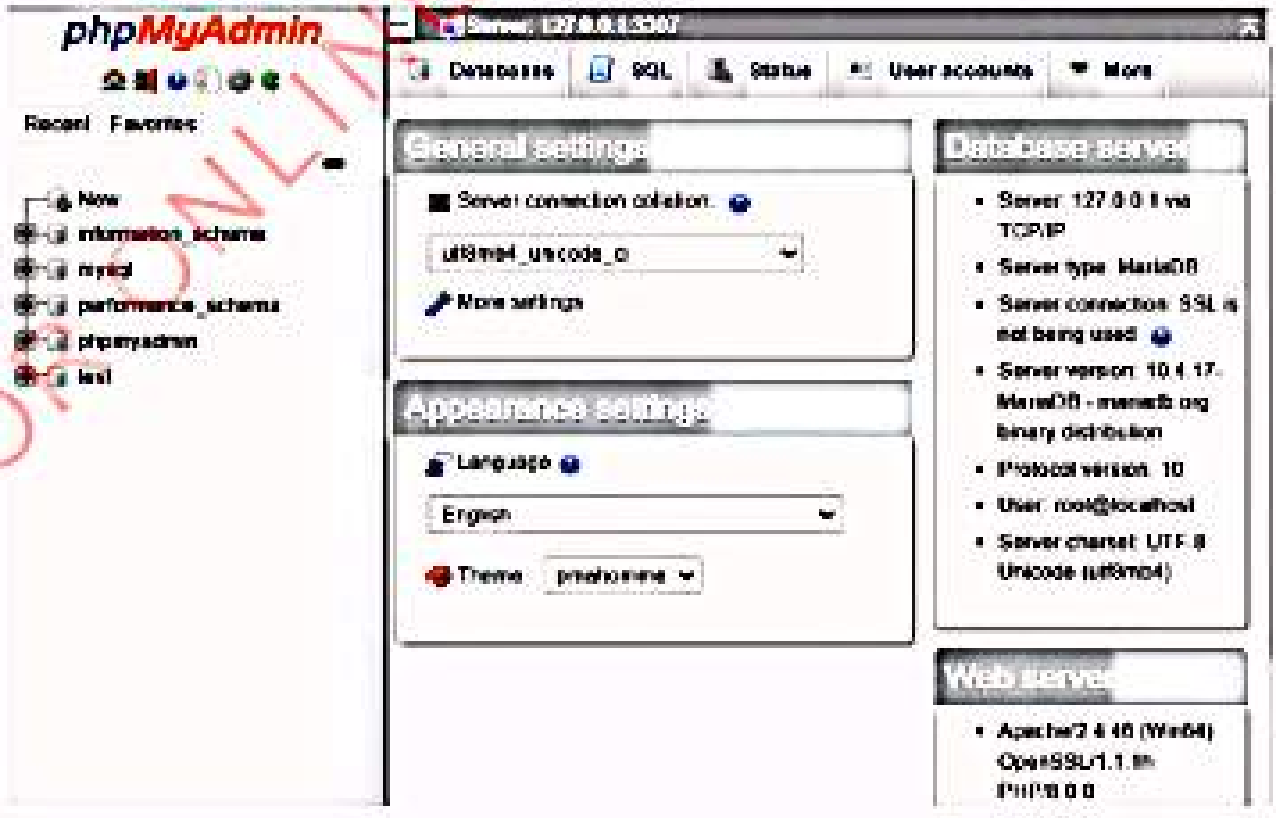


Figure 8.18: *PhpMyAdmin dashboard*

FOR ONLINE READING ONLY

SQL commands

There are five fundamental command categories, in SQL language, a standard language for managing and manipulating databases. These command categories are Data Definition Language (DDL) commands, Data Manipulation Language (DML) commands, Data Query Language (DQL), Data Control Language (DCL) and Transaction Control Language (TCL). It is crucial to keep the following considerations in mind before diving into SQL:

(a) SQL statement starts with SQL command and ends with a semicolon (;);

(b) SQL statements are case insensitive, which means they can be written using upper, lower, or mixed cases. But it is a common practice to write out SQL keyword commands in all upper cases;

(c) Using SQL, you can perform most of the actions in a database; and

(d) SQL data types are critical because they tell the database what type of data to expect. Common data types include:

(i) **INT**: An integer number.

(ii) **VARCHAR**: Variable-length strings. For example, VARCHAR(255) can store strings up to 255 characters.

(iii) TEXT: For long texts.

(iv) **DATE/TIME**: For dates and times.

(v) FLOAT/DOUBLE: For floating-point numbers.

Example context: "If a table needs to store people's names and their ages, you would use VARCHAR for names and INT for ages."

Data Definition Language (DDL) commands

DDL commands change the database's structure, for example, by creating a table, deleting a table, and altering a table. DDL has the following commands: CREATE, ALTER, DROP, and TRUNCATE.

(a) CREATE

The CREATE command is used to create a new database or table in the database. It has the following syntax: CREATE DATABASE database_name; for the database and CREATE TABLE table_name (column_name datatypes [,]); for the table. The database must be created first then, followed by the tables;

Example:

```
CREATE DATABASE student;
CREATE TABLE student_registration (
StudentID int,
FirstName varchar (255),
MiddleName varchar (255),
LastName varchar (255),
Address varchar (255)
);
```

To create a database and table resulting from this example, select SQL to continue with the code and type the code; first, you need to create a database and then the table as indicated in Figure 8.19 this is because the table shall belong to the database.

FOR ONLINE REFERENCE ONLY

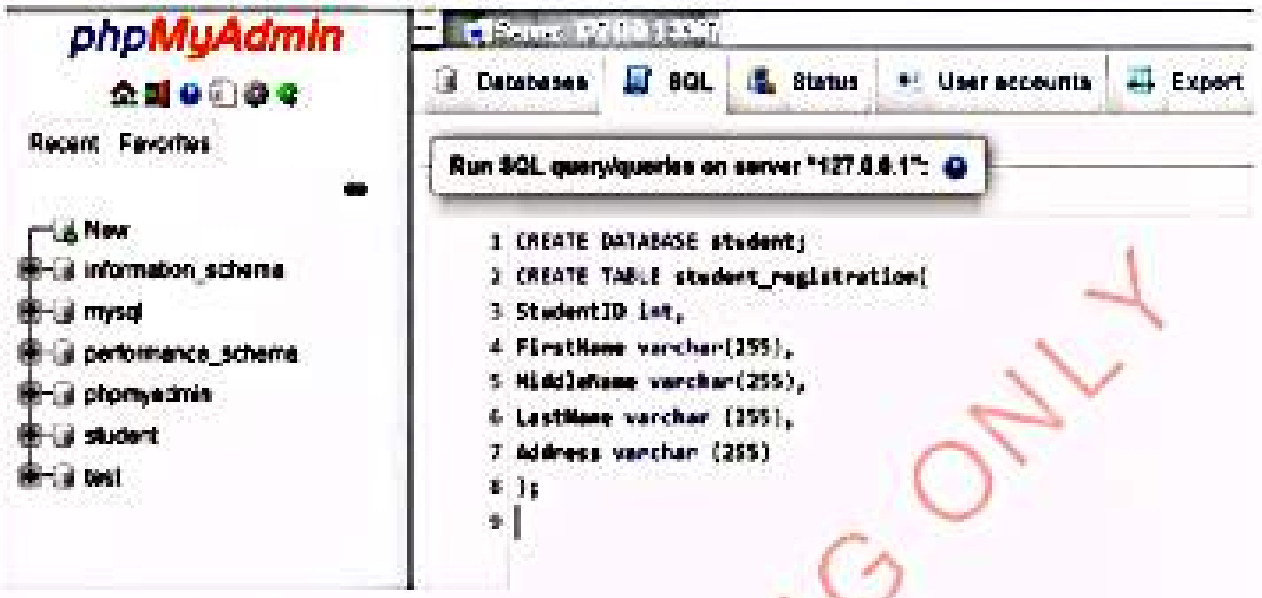


Figure 8.19: *Creating database and table*

When you run the code, it will create the database, and inside the database, the table will also be created, as shown in Figure 8.20.

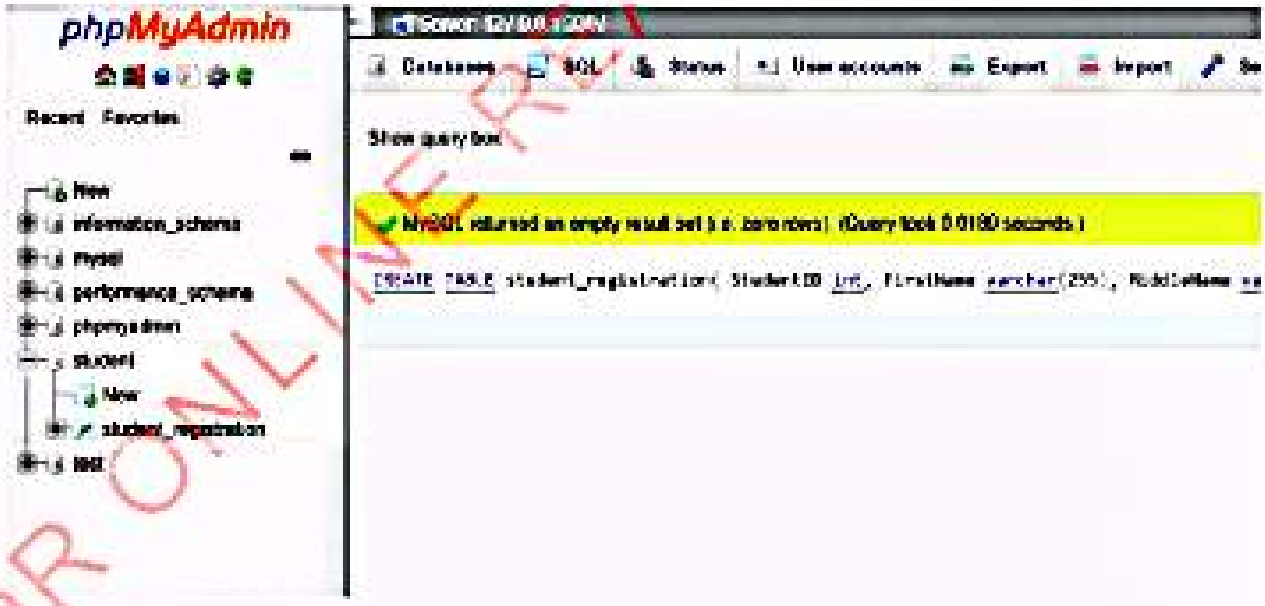


Figure 8.20: *SQL query for creating a table*

(b) ALTER

Alter command is a statement used to add, delete, or modify columns in an existing table. The syntax to alter a table by adding a column is ALTER TABLE table_name ADD column_name datatype; For example, if you want to add a parent name in our database named "student" in the table student_ registration, the SQL command will be: *ALTER TABLE student_registration ADD ParentName varchar(255);* This will add the ParentName column to the student_registration table. To delete a column

FOR ONLINE READING ONLY

in a table, use the following syntax *ALTER TABLE table_name DROP COLUMN column_name*; For example - We need to delete ParentName from the student_registration table; the command will be: *ALTER TABLE student_registration DROP COLUMN ParentName*; To modify a column data type in a table, use the following syntax ALTER TABLE table_name MODIFY COLUMN column_name datatype;

(c) DROP

The drop command is used to delete both the table structure and the record stored in the table. The syntax is DROP TABLE table_name; For example: If you want to delete table student_registration, the command will be DROP TABLE student_registration. This will delete the table and its contents.

(d) TRUNCATE

It is used to delete all rows in the table. The syntax is TRUNCATE TABLE table_name; For example: If you have to delete the rows in the student_registration table, you need to write TRUNCATE TABLE student_registration;

Data Manipulation Language (DML) commands

DML commands are used to modify the database. Those commands are responsible for all forms of CHANGES in the database. The command of DML is not auto-committed, which means it cannot permanently save all the changes in the database. They can be rolled back. DML involves INSERT, UPDATE, and DELETE commands.

(a) INSERT

The INSERT command is used to add new records to the table. The syntax is *INSERT INTO table_name (column1, column2, column3, ...) VALUES ('value1', 'value2', 'value3', ...);* Example: Insert students' data in the *student_registration* table with column (studentID, FirstName, MiddleName, LastName, Address).

An example of an INSERT query

INSERT INTO student_registration (StudentID, FirstName, MiddleName, LastName, Address) VALUES ('1234', 'Mwakalinga', 'Mwakatope', 'Mkinga', '972 Mbeya'); Also, see the query in Figure 8.21.

```
1 INSERT INTO student_registration
2 (studentID, FirstName, MiddleName, LastName, Address)
3 VALUES ('1234', 'Mwakalinga', 'Mwakatope', 'Mkinga', '972 Mbeya');
```

Figure 8.21: *The INSERT query*

FOR ONLINE READING ONLY

If you run this query, it will insert data in the table, as shown in Figure 8.22.

StudentID	FirstName	MiddleName	LastName	Address
1234	Mwakalinga	Mwakatope	Mkinga	972 Mbeya

Show all | Number of rows: 25 Filter rows: Search this table

Figure 8.22: *INSERT data in the table*

(b) UPDATE

The update command is used to update the data value in a table column. After inserting data, you can modify it by using the update command. The syntax is: UPDATE table_name SET column1 = 'value1', column2 = 'value2', WHERE condition;
WHERE is used to specify which data should be modified. If the condition is not specified, ALL the rows in a table will be modified. For example, suppose you want to update the student's first name with studentID number 1234 from Mwakalinga to Musawomda. In that case, you will apply the following query: UPDATE student_registration SET FirstName ='Musawomda 'WHERE studentId = 1234; Output is as shown in Figure 8.18 (a) and (b).

```
UPDATE student_registration SET FirstName= 'Musawomda' WHERE StudentID = 1234
```

Figure 8.23 (a): *Update query*

StudentID	FirstName	MiddleName	LastName	Address
1234	Musawomda	Mwakatope	Mkinga	972 Mbeya

Figure 8.23 (b): *Select query of the updated record*

(c) DELETE

The DELETE command is used to delete existing records in a table. After inserting data, you can delete the data by using the delete command. The syntax is *DELETE FROM table_name WHERE condition;*

For example: if you want to delete all the information about Musawomda from our previous example, the query will be: *DELETE FROM student_registration WHERE FirstName='Musawomda '*; this query will delete all records with the first name Musawomda. If you want to delete only the record for Musawomda with studentId 1234, you should write the following query:

DELETE FROM student_registration WHERE studentId=1234;

FOR ONLINE READING ONLY

Data Query Language (DQL) command DQL is used to fetch the data from the database. It uses only one command, which is SELECT. Its syntax is SELECT column1, column2, ...FROM table_name; Example: If you want to retrieve the first name and middle name only from the student-registration table, the query is such that: *SELECT FirstName, MiddleName FROM student_registration OUTPUT* (see Figure 8.24):

FirstName	MiddleName
Mwakatope	Mkinga
Mkinga	Mwakalinga
Mwakalinga	Mwakatope

Figure 8.24: *The output of the SELECT query*

Data Query Language (DQL) commands

Data Query Language (DQL) commands are a type of SQL commands primarily used for querying and retrieving data from databases. The main DQL command is 'SELECT', which fetches data from one or more tables. This command allows users to specify which columns to *retrieve, filter results using conditions, join tables, and sort the output*. DQL is fundamental for accessing and analyzing data stored in relational databases. Example of a Data Query Language (DQL) command in SQL:

```
SELECT name, age, email
FROM users
WHERE age > 45
ORDER BY name ASC;
```

This query retrieves the name, age, and email columns from the users table where the age is greater than 45, and it orders the results by name in ascending (ASC) order. The primary DQL command in SQL is the SELECT statement. However, SQL uses other statements for querying data indirectly through the SELECT statement, such as using operators and clauses.

