According to the new syllabus of Computer Science of
Board of Intermediate Education, Karachi, Hyderabad, Sukkur and Larkana

WEB INTEGRATED EDITION

(www.computerstudies12.wordpress.com)

COMPUTER STUDIES

with C++

(Theory and Practical)

By: Sir Ahad

for

Class XI
(General Science Group)

By

Gul Shahzad Sarwar

Ph.D.
Department of Physics & Computer Science
Government National College
Karachi

IQRA PUBLISHERS

Urdu Bazar, Karachi

CamScanner

Contents

	page
Preface	xv
CHAPTER 1: OVERVIEW OF COMPUTER SYSTEM & MEMORY	1
Computer System	1
Looking Inside the Machine	2
Input Hardware	3
Keyboard Input Devices	3
Keyboards	3
Terminals	4
Non-Keyboard Input Devices	5
Pointing Devices	5
Scanning Devices	8
Image-Capture Devices	9
Voice-Input Devices	10
Processing and Memory Hardware	11
Central Processing Unit (CPU)	11
Main Memory	11
Output Hardware	11
Softcopy Output Hardware	12
Monitors	12
Cathode-Ray Tube Monitors	12
Flat Panel Monitors	14
Voices and Sounds	14
Speech-Generating Devices (SGDs)	16
Sound-Output Devices	16
Multimedia Projectors	17
Hardcopy Output Hardware	17
Printers	18
Impact Printers	18
Non-impact Printers	18
Plotters	19
Storage Hardware	22
Types of Storage Devices	23
Magnetic Storage—Hard Disks	24
	24

By: Sir Ahad

CamScanner

Characteristics of a Hard Disk	24
Construction of a Hard Disk	25
Optical storage—CD-ROM, DVD-ROM, Blu-ray Disks	27
CD-ROM Disks	29
CD-R Disks	29
CD-RW Disks	29
DVD-ROM Disks	30
DVD-R Disks	30
DVD-RW, DVD-RAM, DVD+RW Disks	30
Blu-Ray Disks	30
Flash Storage—Solid State Drive, Memory Card, USB Drive	31
Solid State Drives	31
Memory Card	32
USB Flash Drive	32
Cloud Storage—OneDrive, Google Drive, iCloud	33
Software	35
The Operating System	36
Device Drivers	36
Language Translators	37
Utility Software	38
Introduction to Artificial Intelligence	41
How does AI work?	42
Importance of Artificial Intelligence	42
Advantages and Disadvantages of Artificial Intelligence	43
Applications of AI Technology	44
Cloud Computing	46
Types of Cloud Computing Services	46
Cloud Computing Deployment Models	47
Characteristics and Advantages of Cloud Computing	48
Disadvantages of Cloud Computing	49
Questions	51
Objectives and MCQs	53
CHAPTER 2: THE SYSTEM UNIT AND COMPUTER MEMORY	**57**
The Power Supply	57
The Motherboard	58
The Microprocessor	59
Main Components of a Microprocessor	60
Features of a Microprocessor	61
External Architecture of a Microprocessor	64
Graphics processing unit (GPU)	65
Main Memory or Random Access Memory (RAM)	66
Differences between RAM and Cache	68
How Data and Programs are Represented in the Computer	68
Binary Coding Schemes	70
Read-Only Memory (ROM)	70
Types of ROM	71

CamScanner

Advantages of ROM	72
Expansion Cards and Slots	72
Ports	73
Cooling System	76
Questions	78
Objectives and MCQs	78
CHAPTER 3: OPERATING SYSTEM	81
What Is a User Interface?	82
Command-line Interface	83
Graphical User Interface	83
Examples of Desktop Operating System	83
MS-DOS and PC-DOS	83
Unix	84
Linux	85
macOS	85
Microsoft Windows	86
Features of an Operating System	87
Batch Processing	87
Examples of Batch Processing System	88
Multitasking	88
Types of Multitasking	89
Time-Sharing	89
Features of Time-Sharing OS	90
Multiprocessing	90
Parallel Processing	91
Distributed System	92
Embedded Operating System	93
Embedded versus non-embedded OSes: What is the Difference?	94
Functions of the Operating System	94
Process Management	96
States of Process in Operating Systems	97
What is Thread?	98
Multithreading vs. Multitasking	100
What is Multitasking?	100
What is Multithreading?	101
What is Windows?	103
Start menu	103
The Desktop	104
The Taskbar	104
Widgets	105
File Explorer	105
Settings	108
DirectX Diagnostic Tool (Dxdiag.exe)	109
System Configuration Utility (msconfig)	109
Introduction to Open-Source Operating System	110
How Open-Source Operating System Works?	110