Common operators and clauses used with SELECT command

These functions are such as 'JOIN', 'WHERE', 'ORDER BY', 'GROUP BY', 'HAVING', and 'UNION', 'EXISTS', 'IN', 'BETWEEN', and 'LIKE'

(a) SELECT

This 'SELECT' command retrieves data from a database. For example,

```
SELECT * FROM employees;
```

FOR ONLINE READING ONLY

(b) JOIN

The 'JOIN' clause merges rows from multiple tables by using a common column that links them. For example,

```
SELECT employees.name, departments.department_name
FROM employees
JOIN departments ON employees.department_id = departments.department_id;
```

(c) WHERE

The 'WHERE' clause is used to filter records. For example,

```
SELECT name, age
FROM employees
WHERE age > 50;
```

(d) ORDER BY

The 'ORDER BY' clause sorts the result set in either ascending or descending order. For example,

```
SELECT name, salary
FROM employees
ORDER BY salary ASC;
```

(e) GROUP BY

The 'GROUP BY' clause organises rows with identical values in specified columns into summary rows. For example,

```
SELECT department_id, COUNT(*)
FROM employees
GROUP BY department_id;
```

COUNT(*) in this statement, tallies the number of records associated with each unique department_id in the employees table, summarizing the total rows per department_id group. This query outputs a list of department_id values alongside the corresponding count of employees within each department.

(f) HAVING

The 'HAVING' clause was introduced in SQL to enable filtering based on aggregate functions, which the 'WHERE' keyword cannot do. For example,

```
SELECT department_id, COUNT(*)
FROM employees
GROUP BY department_id
HAVING COUNT(*) > 10;
```

FOR ONLINE READING ONLY

(g) UNION

The 'UNION' operator is used to combine the result sets of two or more SELECT statements. For example,

```
SELECT name FROM employees
UNION
SELECT name FROM contractors;
```

(h) EXISTS

The 'EXISTS' operator checks whether any records exist in a subquery. For example,

```
SELECT name
FROM employees
WHERE EXISTS (SELECT 1 FROM departments WHERE departments.
department_id = employees.department_id);
```

(i) IN

The 'IN' operator allows you to specify multiple values in a WHERE clause. For example,

```
SELECT name
FROM employees
WHERE department_id IN (1, 2, 3);
```

(j) BETWEEN

The 'BETWEEN' operator selects values within a given range. For example,

```
SELECT name, salary
FROM employees
WHERE salary BETWEEN 650000 AND 1250000;
```

(k) LIKE

The 'LIKE' operator is employed to find a particular pattern within a column. For example,

```
SELECT name
FROM employees
WHERE name LIKE 'J%';
```

The pattern 'J%' in the WHERE clause selects names from the employees table where the name column starts with 'J'. Using '%', a wildcard, matches any characters in SQL queries. Thus, this query retrieves all names starting with 'J'.

FOR ONLINE READING ONLY

Data Control Language (DCL) command

It is used to control users' rights in accessing data in the database. It uses two controls, GRANT and REVOKE.

(a) GRANT

It is used to give users access privileges to a database. The syntax is *GRANT privilege [, privilege], ON privilege_ level* TO account_name: For example, If you need to give user_M access to the database student, the query will be:

GRANT ALL ON
student
TO user_M;

(b) REVOKE

It is used to revoke access permissions that were previously granted to prevent a user from accessing the database objects. The syntax is *REVOKE privilege_name ON object_name FROM {user_name |PUBLIC |role_name}*

For example: If you need to remove user_M from accessing the database student, the query will be:

REVOKE ALL
ON student
FROM user_M;

Transaction Control Language (TCL) command

Transaction Control Language command is used to handle database transactions. It keeps track of the modifications made by the data manipulation language. It also enables the grouping of statements into logical transactions. The transaction control language consists of COMMIT, ROLLBACK, and SAVEPOINT.

(a) COMMIT

COMMIT command is used to save any transaction into the database permanently. When we use any DML command like INSERT, UPDATE or DELETE, the changes made by these commands are not permanent; until the current session is closed, the changes made by these commands can be rolled back. To avoid that, we use the COMMIT command to mark the changes as permanent. The syntax is COMMIT;

Example:
Delete from student_registration
Where address = 250 Dodoma;
Commit;

FOR ONLINE READING ONLY

(b) ROLLBACK

If you have used the UPDATE command to make some changes to the database and realise that those changes were not required, you can use the ROLLBACK command to roll back those changes if they were not committed using the COMMIT command.

Example;

Delete from student_registration

Where address = 250 Dodoma;

Rollback;

(c) SAVEPOINT

Is also a command used within transactions to establish a point to which you can later roll back without affecting the entire transaction. It is used to roll back the transaction to a certain point without rolling back the entire transaction. The syntax for the savepoint commands is *SAVEPOINT savepoint_name;*

Example:

UPDATE class SET name = 'Mwanamuu' WHERE id = '5';

SAVEPOINT A;

Activity 8.9:

Designing a database table using SQL

Aim:

To practice designing and implementing a database table using SQL, focusing on schema creation and data population with 5 to 10 columns.

Materials:

1. Computer with database management system (DBMS) software (such as MySQL, PostgreSQL)
2. SQL database environment set up on your computer or a remote server

Instructions:

Step 1: Identify a problem

Think of a practical scenario or problem that requires storing information in a database table with 5 to 10 columns. For example, it could be a table to store customer information, product details, student records, and so on.

Step 2: Design a database

Design the schema of your database table. Define 5 to 10 columns and their respective data types based on the information you need to store. Consider primary keys, foreign keys, and any necessary constraints.

Step 3: Create SQL table

Using your preferred DBMS software, write SQL commands to create the

FOR ONLINE READING ONLY

database table based on your design. Ensure the table structure aligns with your intended data storage requirements.

Step 4: Populate data

Insert sample data into your newly created table using SQL INSERT statements. Populate the table with meaningful data that reflects the scenario or problem you identified in step 1.

Step 5: Verify data

Execute SQL SELECT queries to verify that data has been successfully inserted into the table. Check for any errors or inconsistencies in the data entry process.

Deliverable:

Submit a report summarizing your database design, implementation steps, and screenshots of the populated table. Reflect on the challenges faced and lessons learned from designing and populating the database table using SQL. Document your work in a portifolio.

Data storage, indexing, and query optimization in managing databases

In Managing Database Management Systems (DBMS) the foundational concepts of data storage, indexing, and query optimization are critical for understanding how databases efficiently store, retrieve, and manage data. Here we briefly cover these essential aspects:

Data storage in DBMS

Data storage in a DBMS refers to how information is systematically stored in databases for efficient retrieval and management. This involves organizing data in various structures seen earlier such as tables (in relational databases), documents (in NoSQL databases), or objects (in object-oriented databases).

Activity 8.10:

Data storage and organisation in a normal school environment

Aim:

To simulate database organisation principles through physical card sorting, exploring concepts like categorization and searchability, and drawing parallels to digital data management.

Materials Needed:

Set of physical cards (such as index cards, sticky notes) and marker pens.

Instructions

Step 1: Prepare the Cards
Take a set of physical cards, each representing a piece of data (such as customer names, orders, product details).

Step 2: Organise the Cards
Organise these cards in a manner that allows for quick retrieval of specific information.

Step 3: Plan Your Organisation Strategy
Before organizing, strategize

FOR ONLINE READING ONLY

how you will categorize or sort the cards based on common attributes.

Step 4: Organise the Cards Use marker pens to label categories or sort the data based on identified attributes or criteria.

Questions

1. Describe how you organised the cards, including the categories or sorting criteria used. Discuss the advantages (ease of retrieval, visual clarity) and challenges (ambiguous categorization, limited space) encountered during the process.
2. Discuss similarities between organizing physical cards and organizing data in a database. Highlight concepts such as categorization (grouping data based on common attributes), searchability (ease of finding specific information), and scalability (handling larger datasets).

Indexing

Indexing in a DBMS is similar to the index found at the back of a textbook. It allows quicker retrieval of data from the database by creating a secondary data structure (the index) that points to the location of data in the primary storage. This means queries can find data faster because the DBMS knows exactly where to look, rather than scanning the entire database.

Activity 8.11:

Indexing in a school environment

Aim:

To demonstrate the importance of indexing in facilitating efficient data retrieval using books as examples.

Materials:

Two identical copies of the same book (one with an index, one without).

Instructions

Follow these steps to find information in two different books and then answer the questions at the end of the activity.

Step 1: Open the book that lacks an index.

Step 2: Select and find specific information of your choice (such as a particular chapter or topic).

Step 3: Open the book that has an index.

Step 4: Again, select and find specific information of your choice using the index.

Questions

1. Explain the process and challenges faced when searching for information in the first book without an index.
2. Describe how the presence of an index in the second book facilitated faster information retrieval.
3. Discuss the impact of having an index on the speed and ease of

FOR ONLINE READING ONLY

searching for information in the second book.

4. Compare the functionality of a book index with indexes in databases, highlighting similarities and differences.

Deliverables:

A comparison report detailing the experience of searching for information in both books, such that: Explanation of the search process in the book without an index; Description of the search process in the book with an index; and Discussion on how the presence of an index affected the speed and ease of searching information. Document all your work in a portifolio.

Another example of index in the context of a school environment in Library. You can imagine an index in your school library that lets you quickly find any book, that will be similar to index in database. Likewise, databases use indexes to speed up the data search.

Query Optimization

Query optimization is the process used by DBMS to find the most efficient way to execute a query. Given a query, there might be multiple ways to retrieve the required data. The DBMS evaluates these possibilities and chooses the path that uses the least resources or the shortest time. It involves decisions like which index to use or in what order to join tables.

Activity 8.12:

Exploring query optimization in database management

Aim:

To develop efficient strategies for optimizing database queries to retrieve specific information effectively.

Materials:

Set of physical data cards (used in previous activities), complex questions or queries related to the dataset.

Instructions

Use the following steps to optimise the dataset and answer questions that follows at the end of activity.

Step 1: Prepare a set of complex questions or queries using the data cards from the previous activity.

Step 2: Plan different strategies, including approaches like filtering, sorting, or utilizing indexes, to accelerate finding answers.

Step 3: Choose and implement your preferred approach or combination of approaches to answer the questions effectively.

Step 4: Execute and refine your queries based on the strategies chosen.

Questions

1. Describe the strategy(s) or approach you selected and discuss any challenges encountered.

FOR ONLINE READING ONLY

2. Explain your process for finding answers to the queries using your chosen approach(es).
3. Discuss how database management system (DBMS) software optimises queries through analyzing query execution plans, utilising indexes, and employing optimization techniques.
4. Analyse how the strategies you employed align with the principles of query optimization in DBMS.
5. Discuss the importance of selecting the most efficient approach based on factors such as data volume, complexity, and performance requirements.

Optimization in context of school environment is similar to organizing your study notes, in such a way that it will make you review them easily before an examination. In this case, optimizing a database makes data retrieval faster and more efficient.

Activity 8.13:

Understanding database functionality and sql operations

Aim:

To understand database functionality, efficient data storage and access, and the role of SQL in managing database operations.

Materials:

Computer with a database management system (DBMS) software (such as MySQL, PostgreSQL); and SQL tutorials or reference guide.

Instructions:

Step 1: Create a database for a topic of your interest, such as your favorite sports team.

Step 2: Manipulate data-insert, update, and delete records in your database. For example, you can add new players, update player statistics, or remove retired players.

Step 3: Query Challenge- create challenges where you must retrieve specific information using SELECT statements with various conditions. For example, finding the top scorer or listing players from a particular position.

Step 4: Optimise querry by creating indexes on your tables and observe the performance difference in query execution. Compare the speed of queries before and after indexing.

Deliverables:

Write a summarized report of your database design, including table structures and relationships. In a report, include examples of SQL statements used for data manipulation and queries, screenshots or descriptions of the performance differences observed during the optimization experiment. Also, write the answers to the questions provided above.

FOR ONLINE READING ONLY

Questions

1. Describe the steps you took to create your database and the tables within it.
2. How did you handle data manipulation tasks? Provide examples of insert, update, and delete operations you performed.
3. Explain the queries you used to retrieve specific information. What challenges did you face, and how did you overcome them?
4. Discuss the performance differences you observed before and after creating indexes on your tables. Why are indexes important for query optimization?
5. Reflect on how understanding database operations and SQL can be beneficial in real-world applications.

Exercise 8.5

1. Create an SQL statement to create a table named "Books". The table should have four columns:
   (a) BookID - as an integer and the primary key,
   (b) Title - as a variable character with a maximum length of 250 characters,
   (c) AuthorName - as a variable character with a maximum length of 100 characters, and
   (d) PublishedYear - as an integer.
2. Write an SQL query to insert the following data into the Books table:
   (a) BookID : 1, Title: "Database System Concepts", Authors: " Abraham Silberschatz, Henry F. Korth, and S. Sudarshan", PublishedYear: 2020
   (b) BookID : 2, Title : "Artificial Intelligence in Education: Promise and Implications for Teaching and Learning ", Authors: "Charles Fadel, Wayne Holmes and Maya Bialik", PublishedYear : 2017
3. Write a SQL query to retrieve all columns for all books in the Books table that were published after the year 2018.
4. The publication year of "2017" by Charles Fadel, Wayne Holmes, and Maya Bialik was mistakenly entered as 2017. It was published in 2019. Write an SQL query to correct this mistake in the Books table.
5. Assuming there was a mistake, and the book with BookID 2 is not supposed to be in the Books table. Write an SQL query to delete this book from the table.

Parallel and distributed Database management systems

The concepts of parallel and distributed database management systems (DBMS) lie in how databases can be optimised for performance, reliability, and scalability. Given the complexity of these topics here we are simplifying them including providing relatable examples when possible.

FOR ONLINE READING ONLY

Parallel database systems

Parallel database systems enhance performance by parallelizing operations like data retrieval, updates, and indexing. Tasks are divided into smaller parts and executed simultaneously across multiple processors in the same computer, allowing multiple queries or parts of a single query to be processed at once. This results in faster data processing and better resource utilization, making parallel DBMS crucial for high-volume and complex queries in big data analytics, Online Transaction Processing Systems (OLTP) systems, and real-time applications. A good example on parallel database system is like a group project where each member works on different sections simultaneously, completing the project faster than if done sequentially.

Case in Tanzania: The E-Government Agency (eGA) uses parallel database systems to handle large volumes of data for public services. When citizens access services like paying utilities or taxes, parallel processing manages multiple requests efficiently, improving service delivery and user experience.

Distributed Database Systems

Distributed databases spread data across multiple locations connected via a network, ensuring data accessibility and consistency across all sites. Systems like MongoDB are used for high availability, scalability, and performance in applications such as e-commerce, social media, and real-time analytics. A distributed DBMS manages data across a dispersed network, providing high performance, reliability, and consistency despite geographical distribution.

This is like a library system with branches across a city, each operating independently but sharing information to ensure data consistency across all branches.

In Tanzania, the National Identification Authority (NIDA) uses a distributed database to manage National ID data from different locations. This ensures data is accurate and available in real-time across the country. For example, if a citizen updates their information at a local NIDA office, the changes are immediately updated in the system, making the latest information available for ID verification anywhere in Tanzania.

Activity 8.14:

The village water systems – a parallel and distributed systems

Aim:

To understand the concepts and benefits of parallel and distributed database systems through a real-world simulation.

Materials:

Cards (or pieces of paper) with tasks and questions; timers or stopwatches; a map of Tanzania (or a fictional region) divided into several areas; Markers or tokens; and A simple "database schema" related to water usage tracking (such as reading meters, tracking water quality, scheduling maintenance).

FOR ONLINE READING ONLY

Instructions

Step 1: Parallel Database Systems

Setup: Imagine you are managing water usage and maintenance tasks for the entire country. Divide the tasks into smaller sub-tasks (such as, calculating water usage, identifying needed maintenance) and write them on cards.

Task: Complete all sub-tasks as quickly as possible, working on them simultaneously (parallel processing).

Outcome: Reflect on how tasks were completed faster through parallel processing and relate this to how parallel DBMS improve performance and efficiency.

Step 2: Distributed Database Systems

Setup: The map represents different areas of Tanzania, each with specific water system-related tasks. Assign tasks to different areas but ensure coordination between them.

Task: Work on area-specific tasks, occasionally collaborating with other areas to solve problems, simulating data sharing and communication in a distributed system.

Outcome: Reflect on the importance of coordination and data consistency in distributed systems. Discuss the challenges and advantages, such as improved data access speed and increased fault tolerance.

Deliverable

A brief report explaining approach to completing the tasks and challenges faced. Including a simple schema related to water usage tracking, explained with descriptions of how the tasks were distributed and processed; A visual representation of the areas with marked tasks and notes on coordination between different areas must be shown. Document your work in a portifolio.

Questions

1. Explain your strategy for completing tasks in parallel.
2. Describe how you coordinated tasks between different areas in the distributed system.
3. Discuss the benefits and challenges of parallel and distributed systems in this analogy.
4. Provide real-world examples in Tanzania where these systems can be applied.

Exercise 8.6

1. Write an SQL query to select all records from a table named "Students" where the "Grade" is 'A'.
2. Construct an SQL query to update the salary of all employees in the "Sales" department by 10%.
3. What attributes in database identify if it that it is either parallel or distributed database systems?
4. Why developers considered data partitioning in parallel database systems an important technique?

FOR ONLINE READING ONLY

Emerging Database Models, Technologies, and Applications

The database landscape is continuously evolving with new models, technologies, and applications. Innovations like Big Data, Cloud-based databases, NoSQL, NewSQL, and Serverless databases offer unique advantages for various use cases such as big data analytics, real-time processing, and IoT applications. These advancements are crucial for handling complex data types, improving operational efficiency, and driving innovation across industries. Below, we explore some of the most significant emerging database models, technologies, and their potential applications.

Emerging database models

(a) **NoSQL databases:** These databases handle various data structures, unlike traditional SQL databases that use a table-based format. They are designed for unstructured data, horizontal scaling, and flexible data models. NoSQL databases are ideal for big data and real-time web applications due to their scalability and ability to manage unstructured data like texts and social media posts. For example, social media platforms use NoSQL to manage vast user-generated data. In Tanzania, startups might use NoSQL databases for quickly launching and scaling applications, such as mobile health apps collecting patient data or agribusiness apps tracking crop availability and prices in rural areas.

(b) **NewSQL databases:** NewSQL databases combine the scalability and flexibility of NoSQL with the ACID properties of traditional SQL databases. They're suitable for businesses needing fast, scalable databases for reliable transactions, like online retailing. In Tanzania, NewSQL databases could support e-commerce platforms, ensuring fast, error-free transactions during peak seasons or festivals.

(c) **Graph databases: Specialise** in storing and analyzing interconnected data, beneficial for scenarios like social networks or wildlife conservation efforts in Tanzania.

(d) **Time-Series Databases (TSDB):** are optimised for analyzing time-stamped data, crucial for IoT and financial analysis. In Tanzania, TSDB could monitor soil moisture levels for efficient water management in agriculture.

Emerging technologies

(a) **Database as a Service (DBaaS):** These a) Database as a Service (DBaaS) simplifies database management by hosting databases on the cloud, allowing developers to focus on application development. For instance, a startup in Tanzania can leverage DBaaS to manage customer data without needing physical servers.

FOR ONLINE READING

(b) **Serverless databases** abstract server management and offer a pay-per-use model, benefiting dynamic workloads like mobile apps with fluctuating traffic. Developers in Tanzania can utilize serverless databases for event-based applications, such as festival ticketing systems.

(c) **Blockchain databases** ensure decentralized, secure records through blockchain technology, suitable for applications like supply chain management. In Tanzania, blockchain databases could securely record land transactions or student credentials, ensuring data integrity.

(d) **Edge databases** run closer to data sources for faster processing, beneficial for IoT applications. In Tanzania, edge databases can aid in wildlife tracking or remote patient monitoring in remote areas, where real-time data analysis is crucial.

Emerging applications

Emerging applications such as the Internet of Things (IoT), real-time analytics, artificial intelligence (AI), and machine learning (ML) are transforming various industries. IoT connects everyday devices to the Internet, enabling data collection and automation. Real-time analytics allows for immediate data processing and insights, crucial for dynamic decision-making. AI and ML drive innovations in personalization, recommendation engines, and advanced problem-solving, enhancing user experiences and operational efficiencies. These technologies are vital for modernizing sectors like healthcare, finance, and manufacturing, offering significant benefits such as improved efficiency, better decision-making, and enhanced user engagement.

Activity 8.15:

Exploring emerging applications

Aim:
To research and document the benefits, features, use cases, industrial applications, and trends of emerging technologies.

Materials:
Computer, Internet or Library access.

Instructions
Explore the details of emerging technologies such as IoT, Real-Time Analytics, AI/ML, Personalization and Recommendation Engines, Privacy-Enhanced Technology, and others through Internet and library research. In your exploration, undertake the following tasks:

1. Analyse the primary benefits offered by each emerging application.
2. Describe the distinct features and ideal scenarios where these applications are most effective.
3. Categorize these applications based on their industrial applications and sectors.
4. Evaluate the significance and impact of these trends across various industries.

FOR ON-LINE READING ONLY

5. Write down your findings and document them comprehensively in a portifolio, summarizing key insights and implications for each emerging technology.

Deliverables:

A comprehensive document summarizing findings from the exploration of emerging applications. Include analyses of benefits, features, ideal use cases, categorization based on industrial uses, and evaluations of trends' importance and relevance across industries.

Relational databases, ER Model and SQL

A relational database organises data into tables with rows and columns, allowing for efficient storage and retrieval of information. It uses an Entity-Relationship (ER) model to represent entities and their relationships graphically, such as students, teachers, and classes in a school database. Constraints like unique keys ensure data integrity. Structured Query Language (SQL) is the standard interface for managing and querying data, enabling complex operations like filtering and sorting. Relational databases, often called SQL databases, use fixed schemas and structured data, ensuring consistency and enforcing relationships between data points. Use Activity 8.15 to learn more about relational databases, ER Model and SQL.

Activity 8.16:

Relational Databases, ER Model and SQL

Aim:

To impart relational database concepts, ER modeling, and SQL querying skills using an online database simulation tool, focusing on creating and querying a school library database.

Materials

1. Computer with Internet access
2. Access to an online database simulation tool (e.g., DB Fiddle, SQL Fiddle)
3. Database schema design for a school library (books, members, loans)
4. SQL queries for data manipulation

Instructions:

Use a computer with Internet access to access the SQL Fiddle online database simulation tool (e.g., DB Fiddle: https://www.db-fiddle.com/, SQL Fiddle: https://sqlfiddle.com/) to create and manipulate a database through the steps below.

Step 1: Build a Sample Database

Use an online database simulation tool to construct a database schema for a school library, including tables for Books, Members, and Loans, and define appropriate data types for each attribute.

FOR ONLINE READING ONLY

Step 2: Manipulate Data with SQL

Write SQL queries to retrieve and manipulate data within the library database. This includes querying specific genres of books, retrieving loan information for individual students, and identifying overdue books based on defined criteria.

Deliverables:

Write a summarized report including a well-defined database schema for a school library database, with tables for books, members, and loans, having appropriate attributes and data types specified; An implemented database created using the online database simulation tool, populated with sample data (books, members, loans); Various SQL queries written and executed using the online tool to retrieve specific information from the database, such as books by genre, loans by student, and identification of overdue books. Document your work in a portifolio.

Non-relational (NoSQL) databases

NoSQL databases, unlike relational ones, offer flexible schemas and support various data formats like documents, key-value pairs, and graphs. They do not enforce strict relationships between data points. There are different types of NoSQL databases.

(a) *Document databases:* Ideal for semi-structured data like user profiles, storing data in JSON-like documents.

(b) *Key-value stores:* Efficiently link unique keys with values, great for simple data retrieval such as shopping cart items.

(c) *Wide-column stores:* Handle large datasets with variable structures, often for analytics workloads like sensor data and financial transactions.

(d) Graph databases represent data as graphs, comprising nodes, edges, and properties. Nodes represent entities, edges depict relationships between nodes, and properties provide additional information about nodes and edges. These databases are optimal for managing highly interconnected data, offering flexibility and performance advantages for scenarios such as social networks and recommendation engines.

On the other hand, NoSQL databases are designed for handling large, dynamic datasets that are unstructured or semi-structured. They excel in environments requiring scalability and schema flexibility but may encounter challenges with data consistency due to their distributed nature and eventual consistency models.

In contrast, traditional relational databases efficiently manage structured, relatively static datasets using SQL queries. They prioritize data integrity and consistency, making them ideal for transactional applications like banking and e-commerce, typically hosted on centralized servers.

Big data technologies address the shortcomings of traditional databases

FOR ONLINE READING ONLY

by enabling the storage, processing, and analysis of massive datasets and diverse data types. They are essential for applications that require handling large volumes of data quickly and extracting valuable insights, such as in scientific research and business analytics.

Activity 8.17:

Non-relational (NoSQL) databases

Aim:

To gain practical experience with NoSQL databases by exploring their functionalities and performing basic data operations and queries

Materials:

Computer with Internet access; Access to an online NoSQL database platform (e.g., MongoDB Atlas Free Tier at https://www.mongodb.com/students. Firebase at https://firebase.google.com/)

Instructions:

Step 1: Choose a NoSQL platform

Select a user-friendly online NoSQL database platform such as MongoDB Atlas Free Tier or Firebase. These platforms offer free tiers suitable for educational purposes.

Step 2: Sign Up and create a database

Complete the signup process on your chosen platform and create a new database instance.

Step 3: Explore the platform: Familiarize yourself with the platform's interface and basic functionalities.

Step 4: Document databases exploration

(a) Create database and collections:

(i) Create a new database.

(ii) Create collections (similar to tables in relational databases).

(b) Create sample documents: Within a collection, create sample documents by defining key fields and their values. For example, create a collection for "movies" with documents that store information like title, director, and genre.

(c) Perform data operations: Insert, update, and delete documents using the platform's interface or simple queries.

Step 5: Querying data:

(a) *Use Query Language:* Learn how to use the platform's query language or interface to filter and retrieve data.

(b) *Basic Queries:* Perform basic queries to find specific documents based on criteria (Such as, find all movies directed by a specific director).

(c) *Advanced Filtering and Sorting:* Explore different

FOR ONLINE READING ONLY

filtering options and sorting functionalities to refine your queries.

Deliverables:

Write a brief report detailing the steps taken, the functionalities explored, and your observations on the differences between NoSQL and relational databases. In this report include a created database instance with relevant collections and sample documents. Also, examples of inserted, updated, and deleted documents. Examples of queries used to filter and retrieve data, along with the results should also be included.

Data warehouse

A data warehouse is a specialized system for analyzing large datasets, acting as a central repository that gathers data from various sources within an organisation. Unlike traditional databases focused on current transactions, data warehouses are subject-oriented, organizing data around business subjects like marketing or sales. They integrate and cleanse data to ensure consistency and accuracy. Storing structured, semi-structured, and unstructured data, data warehouses enable historical trend analysis. Optimised for reading and analysis, they support report generation and complex data mining, improving decision-making, enhancing analytics, speeding up analysis, and reducing the burden on operational systems.

Activity 8.18:

Data warehouse - building a mini data warehouse

Aim:

To understand the basics of data warehousing by designing, implementing, and analyzing a simple data warehouse using spreadsheet software.

Materials:

1. Computers with Internet access
2. Spreadsheet software (e.g., Microsoft Excel)
3. Sample datasets (found online or created for the activity)

Instructions:

Step 1: Design the schema

Plan what data you want to store and decide how you will organise it for efficient querying and reporting.

Step 2: Collect data

Use provided sample datasets or gather data from various sources to populate your data warehouse.

Step 3: Perform an ETL (Extract, Transform, Load) Processes

Use spreadsheet software to clean your collected data. Transform the data into a consistent format and then load them into your designed schema.

FOR ONLINE READING ONLY

Step 4: Perform basic queries

Learn how to perform basic queries on your data warehouse, then retrieve specific information and generate reports.

Step 5: Analyse the data

Analyse the data within your warehouse, gain insights about them and identify trends, patterns, or anomalies.

Deliverable:

Prepare a comprehensive report that includes a designed schema for the data warehouse; a cleaned, transformed, and loaded dataset; reports generated from basic queries; and an analysis report highlighting insights, trends, patterns, or anomalies. Document your work on a portifolio.

Big data

Big Data handles huge, complex datasets from various sources like social media, sensors, and machine logs. It includes structured, semi-structured, and unstructured data that can change quickly. Managing and analyzing Big Data uses distributed processing and specialized tools, often with distributed storage systems. Big Data analytics aims to find insights and patterns, useful for real-time analysis, social media opinions, and scientific studies. NoSQL databases are essential in Big Data for storing and accessing large amounts of unstructured data efficiently.

Activity 8.19:

Exploring big data and key benefits

Aim:

To explore and understand the benefits, features, and applications of big data through research and documentation

Materials:

Computer, Internet or library resources

Instructions:

Use Internet and library search, explore to understand the details of big data. In your exploration, do the following:

Step 1: Analyse key benefits

Investigate and analyse the significant advantages brought about by the advent of big data.

Step 2: Describe features and use cases.

Describe the unique characteristics of big data and identify its ideal applications.

Deliverable:

Record your findings systematically, compile them and document in a portfolio.

Activity 8.20:

Exploring big data with social media analytics

Aim:

To explore big data concepts through social media analytics tools to understand data analysis and interpretation.

FOR ONLINE READING ONLY

Materials:
Computers with Internet access, Social media analytics tools (such as, Google Analytics), Spreadsheet software (For example, Microsoft Excel)

Instructions

Step 1: Choose a social media platform

Select a popular social media platform and use its analytics tool to gather data over a defined period. Focus on metrics such as engagement rates, follower growth, and post reach.

Step 2: Organise and visualise data

Utilise spreadsheet software to organise the collected data. Create charts and graphs to visually represent trends and patterns observed in the data.

Step 3: Interpret findings

Each student should interpret their findings and prepare a concise report addressing questions like:

(a) What insights does the data provide about user engagement?

(b) How does the timing of posts influence reach?

(c) Which types of content generate the most interaction?

Step 4: Explore how big data analytics can impact decision-making in various sectors, including business

Deliverable:
A comprehensive report analysing social media data collected using analytics tools. Include visualisations such as charts and graphs illustrating trends and patterns in the data. Also, Answers to specific questions about user engagement, posting times, and content interaction.

The changing database landscape meets the needs of modern applications with scalability, flexibility, and real-time processing. Understanding new technologies is crucial for industries to make better decisions and stay competitive globally.

Exercise 8.7

1. How can you identify that particular kinds of data are "Big Data"?
2. How can you persuade a user to use a Graph Database against a Relational Database and vice versa?
3. What are the need for developers to use serverless databases ?
4. What key attributes do the Blockchain technology enhance in databases
5. Describe a scenario where a Time-Series Database (TSDB) be MOST valuable?
6. Identify applications that finds it necessary to use a Distributed Database?Explain why?

FOR ONLINE READING ONLY

7. What do you understand by the term 'Database as a Service' (DBaaS)?. What valuable contribution does it have?
8. Discuss the contributions brought by the rise of IoT (Internet of Things) devices?

Designing database Using SQL and PHP

In the digital age, data drives decisions, powers applications, and informs our understanding of the world around us. From the apps on our phones to the websites we browse daily and the trending artificial intelligence, databases are the unseen engines powering much of our digital lives. This section covers the role and importance of databases recognizing how databases support the storage, retrieval, and management of data in various aspects of technology and everyday life, together fundamental concepts and basics of creating and manipulating a database using SQL within PHP scripts, establishing a connection to a database, creating tables, inserting data, querying data using PHP and hands-on activities of designing and interacting with databases using SQL (Structured Query Language) and PHP (Hypertext Preprocessor). It enables you to become familiarised with everyday applications that use databases (e.g., social media platforms, online shopping) and create data-driven web applications within your surrounding environment and beyond using SQL and PHP.