CamScanner

Types of Open-Source Operating System	110
Android OS—An example of the most popular open-source OS	111
History and Development	112
Android OS Features	112
Android OS Versions	112
Hardware	113
Comparisons with other mobile OSes	113
Questions	114
Objectives and MCQs	115
CHAPTER 4: ALGORITHM AND FLOWCHARTS	**117**
Algorithm	117
Pseudocode	119
Mathematical Notations and Functions	119
Algorithmic Notations	120
Flowcharts	121
Categories of Flowcharts	124
Control Structures	125
Sequence	125
Selection	126
Iteration	127
Types of Flowcharts	130
Flowcharting Rules	131
Advantages of Flowcharts	137
Disadvantages of Flowcharts	138
Exercise	139
Objectives and MCQs	143
CHAPTER 5: AN OVERVIEW OF C++ LANGUAGE	**145**
Introduction to C++	145
Advantages of C++	146
The IDE of Dev-C++ Compiler	147
Features of Dev-C++ IDE	147
Using Dev-C++	148
The Basic Structure of a C++ Program	150
Using Comments	152
Escape Sequences	153
Good Programming Practice	155
Exercise	156
Objectives and MCQs	157
CHAPTER 6: C++ FUNDAMENTALS	**159**
The C++ Character Set	159
Identifiers and Keywords	159
C++ Data Types	161
Choosing the Data Type	166
Constants	166

CamScanner

Symbolic Constants	170
Variables	173
Declaration of Variables	173
Arrays	175
Exercise	177
Objectives and MCQs	179
CHAPTER 7: OPERATORS AND EXPRESSIONS	**180**
Expressions	180
Classifications of Operators	181
Arithmetic Operators	181
Order of Precedence	183
Assignment Operators	185
Relational Operators	188
Confusing Equality (==) and Assignment (=) Operators	190
Logical/Boolean Operators	191
Increment and Decrement Operators	194
Conversions of Data Type	196
Casts	197
Library Functions	198
Exercise	200
Objectives and MCQs	202
CHAPTER 8: INPUT AND OUTPUT STATEMENTS	**204**
Standard Output (cout)	205
Standard Input (cin)	207
Single Character Input	209
The getchar() Function	209
Standard Input (cin) and Strings	211
Stringstream	212
C++ Formatting Output	214
The setw() manipulator	214
The setprecision() manipulator	216
Exercise	218
Objectives	219
CHAPTER 9: SELECTION CONTROL STRUCTURES	**220**
The if Selection Structure	221
Multiple Statements	223
Nested if Statements	224
Logical Operators	225
The if-else Selection Structure	229
Nested if-else Statements	231
The switch Statement	237
Exercise	244
Objectives and MCQs	247

CamScanner

CHAPTER 10: ITERATION CONTROL STRUCTURES 250
The for Loop 250
Multiple Statements 253
Nested for Loops 254
The for Loop Variations 257
The while Loop 261
The Relationship of the for and while Loops 264
Nested while Loops 266
The do-while Loop 267
Nested do-while Loops 269
Combining Loop Types 270
The break Statement 271
The continue Statement 273
Exercise 276
Objectives and MCQs 281

CHAPTER 11: FUNCTIONS 284
Advantages of Functions 284
Classification of Functions 285
General Form of Function 285
Functions with or without Arguments 287
Passing Arguments to a Function 288
Passing Multiple Arguments to a Function 289
Function that Return a Value 291
Using More Than One Function in a Program 295
Recursive Function 300
Function Overloading 302
Exercise 306
Objectives and MCQs 307

CHAPTER 12: ARRAYS 309
One-Dimensional Arrays 309
Declaration of more than One Array 310
Array Initialization 311
Entering Values into an Array 311
Finding Length of an Array 312
Manipulating Arrays 314
Examining an Array 316
Sorting an Array 320
Searching an Array 323
Passing an Array to a Function 326
Multi-Dimensional Array 328
Exercise 332
Objectives and MCQs 334

CamScanner

CHAPTER 13: STRINGS 337
C-Style Strings 337
C++-Style Strings 339
Built-in String Functions 339
Array of Strings 345
Exercise 348
Objectives and MCQs 349

CHAPTER 14: STRUCTURES 351
Defining a Structure 351
A simple Structure 353
Initializing Structure 356
Array of Structures 358
Nested Structure 361
Structures and Functions 363
Exercise 366
Objectives and MCQs 367

CHAPTER 15: DATA COMMUNICATION AND NETWORKS 368
Direction of Transmission 369
Simplex Transmission 369
Half-Duplex Transmission 369
Full-Duplex Transmission 369
Data Transmission Medium 370
Guided Media 370
Twisted-Pair Cable 370
Coaxial Cable 371
Fiber Optic Cable 371
Unguided Media 373
Radio Waves 373
Microwaves 374
Infrared (IR) 376
Various Types of Networks 377
LAN (Local Area Network) 377
Topology of LAN 378
Star Topology 378
Ring Topology 378
Bus Topology 379
Mesh Topology 380
Tree Topology 381
Hybrid Topology 381
WAN (Wide Area Network) 382
Typical Networking Devices 383
Network Interface Cards (NICs) 383
Modem 384
Hubs 385
Bridges 385