Introduction to databases and SQL

This part covers an introduction to databases and SQL starting with the basics on what the databases are, why they are essential, and how to perform simple operations using SQL.

Understanding databases

A database is a structured collection of data that is stored and accessed electronically from a computer system. Databases are managed through a Database Management System (DBMS), which provides users the tools to interact with the database's data. These DBMSes offer various functionalities, such as data creation, querying, updating, and administration of databases, ensuring data integrity, security, and performance.

Importance of databases in modern web and mobile applications

Databases are very important for modern web and mobile apps because they store and manage data efficiently, which is essential for how users interact with the apps and how the apps work.

Activity 8.21:

Understanding the role of databases in modern applications

Aim:

To investigate and comprehend the significance of databases in contemporary web and mobile applications.

FOR GOVERNMENT USE ONLY

Materials:

Computer with Internet access; Access to online databases (optional for further exploration), Access to library resources.

Instructions

Explore the Internet and library resources to understand the critical role of databases in modern web and mobile applications. Record your discoveries and document them into a portfolio.

SQL databases

SQL databases, also known as relational databases, organise data into tables which can be linked—or related—based on data common to each. This structure allows for the relational database management systems (RDBMS) to access and assemble data points stored across different tables through SQL queries. SQL databases are characterized by their use of schema, a predefined structure for data that dictates the format of the data stored.

Key features of SQL databases

(i) **Structured data:** SQL databases are highly structured and require data to adhere to predefined schemas, which specify the tables, columns, data types, and relationships among data.

(ii) **ACID properties:** They strongly adhere to ACID (Atomicity, Consistency, Isolation, Durability) properties, ensuring reliable transaction processing.

(iii) **Complex queries:** SQL databases are powerful in handling complex queries, making them suitable for applications where data interrelation is essential.

Examples of SQL databases

MySQL: Widely used open-source RDBMS, suitable for a variety of applications.

PostgreSQL: An open-source, feature-rich RDBMS known for its extensibility and standards compliance.

Oracle Database: A comprehensive commercial RDBMS solution for enterprise-level applications.

Microsoft SQL Server: A commercial RDBMS with extensive tools and features for businesses.

Use Cases of SQL databases

(i) Banking systems for transactional data processing.

(ii) Enterprise applications with complex queries and reporting.

(iii) E-commerce platforms need consistent data integrity and relational operations.

MySQL (relational database management system)

MySQL is one of the most popular relational database management systems (RDBMS) in the world, widely used for

FOR ONLINE READING ONLY

a variety of applications, from small projects to large-scale enterprise solutions.

As an open-source system, MySQL has been instrumental in the proliferation of database-driven websites and applications, thanks to its robust feature set, reliability, and ease of use. Since its inception, it has grown in popularity and usage, becoming a foundational component of the LAMP stack (Linux, Apache, MySQL, PHP/Perl/Python), which powers a significant portion of the web.

Key features of MySQL

(i) ***Open Source:*** One of MySQL's most compelling features is that it is open-source, allowing developers to use and modify the software freely, which has contributed significantly to its popularity and development.

(ii) ***Cross-platform support:*** MySQL is highly versatile, with support for major operating systems, including Linux, Windows, macOS, and Unix, making it suitable for a wide range of applications and environments.

(iii) ***High performance:*** MySQL is known for its high performance, reliability, and efficient data management, even under demanding applications. It includes various storage engines, such as InnoDB (which supports ACID-compliant transactions) and MyISAM, among others, allowing it to be highly customizable for different workload requirements.

(iv) ***Scalability:*** MySQL supports both vertical and horizontal scaling, making it adaptable to applications of different sizes and requirements. It can manage deeply embedded applications as well as massive data warehouses holding terabytes of information.

(v) ***Security:*** MySQL boasts strong data security layers that protect sensitive data from unauthorized access. Features like SSL encryption, user account management, and access control lists (ACLs) are integral parts of its security framework.

(vi) ***Comprehensive application development:*** One of MySQL's strong points is its support for stored procedures, triggers, functions, views, and more, allowing for complex application development directly within the database environment.

Usage of MySQL

MySQL is widely used for web applications and high-traffic websites like Facebook, Twitter, and YouTube, demonstrating its ability to handle large-scale, data-intensive applications. In Tanzania, MySQL supports economic growth, education, and innovation. It's used in government and public sectors for citizen registration, healthcare, and education data management, enabling efficient public services. Small and Medium Enterprises (SMEs) leverage

FOR ONLINE READING ONLY

MySQL for e-commerce and online business operations. The financial sector, including banks and fintech startups, uses MySQL for transaction data and analytics. Telecommunications companies and ISPs use MySQL for managing subscriber data, service provisioning, and billing systems, benefiting from its scalability and reliability to deliver quality services.

Community and ecosystem

MySQL has a vibrant community of developers and users who contribute to its ongoing development and provide support through forums, blogs, and third-party tools. This community-driven approach has resulted in a rich ecosystem of tools, documentation, and resources that help users manage and optimise MySQL databases effectively.
MySQL installation when using local environments

As earlier indicated MySQL is a popular open-source Relational Database Management System (RDBMS) that allows you to efficiently store and manage large amounts of structured data and it is widely used by developers, data analysts, and system administrators for various purposes, including building web applications, managing content, and performing data analysis. Using it with local environments requires the installation which can be on different operating systems though here and in this subject we focus on Windows as indicated below in Activity 8.6.

Activity 8.22:

Installing MySQL using local environments

Materials Used:

Computer with Windows operating system; Internet access; MySQL Installer downloaded from https://www.mysql.com

Aim:

To guide users through the installation of MySQL Community Server on a local Windows environment, enabling them to set up and configure MySQL for database management and development purposes.

Instructions:

Step 1: Download MySQL Installer

(a) Visit https://www.mysql.com and download MySQL Community Server edition.

(b) Run the installer and choose "Developer Default" setup.

Step 2: Select Products to Install

Choose "MySQL Server" and select the latest version for installation.

Step 3: Configuration

(a) Configure installation type as "Server only" and choose port number (default is 3306).

(b) Set a strong root password for MySQL server access.

Step 4: Installation

(a) Execute the installation process after configuring details.

FOR ONLINE READING ONLY

(b) MySQL Installer will download and install selected products.

Step 5: Completing the installation

(a) Review the installation summary screen.

(b) Check "Launch MySQL Instance Configuration Wizard" and click "Finish."

Step 6: MySQL instance configuration

(a) Follow Configuration Wizard steps to configure MySQL server.

(b) Choose "Detailed Configuration" and select "Server Machine" as server type.

Step 7: Authentication method

Opt for "Use Strong Password Encryption" and set or confirm root password.

Step 8: Windows service

(a) Select "Install as Windows Service" and specify service name (or use default).

(b) Check "Launch the MySQL Server automatically" for automatic startup.

Step 9: Apply configuration

(a) Review configuration summary and click "Execute" to apply changes.

(b) Click "Finish" upon completion.

Deliverables

Write a brief report that includes the following items:

(a) Installed MySQL Community Server with configured settings.

(b) Ability to create, manage databases, tables, and perform data operations locally using MySQL, include screenshots to justify this.

Practicing using basics of SQL

Follow these steps to learn SQL concepts and operations using a farming example. Create a database named 'farming_database' with tables 'farmers' and 'crops'. Use a MySQL client or an online platform such as W3Schools to execute these commands and establish the necessary schema and table relationships.

Data Definition Language (DDL)

Activity 8.23:

Creating and managing a SQL database for Farming Data

Aim:

Understanding how to create and manage a SQL database, including the creation of tables and the establishment of relationships between them.

Materials

1. Computers with Internet access
2. MySQL client or command-line interface installed (such as MySQL Workbench, phpMyAdmin) or

FOR ONLINE READING ONLY

access to an online SQL editor (for example, W3Schools)

3. Sample datasets for farming (optional for data entry exercises)

Instructions

Create the database and table using the following steps:

Step 1: Create the "farming_database" using the CREATE DATABASE statement. The codes looks such as:

```
CREATE DATABASE farming_database;
```

Step 2: Switch to this new database using the USE statement. The codes looks as:

```
USE farming_database;
```

Step 3: Create the "farmers" table using the CREATE TABLE statement, defining the columns and their data types. The codes looks as:

```
CREATE TABLE farmers (
  id INT PRIMARY KEY,
  name VARCHAR(50) NOT NULL,
  age INT,
  location VARCHAR(100),
  contact VARCHAR(20)
);
```

Step 4: Create the "crops" table, including a "farmer_id" column that serves as a foreign key referencing the "id" column in the "farmers" table using the FOREIGN KEY constraint.

The codes will look as follows:

```
CREATE TABLE crops (
    id INT PRIMARY KEY,
    name VARCHAR(50) NOT NULL,
    type VARCHAR(50) NOT NULL,
    planting_date DATE,
    harvest_date DATE,
    yield INT,
    price DECIMAL(10, 2),
    farmer_id INT,
    FOREIGN KEY (farmer_id)
    REFERENCES farmers(id)
);
```

The above steps will set up the "farming_database" with the "farmers" and "crops" tables.

Deliverables

Write a comprehensive report that include the following items:

(a) SQL Script file: A file containing the SQL statements used to create the "farming_database", "farmers" table, and "crops" table.

(b) Database schema diagram: A visual diagram illustrating the relationships between the "farmers" and "crops" tables, including primary and foreign keys.

(c) Execution report: A document detailing the steps taken to execute the SQL statements, with screenshots or logs showing the creation of the database and tables.

(d) Data entry examples (optional): If sample datasets were used, provide

FOR ONLINE READING ONLY

examples of data entries for the "farmers" and "crops" tables.

(e) Reflection summary: A brief summary reflecting on the experience, any challenges faced during the activity, and what was learned from creating and managing the SQL database.

Finally, make sure you document your work in a portifolio.

Activity 8.24:

Creating an index and view in SQL.

Aim:

To learn how to optimise data retrieval using indexes and how to create a view to combine and display related data from multiple tables in a SQL database.

Materials

1. Computer with Internet access
2. SQL database management system (such as MySQL, PostgreSQL)
3. SQL client software or an online SQL editor (e.g., phpMyAdmin, MySQL Workbench, or an online platform like W3Schools)
4. Sample dataset for the "farmers" and "crops" tables

Instructions

To add an index to a column in the "crops" table and create a view based on the data in the database, follow these steps:

Step 1: Create an index

Add an index to the 'name' column in the 'crops' table to speed up data retrieval operations, the code looks as:

```
CREATE INDEX idx_crops_name ON
crops(name);
```

This statement creates an index called "idx_crops_name" on the "name" column in the "crops" table. Indexes improve the speed of data retrieval operations on specific columns.

Step 2: Create a view

Create a view to combine data from the "farmers" and "crops" tables based on their relationship, codes look as:

```
CREATE VIEW crop_farmer_
information AS
    SELECT c.id AS crop_id, c.name
        AS crop_name, c.planting_date,
        c.harvest_date, c.yield, c.price,
        f.id AS farmer_id, f.name AS
        farmer_name, f.age, f.location,
        f.contact
    FROM crops c
    JOIN farmers f ON c.farmer_id = f.id;
```

Note: This statement creates a view named "crop_farmer_information" that combines data from the "crops" and "farmers" tables. The view includes columns from both tables and joins the data based on the "farmer_id" foreign key relationship.

Deliverables

Write a comprehensive report including the following:

FOR ONLINE READING ONLY

(a) SQL script file: A file containing the SQL statements used to create the database, tables, index, and view.

(b) Database schema diagram: A diagram showing the relationships between the "farmers" and "crops" tables.

(c) Execution report: A document detailing the steps taken to execute the SQL statements, including screenshots of the process and any output generated.

(d) View query results: A report or screenshot displaying the results of querying the "crop_farmer_information" view.

(e) Reflection: A brief summary reflecting on the experience, challenges faced, and what was learned from the activity.

Finally, document your work in a portifolio.

ALTER: Modify the structure of an existing database object.

In the context of the farming database let us modify the structure of the "farmers" table by adding a new column called "email" to store the email addresses of the farmers. To accomplish this, we use the ALTER TABLE statement to modify the existing table.

Activity 8.25:

Altering the structure of an SQL database

Aim:

Learn how to modify the structure of an existing database table by adding new columns.

Materials:

Computers with Internet access; MySQL client or command-line interface (such as MySQL Workbench, phpMyAdmin) or access to an online SQL editor (e.g., W3Schools)

Instructions

Step 1: Open your SQL client or online SQL editor.

Step 2: Use the following SQL command to add a new column to the "farmers" table:

```
ALTER TABLE farmers ADD COLUMN email
VARCHAR(50);
```

Step 3: Verify the changes by querying the structure of the "farmers" table, codes look as:

```
DESCRIBE farmers;
```

Deliverables

Write a brief report which includes the following:

(a) SQL script that alters the "farmers" table by adding a new column.

(b) Screenshot or output showing the updated structure of the "farmers" table with the new "email" column.

Finally, document all your work in a portifolio.

FOR ONLINE READING ONLY

DROP: Delete a database, table, index, or view

In the context of the farming database example, assume you want to use the DROP statement to delete an index in the "crops" table.

Activity 8.26:

Deleting an Index from a SQL database

Aim:
Understanding how to delete an index from a database table using the DROP statement.

Materials:
Computer with Internet access; MySQL client or command-line interface (such as MySQL Workbench, phpMyAdmin) or access to an online SQL editor (Such as, W3Schools)

Instructions:

Step 1: Open your SQL client or online SQL editor.

Step 2: Use the following SQL command to drop an index from the "crops" table. Codes look as:

```
DROP INDEX idx_crops_name ON
crops;
```

Step 3: Verify that the index has been dropped by checking the structure of the "crops" table. Codes look as:

```
SHOW INDEX FROM crops;
```

Deliverables:

Prepare a brief report tha includes the following:

(a) Documentation: Detailed documentation of the process involved in dropping an index from the "crops" table in the farming database.

(b) Screenshots: Screenshots or output from the SQL client or online SQL editor showing the execution and successful dropping of the index.

(c) Reflection: A brief reflection on the significance of using the DROP statement in database management, discussing potential benefits and considerations when modifying database structures.

Finally, document all your workings in a portifolio.

PHP Fundamentals

PHP (Hypertext Preprocessor) is widely used for developing dynamic websites and interacting with databases. By understanding PHP, you can create dynamic web pages, handle user input, and manipulate data stored in databases more effectively. This is done effectively with earlier covered basic understanding of HTML which is the markup language used for creating website structure, CSS which is responsible for styling web pages, and databases which stores and organises data efficiently for different applications.

FOR ONLINE READING ONLY

Basics of PHP: syntax, commenting, variables, and the Echo statement

In this section, we cover the basics of PHP, including its syntax, commenting, variables, and the echo statement. Understanding these fundamental concepts lays the foundation for your PHP programming journey in the context of database management.

(a) PHP syntax

(i) PHP code is written inside <?php ?> tags. For example: <?php ...code... ?>

(ii) Statements in PHP are terminated with a semicolon (;).

(iii) PHP is not case-sensitive, so you can use either uppercase or lowercase letters for language constructs.

(b) Commenting

(i) Comments in PHP are used to document or temporarily disable parts of the code.

(ii) Single-line comments start with // or #. For example: // This is a comment.

(iii) Multi-line comments are enclosed between /* and /. For example:

```
/
This is a multi-line comment.
It can span multiple lines.
*/
```

(c) Variables

(i) Variables are used to store data and give them meaningful names.

(ii) In PHP, variables start with a dollar sign ($), followed by the variable name. For example: $name, $age.

(iii) Variable names can contain letters, digits, and underscores, but they must start with a letter or underscore.

(iv) PHP uses dynamic typing, meaning you don't need to declare the variable type explicitly.

(v) Assigning a value to a variable is done using the assignment operator (=). For example: $name = "John";

(d) The echo statement

(i) The echo statement is used to output text or variables onto the web page.

(ii) To display a string literal, enclose it in quotes (single or double). For example echo "Hello, World!";

FOR ONLINE READING ONLY

(iii) You can also display the value of a variable using the echo statement. For example: $name = "John"; echo $name;

(iv) Multiple values can be concatenated within the echo statement using the period (.) operator. For example: echo "Hello, ". $name;

(v) The echo statement does not require enclosing parentheses.

An example that combines these concepts is shown in Program Example 8.1:

Program Example 8.1:

PHP program to demonstrate echo statement

```
php
<button><svg><path></path></svg><span>Copy code</span><span></span></button>
<?php
    // This is a single-line comment
    /*This is a  multi-line comment */
      $name = "John"; // Assigning a value to the $name variable
      $age = 25;   // Assigning a value to the $age variable
      echo "Hello, World!";              // Outputting a string literal
      echo "My name is ", $name , ".";    // Concatenating a string with a variable
      echo "I am ", $age , " years old."; // Outputting variables with text
 ?>
```

Output:

Hello, World!
My name is John.
I am 25 years old.

Understanding PHP syntax, commenting, variables, and echo statements is crucial as you progress with PHP programming in connection with database management.

The concept of server-side scripting in the web development

Server-side scripting is a web server technology in which a user's request is fulfilled by running a script directly on the web server to generate dynamic web pages. It is a key component of web development that enables the creation of customized responses for each user request. Unlike client-side scripting where scripts are executed by the browser, in server-side scripting, the script is executed on the server and the client only sees the resulting HTML.

FOR ONLINE READING ONLY

Implementation of server-side Scripting

(a) **User request:** A user requests a page from their web browser by clicking a link, submitting a form, or entering a URL.

(b) **Server process:** The web server receives the request and finds the appropriate server-side script to handle it.

(c) **Script execution:** The script runs on the server, performing actions such as querying a database, processing data, or any other logic necessary to generate the response.

(d) **HTML generation:** The script produces HTML (and possibly CSS and JavaScript) as output.

(e) **Response sent:** The web server sends this generated HTML back to the browser as the response to the original request.

(f) **Browser rendering:** The user's browser renders the HTML, presenting the web page to the user.

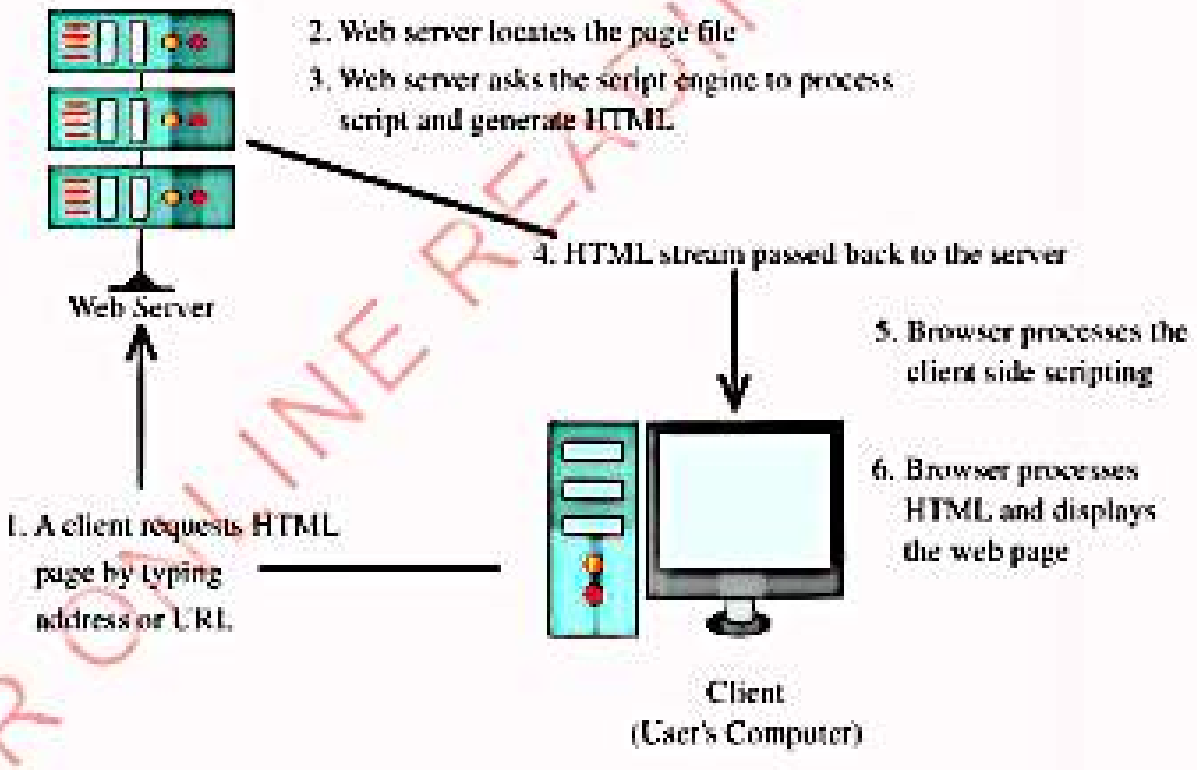


Figure 8.25: *Server-side scripting*

PHP's role in web development stack

PHP is one of the most popular server-side scripting languages designed specifically for web development. It fits into the server side of the web development stack, interacting with databases, managing session data, and performing condition-based operations to generate dynamic web content. PHP scripts can be embedded directly within HTML documents, allowing for a seamless combination of static and dynamic content.

FOR ONLINE READING ONLY

Characteristics that make PHP well-suited for web development

(a) **Ease of use:** PHP's syntax is clear and easy to learn, especially for those familiar with C, Java, or HTML, making it an excellent choice for beginners and professionals alike.

(b) **Flexibility:** PHP is platform-independent; it can run on various operating systems like Windows, Linux, and macOS, and supports a wide range of databases (including MySQL and PostgreSQL).

(c) **Integration:** PHP is compatible with virtually all web servers used today (e.g., Apache) and can be easily integrated with other technologies used in web development, like JavaScript, HTML, and CSS.

(d) **Prebuilt solutions:** There is a vast ecosystem of frameworks (Laravel, Symfony), CMS (WordPress, Drupal), and tools built with or for PHP, speeding up the development process and providing robust solutions for common tasks.

(e) **Community and documentation:** A large and active community means a wealth of resources, forums, tutorials, and documentation are available for learning and troubleshooting.

Generally, PHP fits into the server side of the web development stack, where it is used to develop dynamic web applications. It runs on the web server, processes user requests by performing operations such as reading from or writing to a database, and generates the HTML content that is sent back to the client. PHP's power, ease of use, and flexibility, along with its vast ecosystem, make it a cornerstone of modern web development, capable of building anything from small websites to large-scale web applications.

Embedding PHP in HTML and CSS to create dynamic content

Building a website using PHP involves integrating PHP code with HTML and CSS. HTML provides the structure and semantics of the web page, while CSS is used to style and enhance the visual appeal. PHP is embedded within HTML tags and allows for dynamic content generation and server-side processing. To embed PHP code within HTML and CSS to create dynamic content, use the following steps:

Step 1: Create a PHP file with the necessary PHP code to generate dynamic content. Program Example 8.2 shows how you can use PHP to fetch data from a database and display it on the webpage.

FOR ONLINE READING ONLY

Program Example 8.2:

PHP program to generate dynamic content

```
<?php
    // Connect to database
    $servername = "localhost";
    $username = "username";
    $password = "password";
    $dbname = "database";
    $conn = new mysqli($servername, $username, $password, $dbname);
    // Check connection
  if ($conn->connect_error) {
  die("Connection failed: " . $conn->connect_error);
 }
   // Fetch data from the database
   $sql = "SELECT * FROM table";
   $result = $conn->query($sql);
   // Display data on the webpage
   echo "<ul>";
   while($row = $result->fetch_assoc()) {
      echo "<li>" . $row["column_name"] . "</li>";
   }
   echo "</ul>";
   // Close connection
   $conn->close();
?>
```

Step 2: Embed the PHP code within the HTML file using the `<?php ?>` tags are shown in Program Example 8.3.

FOR ONLINE READING ONLY

Program Example 8.3:

PHP program to embed PHP code within HTML

```
<!DOCTYPE html>
<html>
<head>
  <title>Dynamic Content</title>
   <style>
    /* CSS styles */
    ul {
      list-style-type: none;
      padding: 0;
    }
    li {
      padding: 10px;
      background-color: #f0f0f0;
      margin-bottom: 5px;
    }
  </style>
</head>
<body>
  <h1>Dynamic Content</h1>
  <?php include 'dynamic_content.php'; ?>
</body>
</html>
```

FOR ONLINE READING ONLY

Step 3: Save the PHP file as `dynamic_content.php` and the HTML file as `index.php` in the same directory.

Step 4: Open the `index.php` file in a web browser to see the dynamic content generated by the PHP code embedded within the HTML and styled using CSS.

By embedding PHP code within HTML and CSS, you can create dynamic content that can be updated and displayed on the webpage based on user interactions or backend data.

An example of how to embed PHP in HTML and CSS to create dynamic content using an imaginary "farming" database with a table called "crops" containing columns like "image_url", "description", "crop_name", and "harvest_season". The PHP code connects to the database, fetches the data, and then the HTML file embeds the PHP code to display the dynamic content in a gallery format using CSS. It is as follows:

First, create a PHP file to connect to the database and fetch data as shown in Program Example 8.4:

Program Example 8.4:

PHP program to connect to the database and fetch data

```
<?php
// Connect to the farming database
$servername = "localhost";
$username = "username";
$password = "password";
$dbname = "farming_database";
// Create connection
$conn = new mysqli($servername, $username, $password, $dbname);
// Check connection
if ($conn->connect_error) {
    die("Connection failed: " . $conn->connect_error);
}
// Fetch data from the farming database
$sql = "SELECT * FROM crops";
$result = $conn->query($sql);
// Close the database connection
$conn->close();
?>
```

FOR ONLINE READING ONLY

Then, embed the PHP code within the HTML file to display the dynamic content as shown in Program Example 8.5:

Program Example 8.5:

PHP program to embed the PHP code within the HTML file to display the dynamic content

```
<!DOCTYPE html>
<html>
<head>
  <title>Farming Crops</title>
  <style>
    /* CSS styles */
    .crop {
      border: 1px solid #ccc;
      padding: 10px;
      margin: 10px;
      display: inline-block;
    }
  </style>
</head>
<body>
  <h1>Farming Crops</h1>
  <div class="crop-gallery">
    <?php
    // Display data fetched from the farming database
    while($row = $result->fetch_assoc()) {
      echo '<div class="crop">';
      echo '<img src="' . $row["image_url"] . '" alt="' . $row["description"] . '">';
      echo '<h3>' . $row["crop_name"] . '</h3>';
      echo '<p>' . $row["harvest_season"] . '</p>';
      echo '</div>';
    }
    ?>
  </div>
</body>
</html>
```

FOR ONLINE READING ONLY

In this example when you open the HTML file in a web browser, it will display the dynamic content fetched from the database and styled using the CSS. This allows you to create a dynamic gallery of farming crops based on the data stored in this database.

Activity 8.27:

Creating dynamic content with php

Aim:

To demonstrate the integration of PHP within HTML and CSS to create dynamic web content, including displaying text and current date dynamically retrieved from a server-side script.

Materials:

1. Computer with Internet access
2. PHP development environment (Such as XAMPP, WAMP)
3. Text editor (Such as Visual Studio Code, Sublime Text)

Instruction:

Develop a basic PHP script to showcase "Welcome to the computer science course!" along with today's date on a web page. Additionally, integrate PHP within HTML and CSS to generate dynamic content from a local database table with four columns.

Deliverables:

(a) PHP script displaying "Welcome to the computer science course!" and the current date.

(b) HTML page integrating PHP to display dynamic content.

(c) Optionally, demonstrate the use of PHP to fetch and display data from a local database table.

Integrating or connecting PHP with SQL database

This section connects PHP to an SQL database, enabling you to create, read, update, and delete database records through a PHP application.

PHP and SQL can be integrated for web development. When a user requests a web page that needs database access, the PHP script on the server retrieves the data from the database and generates the HTML code for the page. Thus, Integrating PHP with SQL typically involves connecting to a database using PHP code and then executing SQL queries to retrieve or manipulate data. To connect PHP to MySQL, you can use the MySQLi extension or PDO (PHP Data Objects). MySQLi is object-oriented and only works with MySQL databases, while PDO can work with different database systems. These methods allow developers to create, read, update, and delete database records.

Case 1: Connecting PHP to MySQL using MySQLi function

Connecting PHP to SQL systems like MySQL using the MySQLi extension is a common approach. It involves setting up a connection by providing details

FOR ONLINE READING ONLY

such as the MySQL server's hostname, port number, and authentication credentials. Once connected, PHP can execute SQL statements to manage database operations. The following is an example using a farming database named "farming_db" with a table named "crops":

Connecting to a database

PHP can be connected to database using MySQL. You can establish a connection to the MySQL database as shown in Program Example 8.6.

Program Example 8.6:

PHP program to connect to a database

```
php
<button><svg><path></path></svg><span>Copy code</span><span></span></button>
$serverName = "localhost";
$username = "root";
$password = "";
$databaseName = "farming_db";

// Create connection
$conn = mysqli_connect($serverName, $username, $password, $databaseName);

// Check connection
if (!$conn) {
    die("Connection failed: " . mysqli_connect_error());
}
echo "Connected successfully<br>";
```

FOR ONLINE READING ONLY

2. Inserting data into the database

Next, add a new crop record to the crops table. This will insert data into the database as shown in Program Example 8.7.

Program Example 8.7:

PHP program to insert data to a database

```
php
<button><svg><path></path></svg><span>Copy code</span><span></span></button>
$sql = "INSERT INTO crops (name, yield_per_hectare) VALUES ('Wheat', 4000)";
```

```
if (mysqli_query($conn, $sql)) {
    echo "New crop record created successfully<br>";
} else {
    echo "Error: ". $sql. "<br>" . mysqli_error($conn);
}
```

3. Fetching data from the database

Now, fetch and display all crop records from the crops table as shown in Program Example 8.8

Program Example 8.8:

PHP program to fetch data from a database

```
php
<button><svg><path></path></svg><span>Copy code</span><span></span></button>
$sql = "SELECT id, name, yield_per_hectare FROM crops";
$result = mysqli_query($conn, $sql);

if (mysqli_num_rows($result) > 0) {
    // Output data of each row
    while($row = mysqli_fetch_assoc($result)) {
        echo "id: ". $row["id"]. " - Name: " . $row["name"]. " - Yield per hectare: " .
$row["yield_per_hectare"] . "kg<br>";
    }
} else {
    echo "0 results found<br>";
}
```

4. Closing the database connection

Finally, close your database connection when you're finished with it.

```
php
<button><svg><path></path></svg><span>Copy code</span><span></span></button>
mysqli_close($conn);
```

Combining all the snippets in this example, you have a complete PHP script that connects to a MySQL database, inserts a new record, retrieves all users, and closes

FOR ONLINE READING ONLY

the connection. Program Exampe 8.9 shows how the complete script looks like when you combine all four parts:

Program Example 8.9:

PHP program for all four parts

```
php
<button><svg><path></path></svg><span>Copy code</span><span></span></button>
<?php
$serverName = "localhost";
$username = "root";
$password = "";
$databaseName = "farming_db";
// Create a connection
$conn = mysqli_connect($serverName, $username, $password, $databaseName);

// Check connection
if (!$conn) {
    die("Connection failed: " . mysqli_connect_error());
}
echo "Connected successfully<br>";
// Insert a new crop
$sql = "INSERT INTO crops (name, yield_per_hectare) VALUES ('Wheat', 4000)";
if (mysqli_query($conn, $sql)) {
    echo "New crop record created successfully<br>";
} else {
    echo "Error: " . $sql . "<br>" . mysqli_error($conn);
}
// Fetch and display all crops
$sql = "SELECT id, name, yield_per_hectare FROM crops";
$result = mysqli_query($conn, $sql);
if (mysqli_num_rows($result) > 0) {
    while($row = mysqli_fetch_assoc($result)) {
        echo "id: " . $row["id"] . " - Name: " . $row["name"] . " - Yield per hectare: " . $row["yield_
per_hectare"] . "kg<br>";
    }
} else {
    echo "0 results found<br>";
}
// Close the connection
mysqli_close($conn);
?>
```

FOR ONLINE READING ONLY

Case 2: Connecting PHP to MySQL using PDO

Using PDO (PHP Data Objects) for database interactions in PHP applications provides a uniform method to access databases. PDO supports multiple database systems like MySQL, PostgreSQL, SQLite, etc., making it a flexible choice for developers. It also offers a simple and consistent API across different database systems, making it an ideal choice for developers working with multiple databases.

This example is based on the same farming database to demonstrate the basics of using PDO with MySQL, covering connecting to a database, inserting data, querying and fetching data, and disconnecting.

Setting up a database connection

First, establish a connection to your database using PDO, as shown by codes in Program Example 8.10.

Program Example 8.10:

PHP program to establish a connection to database using PDO

```
php
<button><svg><path></path></svg><span>Copy code</span><span></span></button>
<?php
$host = 'localhost';
$db   = 'farming_db'; // Example database name
$user = 'root';
$pass = '';
$charset = 'utf8mb4';

$dsn = "mysql:host=$host;dbname=$db;charset=$charset";
$options = [
    PDO::ATTR_ERRMODE            => PDO::ERRMODE_EXCEPTION,
    PDO::ATTR_DEFAULT_FETCH_MODE => PDO::FETCH_ASSOC,
    PDO::ATTR_EMULATE_PREPARES   => false,
];

try {
     $pdo = new PDO($dsn, $user, $pass, $options);
     echo "Connected successfully";
} catch (\PDOException $e) {
     throw new \PDOException($e->getMessage(), (int)$e->getCode());
}
?>
```

FOR ONLINE READING ONLY

Inserting data into the database

To insert data securely, use prepared statements to prevent SQL injection, PHP codes in Program Example 8.11 shows how to insert data securely:

Program Example 8.11:

PHP program to insert data securely

```
php
<button><svg><path></path></svg><span>Copy code</span><span></span></button>
$sql = "INSERT INTO crops (name, yield_per_hectare) VALUES (?, ?)";
$stmt = $pdo->prepare($sql);
$stmt->execute(['Corn', 3000]);
echo "New record created successfully";
```

Fetching data from the database

You can query and fetch data. You can also use prepared statements for fetching data as shown in Program Example 8.12, especially when including variables in your query:

Program Example 8.12:

PHP program to insert data securely

```
php
<button><svg><path></path></svg><span>Copy code</span><span></span></button>
$sql = "SELECT id, name, yield_per_hectare FROM crops";
$stmt = $pdo->prepare($sql);
$stmt->execute();
while ($row = $stmt->fetch()) {
    echo "id: ".$row['id']." - Name: ".$row['name']." - Yield per hectare:
".$row['yield_per_hectare']."kg<br>";
}
```

Disconnecting from the database

With PDO, the connection is automatically closed when the PDO object is destroyed, which usually happens when the script ends. However, if you wish to explicitly close the connection within a script, you can set the PDO object to null.

```
;
php
<button><svg><path></path></svg><span>Copy code</span><span></span></button>
$pdo = null;
```

FOR ONLINE READING ONLY

Combining all of the above you get a complete example of using **PDO** to connect to a MySQL database, insert a record, fetch data, and disconnect, as shown in Program Example 8.13:

Program Example 8.13:

PHP program with combined using PDO to connect to MySql

```
php
<button><svg><path></path></svg><span>Copy code</span><span></span></button>
<?php
$host = 'localhost';
$db   = 'farming_db';
$user = 'root';
$pass = '';
$charset = 'utf8mb4';
$dsn = "mysql:host=$host;dbname=$db;charset=$charset";
$options = [
    PDO::ATTR_ERRMODE            => PDO::ERRMODE_EXCEPTION,
    PDO::ATTR_DEFAULT_FETCH_MODE => PDO::FETCH_ASSOC,
    PDO::ATTR_EMULATE_PREPARES   => false,
];
try {
    $pdo = new PDO($dsn, $user, $pass, $options);
    echo "Connected successfully<br>";
    // Insert a new record
    $sql = "INSERT INTO crops (name, yield_per_hectare) VALUES (?, ?)";
    $stmt = $pdo->prepare($sql);
    $stmt->execute(['Corn', 3000]);
    echo "New record created successfully<br>";
    // Fetch records
    $sql = "SELECT id, name, yield_per_hectare FROM crops";
    $stmt = $pdo->prepare($sql);
    $stmt->execute();
    while ($row = $stmt->fetch()) {
        echo "id: {$row['id']} - Name: {$row['name']} - Yield per hectare:
{$row['yield_per_hectare']}kg<br>";
    }
} catch (\PDOException $e) {
    throw new \PDOException($e->getMessage(), (int)$e->getCode());
}
// Close the connection
$pdo = null;
?>
```

FOR ONLINE READING ONLY

Activity 8.28:

Building a task management web application with PHP, PDO, and MySQL

Aim:

To develop proficiency in PHP and PDO for performing CRUD operations on a MySQL database while creating a functional web-based task management application.

Materials:

1. Computers with Internet access
2. Web server environment (Such as XAMPP, WAMP, or any web server with PHP and MySQL support)
3. Text editor or IDE for PHP development
4. MySQL database management tool (Such as phpMyAdmin)
5. Sample data for testing (optional)

Instructions

Create a practical web application using PHP and PDO to perform CRUD operations on a MySQL database. Develop a "Task List" application where users can manage tasks by adding new tasks, viewing a list of tasks with status, updating task status, and deleting tasks. This activity aims to understand PHP and PDO for database interactions, practice CRUD operations, and build a basic but functional web application using PHP.

Step 1: Database setup

(a) Create a MySQL database named task_manager.

(b) Within task_manager, create a table named tasks with fields: id (INT, Primary Key, Auto Increement), task_name (VARCHAR(255)), and is_completed (BOOLEAN or TINYINT defaulting to 0).

Step 2: PHP application setup:

(a) Create index.php for listing tasks.

(b) Create add_task.php for adding new tasks.

(c) Create update_task.php for updating task completion status.

(d) Create delete_task.php for deleting tasks.

Step 3: CRUD Operations:

(a) Create (Add new task): Implement a form in add_task.php to insert new tasks into the tasks table using PDO.

(b) Read (List tasks): Query all tasks in index.php and display them in a list format showing task names and completion status. Include functionality for "Mark as Completed/Not Completed", "Edit", and "Delete".

(c) Update (Mark task as completed/not completed): Develop logic in update_task.php to toggle the completion status of tasks based on user interaction.

(d) Delete (Remove a Task): Implement functionality in delete_task.php to delete tasks from the tasks table.

FOR ONLINE READING ONLY

Step 4: Testing

(a) Test adding new tasks through add_task.php.

(b) Verify the list of tasks and their completion status in index.php.

(c) Test updating task status using update_task.php.

(d) Ensure deletion of tasks functions correctly with delete_task.php.

Deliverables:

Write a brief report that includes the following:

1. PHP files: index.php, add_task.php, update_task.php, delete_task.php
2. MySQL database named task_manager with a table named tasks
3. Functional web application that allows users to Add new tasks, View a list of tasks with completion status, Update task completion status, Delete tasks

Exercise 8.8

1. Describe about PHP and its role in web development??
2. What advantages does PDO offer compared to MySQLi for developers?
3. How can you connect to a MySQL database using PDO in PHP?
4. How would you fetch data from a database using PDO?

NoSQL databases

NoSQL emerged as a response to the limitations of SQL databases, particularly in handling large volumes of unstructured or semi-structured data, and the need for more flexible data management solutions. NoSQL databases do not adhere to a fixed schema, making them highly adaptable to changes and variations in data types.

Key Features of NoSQL databases

(i) **Schema-less data:** They allow the storage of data without a predefined schema, offering flexibility in handling various data formats.

(ii) **Scalability:** NoSQL databases are designed to scale out using a distributed architecture, making them adept at handling large, rapidly growing datasets across multiple servers.

(iii) **Types:** There are several types of NoSQL databases, including document databases, key-value stores, wide-column stores, and graph databases, each optimised for specific types of data and access patterns.

Examples of NoSQL databases

MongoDB (document), Redis (key-value), Cassandra (wide-column), Neo4j (graph).

FOR ONLINE READING ONLY

Use cases

(i) Big data applications and real-time analytics.

(ii) Content management systems and e-commerce platforms with evolving data models.

(iii) Social networks, gaming applications, and IoT systems need scalable, high-performance data storage solutions.

NoSQL databases are a diverse group of database management systems that are designed to handle various types of *unstructured, semi-structured, and structured* data.

Four main classifications of NoSQL databases

(a) Document stores: These databases store and retrieve semi-structured data in JavaScript Object Notation (JSON) or Binary Javascript Object Notation (BSON) format. Each document can contain a different number of fields, and the structure can vary from one document to another. Example of document stores include Mongo database (MongoDB).

(b) Key-value stores: In these databases, data is stored in a schema-less way, where each item in the database is stored as an attribute-value pair. They are highly scalable and are often used for caching and session management. Examples include Redis.

(c) Column-family stores: These are also known as wide-column stores, these databases organise data into columns rather than rows. Each column family contains rows with a unique row key, and each row consists of multiple columns. They are suitable for storing and querying vast amounts of data with high availability. Example of column-family stores database includes HBase.

(d) Graph databases: These databases are optimised for storing and querying data with complex relationships. They represent data as graphs consisting of nodes, edges, and properties. Example of graph database include Neo4j.

Creating a NoSQL database



Case study: *To create a MongoDB as a NoSQL database*

To create a MongoDB database, you first set up a MongoDB server. Then, you use a MongoDB client or driver to work with it. Use the following step-by-step guideline on how to create a MongoDB database

Step1: Install MongoDB:

Visit the MongoDB download center (https://www.mongodb.com/try/download/community) to download and install MongoDB Community

FOR ONLINE READING ONLY

Edition for your operating system. Follow the installation instructions for your platform.

Step 2: Start MongoDB server

After installation, start the MongoDB server. On most systems, you can start the MongoDB server by running 'mongod' in your command prompt.

Step 3: Connect to MongoDB

MongoDB provides a command-line interface called 'mongo' or mongosh to interact with the database. Open a new command prompt window and type 'mongo' to start the MongoDB shell. This will connect to the default MongoDB server running on 'localhost' port '27017' as shown in Figure 8.26.



Figure 8.26: *Connecting to default Mongo server*

Step 4: Create database

In the MongoDB shell, you can create a new database using the 'use' command followed by the name of the database you want to create. Example in Figure 8.27 shows how to create a database called "studentRecord", in NoSQL this is referred to as collection.

```
test> show dbs
admin   40.00 KiB
config  60.00 KiB
local   72.00 KiB
test> use studentRecords
```

Figure 8.27: *Creating database 'studentRecord'*

Step 5: Insert document into collection

You can create a collection within the database by inserting a document into it. MongoDB creates collections automatically when you insert data into database. Consider examples in (a) and (b) below:

(a) Insert a single document into a collection: A query example that insert a single document into a collection is as presented in Figure 8.28.



Figure 8.28: *Inserting a document in MongoDB*

(b) Insert multiple documents into a collection: A query example that inserts multiple documents into a collection is presented as shown in Figure 8.29:



Figure 8.29: *Inserting multiple documents in MongoDB*

(c) Find document from a collection: To verify that the documents you entered are there in the collection run the command "db.collection_name.find ()". The collection name can be replaced by the name of the database/collection you are referring to as shown in Figure 8.30.



Figure 8.30: *Finding a document from a collection in MongoDB*

FOR ONLINE READING ONLY

Common commands used in MongoDB are summarized in Table 8.18:

Command	Function
>db.collection_name.drop()	drop the collection
>db.dropDatabase()	drop the current Database
>db	show the current Database name
> show dbs	show all the list of databases
>show collections	show the collection name
>db.collection_name.find()	find the documents in the collection
> use databasename	create the new DB
>db.createCollection(name,options)	create new collection

Comparison of SQL against NoSQL

(i) **Data Structure and Scalability:** If the data is highly structured and integrity is paramount, an SQL database is preferred. For unstructured data with a need for horizontal scaling, NoSQL is more suitable.

(ii) **Development Speed and Flexibility:** NoSQL databases offer more flexibility and can speed up development by allowing changes to data models without impacting existing data.

(iii) **Query Complexity:** SQL databases are better suited for complex queries requiring joins and transactions.

(iv) **Consistency Requirements:** SQL databases offer strong consistency (ACID), whereas NoSQL databases might offer eventual consistency, making them more apt for distributed systems. Understanding these differences, strengths, and limitations is crucial in choosing the appropriate database system for specific needs, ensuring the efficient, reliable, and scalable management of data.

Exercise 8.9

1. Compare and contrast the benefits of using a wide-column store versus a graph database in different application scenarios.
2. Discuss the use cases where a document-oriented database like MongoDB would be more suitable than a relational database system.
3. What are the advantages of using NoSQL databases for handling unstructured data compared to traditional relational databases?
4. Set up a MongoDB database using MongoDB Atlas Free Tier. Then, perform the following:
   (a) Create a new MongoDB cluster on MongoDB Atlas.
   (b) Define a collection named "books" with fields like title, author, and genre.
   (c) Insert sample documents into the "books" collection.

FOR SCHOOL READING ONLY

(d) Write queries to retrieve books based on author and genre.

5. Implement a simple key-value store using Redis by performing the following:
   (a) Install and set up Redis on your local machine or use a cloud-based Redis service.
   (b) Define keys for storing data related to user sessions (such as session IDs and user data).
   (c) Implement operations to set, get, and delete key-value pairs.
   (d) Test the key-value store by storing and retrieving sample data.
6. Set up a Cassandra cluster and perform basic operations by implementing the following:
   (a) Install Cassandra on a local machine or use a cloud-based Cassandra service.
   (b) Create a keyspace named "library" and define a table "books" with columns for book ID, title, author, and genre.
   (c) Insert sample data into the "books" table.
   (d) Write CQL queries to retrieve books based on book ID and author.

Chapter Summary

Reflect on what you have learnt from this chapter, summarise and keep it in your portfolio.

Revision Exercise 8

1. How might the use of databases in academic records management align with broader trends in digital transformation and information technology?
2. What is a Structured Query Language? Briefly explain how it is used to interact with relational databases.
3. How does a NoSQL database differ from a relational database in terms of data structure and flexibility?
4. In what scenario might a NoSQL database be a better choice compared to a traditional relational database?
5. What is a data warehouse, and how does it differ from a traditional database in terms of its purpose and data organisation?
6. Explain the challenges of big data associated with collecting, managing, and analysing massive datasets.
7. Provide an example of how big data and data warehouses could be used in a specific industry (such as, healthcare, finance, education, local governments).
8. Identify any correlation between where the user is browsing from and the specific actions they perform on the website.
9. Analyse the content of user messages left on different website

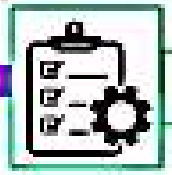
FOR ONLINE READING ONLY

pages, to identify any recurring themes or concerns expressed by users of specific services provided by the website.

(a) What features would you find in an object-oriented database model that are not typically present in a relational database model?

(b) How do data structures like trees relate to database models? Provide an example of a database model that uses this structure.

10. Construct an SQL query to update the salary of all employees in the "Sales" department by 10%.
11. How can you respond to someone who need to understand about parallel and distributed database systems?
12. In parallel database systems, why is data partitioning considered an important technique?
13. What is the primary reason for using tables (comprising rows and columns) in a relational database management system (RDBMS) for data storage?
14. Why are indexes used in a database management system (DBMS)?
15. In the context of a database management system (DBMS), what is the goal of query optimization?
16. Consider a national healthcare system in Tanzania that stores patient records, medical histories, and treatment information. Which system between parallel and distributed database systems, would be most appropriate if the goal is to ensure reliable access to complete patient records at any hospital across the country, and why?
17. Describe demands created by the rise of IoT (Internet of Things) devices
18. What is the purpose of the echo statement in PHP?
19. What makes GET and POST HTTP methods not to look the same?
20. Demonstrate with code how to implement a CRUD operation for a 'users' table in PHP using PDO.



Project: *Multi-functional Library Management System*

Project Overview:

Develop a comprehensive Library Management System (LMS) that integrates various functionalities relevant to the topics covered throughout the course. The LMS will allow users to manage library resources, borrow and return books, search for books, and provide administrative functionalities.

FOR ONLINE READING ONLY

Project Components:

1. Computer architecture and processor optimization:
Implement efficient data structures and algorithms for managing large datasets of books and user information.
Optimise data retrieval and storage operations to minimize processing time using techniques such as indexing and caching.

2. Object-Oriented Programming (C++, Java, Python):
Choose one programming language (C++, Java, or Python) to implement the core functionalities of the LMS.
Use object-oriented principles to design classes such as Book, User, Library, Transaction to model the entities and interactions within the system.

3. Computer networks:
Implement network communication aspects such as client-server architecture for remote access to the LMS.
Design protocols for data exchange between the client (library users) and server (library database).

4. Operating systems:
Consider concurrency and synchronization issues when implementing features like book borrowing and returning to ensure data consistency.
Utilize file handling and memory management concepts for efficient data storage and retrieval.

5. Databases and Database Management Systems:
Design a relational database schema using SQLite, MySQL, or PostgreSQL to store information about books, users, transactions, etc.
Implement CRUD (Create, Read, Update, Delete) operations to interact with the database from the LMS application.

Project Deliverables:

1. Functional LMS application: A fully functional application with a user-friendly interface for library users (borrowers) and administrators (librarians).
2. Documentation: Detailed documentation including system design, database schema, class diagrams, user manuals, and technical specifications.
3. Presentation: A final presentation demonstrating the LMS functionalities, architecture, and optimisation strategies used.

PROJECT WORK

(End of Advanced Level Secondary Education Project):

Select an area of your interest related to Computer Science, then design and conduct a project.

FOR IN-HOUSE READING ONLY

Glossary

Term	Description
MongoDB Shell	An interactive JavaScript interface to MongoDB, used to query and interact with the database.
BSON	Binary JSON, a binary-encoded serialization of JSON-like documents, used by MongoDB to store data.
Aggregation	A way of processing data in a collection by passing it through a pipeline of operations. *Example:* Grouping students by major and counting the number of students in each major: db.students.aggregate([{$group: {"_id": "$major", "count": {$sum: 1}}}]);
eNodeB	The eNodeB, or Evolved Node B, is the hardware responsible for establishing the connection between mobile devices and the network in LTE (Long Term Evolution) and LTE-Advanced networks, which are part of the 4G wireless communication standard. Acting as a crucial component within the LTE network architecture, the eNodeB undertakes essential functions, including managing radio resources, compressing IP headers, encrypting data, and handling mobility management.
IS-95	Standard for CDMA Networks: IS-95, also known as cdmaOne, is a 2G mobile telecommunications standard that utilizes Code Division Multiple Access (CDMA) technology. Developed by Qualcomm and standardized in the early 1990s, IS-95 enables multiple users to share the same frequency spectrum simultaneously through unique spreading codes. It supports features like soft handoff, power control, and operates in the 800 MHz and 1900 MHz frequency bands with a 1.25 MHz channel bandwidth. The standard laid the foundation for advanced CDMA technologies, leading to higher network capacity, improved call quality, and efficient spectrum utilization, ultimately evolving into CDMA2000 for 3G networks.
S1-AP (S1 Application Protocol)	S1-AP (S1 Application Protocol): S1-AP is a protocol used in LTE (Long-Term Evolution) networks to manage the signaling between the evolved NodeB (eNodeB) and the Mobility

FOR ONLINE READING ONLY

Management Entity (MME). It facilitates critical functions such as establishing, maintaining, and releasing connections, handling mobility management, bearer management, and the delivery of NAS (Non-Access Stratum) signaling messages. S1-AP ensures efficient communication and coordination between the radio access network and the core network, playing a crucial role in maintaining smooth connectivity and high-performance service delivery in LTE networks.

X2-AP (X2 Application Protocol)

X2-AP is a protocol used in LTE (Long-Term Evolution) networks to manage communication and coordination between neighboring eNodeBs (evolved NodeBs). This protocol is essential for handling inter-cell handovers, ensuring that data sessions are smoothly transferred from one cell to another as users move. It also manages the load balancing of traffic and the sharing of status information between eNodeBs. X2-AP supports functions like user plane traffic forwarding, context transfer, and measurement reporting, which are vital for maintaining network performance, optimizing resource utilization, and providing a high-quality user experience in LTE networks.

GTP (GPRS Tunneling Protocol)

GTP is a group of IP-based communication protocols used in the GPRS (General Packet Radio Service) core network to facilitate the transmission of data between different network nodes. It consists of three versions: GTP-C (Control Plane), GTP-U (User Plane), and GTP' (Charging Data Collection). GTP-C is used for managing sessions, such as setting up and tearing down connections. GTP-U handles the actual user data transfer, ensuring that data packets are efficiently routed between the user's device and the Internet. GTP' is used for the transfer of billing and charging information. Together, these protocols enable reliable and efficient data transfer within 2G, 3G, and 4G mobile networks, supporting various data services and applications.

NG-RAN (Next Generation Radio Access Network)

NG-RAN (Next Generation Radio Access Network) is a key component of 5G networks, designed to improve performance, efficiency, and flexibility. It consists of gNodeBs (gNBs) that provide connectivity between user devices and the 5G core network. NG-RAN supports higher data rates, lower latency,

FOR ONLINE READING ONLY

	and massive connectivity, enabling advanced applications like IoT, autonomous vehicles, and smart cities. It utilizes advanced technologies such as beamforming and massive MIMO to enhance coverage and capacity, and is designed to work with various spectrum bands, including both sub-6 GHz and mmWave frequencies.
5GC (5G Core)	The 5G Core (5GC) is the central component of 5G networks, providing the backbone for enhanced connectivity and advanced services. It is designed to support the diverse requirements of 5G, such as higher data speeds, ultra-low latency, and massive IoT connectivity. The 5GC architecture is service-based, modular, and flexible, allowing for easy integration and scalability. It incorporates key functions such as the Access and Mobility Management Function (AMF), Session Management Function (SMF), and User Plane Function (UPF) to manage network access, session handling, and data transport. The 5GC enables network slicing, which allows operators to create customized virtual networks tailored to specific applications and services, thereby enhancing efficiency and performance.
MAP (Mobile Application Part)	The Mobile Application Part (MAP) is a protocol used in GSM, UMTS, and GPRS networks to enable communication between network elements and provide various services to users. MAP plays a crucial role in call setup, call routing, and Short Message Service (SMS) in telecommunication networks. It allows access to important network elements such as the Home Location Register (HLR), Visitor Location Register (VLR), Mobile Switching Center (MSC), and more. The specifications for MAP are defined by organisations like the GSM Association and ETSI/3GPP. It is an integral part of the core network infrastructure, facilitating seamless communication and service provisioning in mobile networks.
BSSAP (Base Station Subsystem Application Part)	The Base Station Subsystem Application Part (BSSAP) is a protocol used in cellular networks to facilitate communication between the Base Station Subsystem (BSS) and the Mobile Switching Center (MSC). It is a part of the Signaling System 7 (SS7) protocol suite, which manages signaling messages in the Public Switched Telephone Network (PSTN). BSSAP consists of several layers, including the BSSMAP layer, which handles

FOR ONLINE READING ONLY

BSS-related functions like radio channel establishment and release, handover management, and paging. The DTAP layer is responsible for exchanging signaling messages between the MSC and Mobile Station (MS), while the SCCP layer provides connection-oriented and connectionless services. BSSAP enables the transfer of Mobility Management and Session Management information, allowing the allocation of traffic channels, management of radio resources, and coordination of voice and data communication in cellular networks.

SS7 (Signaling System No. 7)

Signaling System No. 7 (SS7) is an international telecommunication protocol standard that facilitates the exchange of information and control signals among network elements in a Public Switched Telephone Network (PSTN). It comprises a set of telephony signaling protocols used for call setup and teardown in most parts of the global PSTN. SS7 enables advanced calling features like call routing, billing, and Short Message Service (SMS). It also plays a crucial role in services such as number translation, local number portability, and prepaid billing. As a critical component of modern telecommunications systems, SS7 provides the necessary signaling and control for various network services and capabilities.

FOR ONLINE LEARNING ONLY

Bibliography

Behrouz A. Forouzan (2006). *Data communication and Networking, 4th Edition*, Mc Graw-Hill, India.

Brett, M., Gary, P., & David, W. (2006). *Head First Object-Oriented Analysis and Design*.

Brookshear, J. G., Brylow, D., & Manasa, S. (2009). *Computer science: An overview*. Pearson Education Limited

C. Hamacher, Z. Vranesic and S. Zaky, (2011). *Computer Organization, 5th Edition*, McGraw Hill.

Cambridge Assessment International Education (2021). *Cambridge International AS and A Level Computer Science. Computer Science. (Syllabus 9618)*

Cambridge Assessment International Education (2021). *Cambridge International AS and A Level Computer Science. Computer Science. (Syllabus 9618)*

Connolly, T., & Begg, C. (2019). *Database systems: A practical approach to design, implementation, and management* (7th ed.). Pearson.

CRC Press (2016). *Fundamentals of Parallel Multicore Architecture: Multichip and Multicore Systems*. by Yan Solihin Publisher.

D. A. Patterson and J.L. Hennessy (2011). *Computer Architecture: A Quantitative Approach, 5th Edition*. Morgan Kaffman.

D. A. Patterson and J.L. Hennessy (2016). *Computer Organization and Design: The Hardware/Software Interface, 5th Edition*". Elsevier India.

Date, C. J. (2020). *An introduction to database systems* (9th ed.). Pearson.

Elmasri, R., & Navathe, S. B. (2016). *Fundamentals of database systems* (7th ed.). Pearson.

Garcia-Molina, H., Ullman, J. D., & Widom, J. (2021). *Database systems: The complete book* (3rd ed.). Pearson.

J.P. Hayes (1998). *Computer Architecture and Organization, 3rd Edition*. McGraw Hill.

Kaufmann, M., John L. Hennessy and David A. Patterson (2012). *Computer Architecture: A Quantitative Approach*,5th Edition

Kendal, S. (2009). *object-oriented-programming-using-java*. Ventus Publishing Java: A Beginner's Guide

FOR ONLINE READING ONLY

Kroenke, D. M., & Auer, D. J. (2018). *Database concepts* (8th ed.). Pearson.

Kurose, Ross (2010). *Computer Networking: A top down approach*, Pearson Education, India.

MOEVT.(2010). *Computer Science Syllabus for Advanced Level Secondary Education Form V– VI (2nd Ed)*.. Tanzania Institute of Education.

MOE & UCLES (2020). *Singapore–Cambridge General Certificate of Education. Advanced Level Higher 2 (2022) Computing (Syllabus 9569)*. Ministry of Education.

MOE & UCLES (2020). *Singapore–Cambridge General Certificate of Education. Advanced Level Higher 2 (2022) Computing (Syllabus 9569)*. Ministry of Education.

Ramakrishnan, R., & Gehrke, J. (2020). *Database management systems* (4th ed.). McGraw-Hill Education.

Silberschatz, A., Korth, H. F., & Sudarshan, S. (2020). *Database system concepts* (7th ed.). McGraw-Hill Education.

W. Stallings (2015). *Computer Organization and Architecture: Designing for Performance*. Pearson.

FOR ONLINE READING ONLY

APPENDIX A: CODE OF ETHICS AND PROFESSIONAL CONDUCT

The ACM Code of Ethics expresses the conscience of the computing profession, and it affirms an obligation of computing professionals both individually and collectively to use their skills for the benefit of society.

1. GENERAL ETHICAL PRINCIPLES.

A computing professional should...

1.1 Contribute to society and to human well-being, acknowledging that all people are stakeholders in computing.

1.2 Avoid harm.

1.3 Be honest and trustworthy.

1.4 Be fair and take action not to discriminate.

1.5 Respect the work required to produce new ideas, inventions, creative works, and computing artifacts.

1.6 Respect privacy.

1.7 Honor confidentiality.

2. PROFESSIONAL RESPONSIBILITIES.

A computing professional should...

2.1 Strive to achieve high quality in both the processes and products of professional work.

2.2 Maintain high standards of professional competence, conduct, and ethical practice.

2.3 Know and respect existing rules pertaining to professional work.

2.4 Accept and provide appropriate professional review.

2.5 Give comprehensive and thorough evaluations of computer systems and their impacts including analysis of possible risks.

2.6 Perform work only in areas of competence.

2.7 Foster public awareness and understanding of computing, related technologies, and their consequences.

2.8 Access computing and communication resources only when authorised or when compelled by the public good.

2.9 Design and implement systems that are robustly and usably secure.

FOR ONLINE READING ONLY

APPENDIX B: STANDARDS AND STANDARDS-MAKING BODIES

Principal roles in standardisation for computer science and engineering are played by the International Standards Organization (ISO), the American National Standards Institute (ANSI), and the Institute of Electrical and Electronics Engineers (IEEE). These organizations are briefly described below, with pointers to their Web pages provided for further information.

1. The International Organization for Standardization (ISO)

"ISO is a network of the national standards institutes of 147 countries, on the basis of one member per country, with a Central Secretariat in Geneva, Switzerland, that coordinates the system. ISO is a non-governmental organization: its members are not, as is the case in the United Nations system, delegations of national governments. Nevertheless, ISO occupies a special position between the public and private sectors. This is because, on the one hand, many of its member institutes are part of the governmental structure of their countries, or they are mandated by their governments. On the other hand, other members have their roots uniquely in the private sector, having been set up by national partnerships of industry associations."

2. The American National Standards Institute (ANSI)

The American National Standards Institute (ANSI) is a private, non-profit organization (501(c)3) that administers and coordinates the U.S. voluntary standardization and conformity assessment system. The Institute's mission is to enhance both the global competitiveness of U.S. business and the U.S. quality of life by promoting and facilitating voluntary consensus standards and conformity assessment systems, and safeguarding their integrity. (See http://www.ansi.org for more details).

ANSI standards in computer science exist in the areas of architecture, graphics, and programming languages. The International Committee for Information Technology Standards (INCITS) is accredited by ANSI to create and maintain standards in information technology including the various areas of computer science.

3. Institute of Electrical and Electronics Engineers (IEEE).

IEEE is world's largest not-for-profit organisation for electronic engineering and electrical engineering with its corporate office in New York. The mission of IEEE is advancing techology for the benefit of every human kind.

FOR ONLINE READING ONLY

Indices

1G (Analog) 207, 533
2G (GSM) 207, 533
5G Protocols 208, 533

A

Abstract class 313, 360, 405, 533
Abstraction 220, 221, 222, 300, 312, 360, 396, 399, 405, 533
Acknowledgments 161, 169, 533
Addressing 51, 65, 137, 144, 152, 533
Addressing modes 51, 65, 533
Address Resolution Protocol (ARP) 149, 533
Advanced RISC Machine (ARM) 67, 533
Alternate key 446, 533
AND gate 21, 22, 25, 27, 31, 32, 33, 34, 533
Application layer 125, 172, 175, 176, 533
Application of logic gates 29, 533
Application Programming Interfaces 416, 533
Architectures 52, 84, 113, 414, 533, 534
Arithmetic 2, 4, 30, 52, 54, 55, 60, 63, 67, 75, 77, 249, 256, 326, 327, 330, 533
ARMv8 (AArch64) 52, 56, 533
Arrays iv, 283, 533
Assignment 249, 250, 251, 258, 337, 341, 533
Assignment operators 249, 250, 533
Assignments 249, 258, 455, 533
Attribute 294, 444, 451, 454, 533
Average Memory Access Time (AMAT) 40, 87, 533

B

Big data 487, 490, 518, 533
Bitwise AND 332, 333, 341, 533
Bool data types 237, 238, 533
Boolean algebra expression and simplification 533
Boolean operations 12, 21, 533
break 182, 222, 267, 268, 269, 275, 393, 396, 427, 463, 533
Buffering 166, 167, 168, 533

C

Cable Crimpers 186, 533
Cable Termination equipment 187, 533
Cable testing tools 185, 533
Cable Trays 188, 533
Cabling Pathways (Conduit) 188, 533
Cache Hit 39, 533
Cache Line 39, 533
Cache Miss 39, 533
Candidate key 446, 533
CDMA (Code Division Multiple Access) 208, 533
Central Office (CO) 205, 533
Character data types 238, 533
Class 152, 153, 154, 155, 156, 157, 162, 216, 293, 298, 310, 311, 325, 345, 354, 376, 378, 379, 380, 395, 399, 400, 401, 533
Classes in Java 310, 533
Classification of computer architecture 533
Coaxial cable 178, 181, 533
Combinational logic circuit 31, 533
COMMIT 475, 476, 533
Common Cabling Tools 186, 533
Communication satellite 183, 533
Composite Key 446, 533
Compound assignment operators 249, 251, 252, 533
computer architecture iii, 1, 2, 7, 8, 9, 10, 30, 36, 51, 70, 71, 74, 80, 81, 533
Computer memory iii, 35, 36, 533
Computer Memory Hierarchy 38, 533
Computer Memory Interfacing 38, 533
Computer Memory Technologies 40, 533
Concept of Boolean algebra 10, 533
Conceptual data model 442, 533
Conditional Operator 339, 533
Congestion Avoidance 167, 168, 533
Congestion control 150, 151, 533
Connectionless-oriented transport 171, 533
Connection-oriented transport 164, 533
Constants 244, 341, 342, 533
Construction of full adder 34, 533
Construction of half adder 33, 533
Construction of half and full adder 33, 533
Construction of truth tables 15, 533
Constructor overloading 351, 533
Constructors iv, 348, 533
Continuity testers 185, 533
Control structures 260, 533
CPU scheduling 421, 427, 428, 431, 432, 533
CPU simulator 56, 57, 58, 61, 85, 87, 422, 423, 533
Crossover Cable Construction 190, 533

D

Data allocation in memory 5, 533
Database Management Systems (DBMS) v, 466, 477, 533
Database models v, 433, 437, 533
Database normalisation forms 458, 533
Database system 434, 530, 533
Data de-encapsulation 133, 533
Data Encapsulation 131, 533

FOR ONLINE READING ONLY

Datagram Service 145, 533
data Level parallelism iii, 82, 109, 533
Data Level Parallelism iii, 116, 533
Data link layer 136, 533
data packets 142, 144, 145, 146, 147, 148, 151, 171, 199, 526, 533
Data Partitioning 117, 119, 533
Datapath and Control 62, 533
Data types 237, 238, 322, 392, 444, 533
Data Warehouse 489, 533
Deadlock 425, 426, 534
Decoding the Instruction 63, 534
decrement operators 249, 252, 253, 534
DeMorgan's theorem 18, 19, 20, 25, 534
Denial-of-Service Attacks 199, 534
Derived data types 239, 534
Designing Local Area Network iv, 191, 534
Destructors in Java 352, 534
Device Drivers 415, 534
Direct Mapping 44, 534
Distributed Database Systems 482, 483, 534
DMA Technology 73, 534
DMA Transfers 72, 534
DNS Hacking Attacks 199, 534
Domain vi, 119, 174, 199, 445, 534
DRAM 40, 42, 534

E

Echo Statement 501, 534
Emerging database models 484, 534
Encapsulation 131, 133, 134, 213, 214, 298, 313, 361, 369, 376, 378, 399, 403, 533, 534
Entity Relationship 439, 440, 442, 453, 534
Entity-relationship model 451, 534
evolution 206, 412, 534
Exceptions 74, 534
Execute 59, 63, 84, 92, 99, 122, 232, 477, 479, 495, 496, 534
Expressions 256, 257, 258, 391, 534

F

Fetching of Instruction 63, 534
Fibre Optical Cables (FOC) 182, 534
file-based system 434, 436, 437, 534
Final 341, 342, 367, 368, 534
Finalize() method 353, 534
Firewalls 200, 534
First Come First Served (FCFS) 421, 427, 534
First Normal Form (1NF) 458, 459, 460, 534
Fish Tapes 187, 534
Flash Memory 40, 534
Flat file Database model 440, 534
Float data types 238, 534
Flow control 164, 172, 534
Flynn's taxonomy architecture 9, 534
Foreign Key 446, 447, 534
forwarding 90, 92, 93, 94, 95, 101, 102, 122, 138, 140, 195, 526, 534
Fully Associative Mapping 45, 534

G

Getter 369, 370, 384, 403, 534
Getter and setter methods 370, 534
GRANT 475, 534
Graph databases 484, 487, 534
GSM (Global System for Mobile Communications) 534
Guided transmission media 178, 179, 534

H

Harvard architecture 2, 7, 8, 534
Hazards 89, 90, 534
Header format 161, 162, 534
Hello fellow Africans 388, 534
Hierarchical database model 437, 534
Hit Rate 39, 534
Hold and wait 426, 534

I

ICMP 149, 150, 200, 534
IDEs 302, 377, 387, 388, 534
IDLE 388, 389, 398, 534
increment operators 252, 534
Infrared waves 183, 184, 534
Inheritance 215, 216, 218, 295, 314, 354, 357, 371, 377, 378, 399, 404, 534
Input and output streams 247, 534
Instruction execution cycles 62, 534
Instruction Level Parallelism (ILP) iii, 82, 95, 96, 97, 101, 102, 534
Instruction Set Architectures 52, 84, 534
Integer data types 237, 534
Intel x86 52, 53, 534
Interface 24, 84, 140, 198, 226, 313, 355, 357, 358, 415, 432, 529, 534
Internet Control Message Protocol 149, 534
Internet Protocol Version 6 (IPV6), 534
Interrupts 74, 534
IPv4 Subnetting 158, 534
Iteration 105, 106, 107, 269, 534

J

Java Classes 345, 534
java comments 321, 534
Java Interfaces 357, 534

K

kernel 414, 415, 418, 534

L

logical AND 67, 257, 336, 534
Logical Data model 442, 534
logical OR 257, 336, 534
Logic circuits iii, 30, 534
Logic errors 235, 534

FOR ONLINE READING

Logic expressions and laws 12, 534
Logic gates iii, 10, 21, 29, 534
Logisim simulator 24, 27, 29, 80, 534
Looping 269, 392, 534
LTE (Long Term Evolution) 208, 525, 534

M

Magnetic Disk 40, 41, 534
Memory mapping 43, 534
Method overloading, 534
Method overriding 218, 373, 402, 534
microkernel 414, 415, 534
Microwave transmission, 534
Modified Harvard architecture 8, 534
Modules in Python 397, 534
MongoDB 482, 488, 517, 518, 519, 520, 521, 525, 534
monolithic kernel 414, 534
Movement of data 2, 534
movement of instruction 85, 87, 534
Multidimensional array 284, 285, 534
Multiple inheritance 357, 358, 534
Multithreading 109, 111, 112, 114, 534
Mutual exclusion 426, 534
MySQL 434, 444, 447, 449, 451, 457, 464, 465, 466, 467, 476, 480, 493, 494, 495, 496, 498, 499, 500, 504, 509, 510, 511, 513, 515, 516, 517, 524, 535

N

NAND gate 22, 23, 535
Network database model 438, 535
Network layer 125, 131, 132, 133, 137, 142, 144, 145, 535
Network Masks 156, 535
Network models iv, 131, 535
Network prefix 157, 535
Network security principles iv, 124, 196, 535
NewSQL databases 484, 535
Non-relational (NoSQL) databases 487, 488, 535
Non-trivial functional dependency 455, 535
Non Von Neumann architecture 2, 8, 9, 535
No preemption 426, 535
NOR gate 23, 535
Normalisation 457, 535
NoSQL database 488, 518, 522, 535
NOT gate 11, 21, 22, 27, 32, 535
NOT gate (Inverter) 22, 535

O

Object iv, vi, viii, 211, 212, 213, 217, 219, 220, 221, 223, 224, 293, 308, 309, 310, 314, 321, 325, 331, 346, 363, 365, 385, 439, 518, 524, 529, 535
object-oriented database 439, 466, 523, 535
Object Oriented Programming iv, vi, viii, 211, 212, 217, 223, 308, 309, 314, 321, 385, 535
OOP iv, v, vi, 211, 212, 213, 214, 215, 217, 218, 219, 220, 221, 222, 223, 224, 292, 299, 308, 309, 312, 313, 314, 315, 343, 348, 354, 361, 384, 385, 386, 399, 402, 406, 535
Operating Systems v, viii, 410, 412, 413, 432, 524, 535
Operators iv, 31, 249, 256, 258, 326, 330, 331, 332, 333, 336, 337, 391, 535
OR gate 21, 23, 27, 31, 32, 35, 535
OS 76, 111, 113, 126, 410, 411, 412, 413, 414, 415, 416, 417, 418, 419, 420, 423, 424, 425, 428, 429, 430, 431, 432, 535
OSI model 125, 126, 128, 129, 130, 131, 134, 136, 137, 138, 142, 143, 173, 175, 177, 535

P

Packet-Mistreatment Attacks 199, 535
Packet switching 535
Parallel database systems 482, 535
Patch cords 188, 535
Patch Panel 187, 535
PHP Syntax 501, 535
Physical Data Model 442, 535
Physical layer 125, 132, 133, 134, 135, 197, 535
pipeline stages 85, 86, 87, 88, 92, 102, 535
Pipelining iii, 69, 82, 83, 84, 87, 89, 97, 110, 535
Polymorphism 217, 218, 220, 296, 297, 314, 371, 399, 402, 535
Primary and secondary memory 48, 535
Primary key 445, 535
Primitive types 322, 535
Principle of locality 36, 535
print 79, 115, 237, 266, 284, 294, 299, 312, 320, 321, 346, 348, 349, 366, 372, 379, 380, 381, 382, 388, 390, 392, 393, 395, 396, 397, 399, 401, 402, 403, 404, 406, 408, 535
Privileged Instructions 70, 76, 535
Process Management 418, 535
projection 444, 447, 448, 449, 450, 451, 535
PSTN 203, 204, 205, 206, 207, 208, 209, 527, 528, 535
Public Switched Telephone Networks iv, 203, 535
Punch-Down Tools 186, 535
Python iv, v, viii, 219, 223, 224, 376, 377, 380, 381, 385, 386, 387, 388, 389, 390, 391, 392, 393, 394, 396, 397, 398, 399, 400, 402, 403, 404, 405, 406, 407, 408, 409, 494, 524, 534, 535
Python Functions v, 396, 535
Python fundamentals v, 389, 535
Python Objects 400, 535

FOR ONLINE READING ONLY

Q

Query Optimization 479, 535

R

Raceways 188, 535
Radio waves 183, 184, 535
Read Operands 63, 535
Redundancy 132, 434, 535
Register renaming 108, 535
Registers 37, 38, 54, 55, 59, 61, 63, 67, 76, 535
Register Transfer Language 51, 66, 535
Relation 445, 535
Relational database 438, 441, 444, 450, 535
Relational database model 438, 441, 535
Relational operators 249, 253, 330, 332, 535
REVOKE 475, 535
RISC-V 52, 55, 535
ROLLBACK, 475, 535
Round Robin (RR) 421, 428, 535
Routing 144, 145, 148, 157, 199, 205, 207, 535
routing algorithm 145, 146, 148, 152, 535
Run-time error 235, 236, 535

S

SAVEPOINT 475, 476, 535
scalar pipelining 102, 535
Second Normal Form (2NF) 458, 459, 462, 535
Security 77, 145, 161, 175, 196, 197, 200, 202, 203, 208, 317, 413, 416, 420, 450, 494, 535
Selection 260, 447, 448, 449, 535
Semantic errors 234, 536
Semaphores 426, 535
Sequencing 164, 170, 535
Sequential logic circuit 31, 535
Serverless Databases 485, 535
Set-Associative Mapping 45, 535
Shielded twisted pair cable 180, 536
Shortest Job First (SJF) 421, 425, 427, 428, 536
Simulation of Von Neumann architecture 2, 5, 536
Split Cache 41, 536
SQL commands 468, 472, 476, 536
SQL Databases 493, 536
SRAM 40, 68, 536
Stalling 92, 94, 536
Standard logic gates symbols 21, 536
Static 40, 97, 103, 105, 157, 158, 243, 311, 325, 326, 342, 536
STP 179, 180, 186, 195, 536
Straight -Through cable construction ix, 536
String 53, 287, 288, 310, 318, 320, 324, 325, 326, 327, 328, 330, 331, 332, 334, 335, 339, 342, 344, 346, 347, 349, 350, 351, 352, 353, 355, 358, 359, 361, 363, 364, 365, 366, 367, 368, 369, 370, 372 - 375, 384, 536
Structured Query Language 466, 467, 486, 492, 522, 536
Structure of C++ program 224, 536
Subnetting 158, 159, 160, 161, 534, 536
Superscalar pipelining 102, 103, 536
Switched LANs 137, 536
Switching Centers 205, 207, 536

T

TCP/IP iv, vi, 126, 128, 129, 130, 131, 145, 152, 153, 155, 172, 173, 175, 209, 210, 536
TCP/IP model iv, 128, 129, 130, 131, 173, 209, 536
Ternary operator 338, 536
The principle of cost-performance ratio 37, 536
Third Normal Form (3NF) 442, 458, 459, 462, 464, 536
Thunny 388, 394, 395, 536
Thread iii, 82, 109, 110, 111, 112, 113, 115, 117, 119, 121, 123, 536
throughput 72, 73, 83, 87, 88, 89, 98, 168, 199, 536
Time-Series Databases 484, 536
Tone generators and amplifier probes 185, 536
Translation Lookaside Buffer (TLB) 50, 536
Transmission media iv, 124, 178, 536
Transmission Media 205, 536
Transport Layer 163, 171, 175, 202, 208, 536
Trivial functional dependency 455, 536
Truth tables iii, 15, 536
Twisted pair cable 179, 536
Types of a logic circuit 30, 536

U

Unguided or wireless transmission media 536
Unified Cache 41, 536
Un-Normalised Form (UNF) 459, 536
Unshielded twisted pair cable 179, 536
User-defined data type 239, 536
UTP 179, 180, 186, 189, 190, 536

V

Variable declaration 242, 305, 323, 536
Variables iv, 19, 241, 242, 243, 311, 323, 390, 501, 536
Virtual Circuit Service 145, 536
Virtual Memory 50, 536
Virtual Private Networks 201, 536
VLANs (Virtual Local Area Networks) 138, 536
Von Neumann architecture 2, 5, 7, 8, 9, 535, 536
VPN 200, 201, 536

W

Wall plate 187, 536
Windowing 166, 168, 536
Wire Cutters 186, 536
Wire-map testers 185, 536
Wire Strippers 186, 536
Write Back 64, 84, 122, 536

FOR ONLINE EDUCATION ONLY