CamScanner

Switches 385
Routers 386
Gateways 387
Network Communications Standards 387
Network Architectures 388
The OSI Model 388
TCP/IP Model 390
Internet Model 391
Questions 393
Objectives and MCQs 394

APPENDIX A: Syllabus of Computer Science for Class XI 397

BIBLIOGRAPHY 406

CamScanner

List of Practicals

page

Practical 1: Writing a program to input a 4 (four) digit number and display each digit in separate line. 213

Practical 2: Writing a program of simple marksheet by using selection structures. 234

Practical 3: Writing a program to input a number and find out whether it is a prime or composite. 236

Practical 4: Writing a program of simple calculator using arithmetic operators. 240

Practical 5: Writing a program to calculate factors of a given number. 260

Practical 6: Writing a function that returns larger number from 2 arguments. 293

Practical 7: Writing a function to draw stars of a given size and shape. 298

Practical 8: Writing a program to calculate average of all array members. 313

Practical 9: Writing a program to sort list of items in an array. 321

Practical 10: Writing a program to search a number from inputted list of items. 323

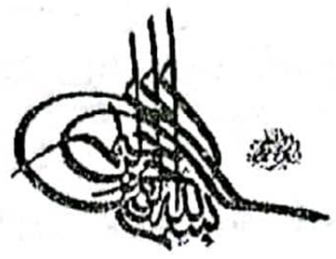
Practical 11: Writing a program to find addition of two matrices. 329

Practical 12: Writing a program to display your inputted name in reverse order using `strlen()`. 344

Practical 13: Writing a program to create structure to display employee data with structure members as name, designation and salary. 356

CamScanner

Preface



Thanks to God Almighty, Who enabled me to present the book "Computer Studies—Theory and Practical".

Although few professions remain untouched by computers today or will remain so in future, but it is crucial in today's world to be able to use and understand personal computers. Most schools, colleges and universities acknowledge that one of their responsibilities is to teach students about computers. There are millions of computer systems in the world today, and collectively, they are capable of doing thousands of different tasks. Some of the tasks that computer systems can now handle, such as creating virtual worlds and beating a world chess champion at his own game, were thought impossible not too long ago. No matter who you are or what you do for a living, it is highly likely that computers somehow impact both the way you work and your success at your work.

The importance of computers in virtually every profession brings us to the purpose of this book. *Computer Studies* is intended for anyone who is interested in computers and C++ programming. It is designed to meet the needs of beginners as well as advanced learners who wish to deal with computers and programming. As computers are being used today in every walk of life, this book can be used by students of computer science, information technology, C++ programming, management, engineering, and general science as well. At the intermediate level, where the course of computer science is included as a part of the curriculum, this book can be used as a textbook at first year level. The book can also serve as course material for participants of short-term course conducted by various training organizations. Programmers and other computer professionals will also find the book helpful and informative. In short, this book is for everyone who is either excited about computers or interested in knowing about computers.

The text of the book has been divided into two parts. The first part explains in straightforward terms the importance of learning about computers, computer systems, computer networks and communications among these networks, principles by which computer systems work, and practical applications of computers and related technologies. Apart from this the concept of operating system, and some topics on

CamScanner

Internet has also been included in the book. All the concepts presented in the book are illustrated with suitable examples and illustrations as and when required. This will enable the readers to use the book effectively for self-study.

The second part presents the fundamentals of programming technique and C++ programming using Dev-C++ version 6.3. C++ is one of the most wonderful programming languages. The reason for this is simple: programmers like it. Once a person learns to program in C++, it is very uncommon for him or her to switch to another language. C++ combines delicacy and elegance with raw power and flexibility. It is a structured language that does not confine. C++ also is a language that puts you firmly in charge. C++ was created by a programmer for programmers. It is not an abstract product of a committee, but rather the outcome of programmers looking for a better language.

The goal of the book is to provide students both a solid knowledge of computer basics and a framework for using this knowledge in the workplace effectively. In addition to this, each chapter contains a number of review questions and exercises which are carefully designed to provide an insight into the subject matter.

For the completion of this book, I am indebted to my wife Farhat Shahzad who motivated me to write this book and for her general encouragement and deep interest regarding the work of this book. I would also like to express my gratitude to the publishers—Professor Shaikh Mubarak Ali—for rendering assistance in the production and publishing of this book.

Although every effort has been made to check minutely the proofs and the subject matter, still there may be some errors and omissions. I shall be grateful to readers for pointing out any shortcoming in the book so that they can be rectified in subsequent editions. I also look forward to their suggestions about adding new topics so that the book can be used by a wider circle of students.



Gul Shahzad Sarwar

1st September, 2023

CamScanner

CHAPTER 1

OVERVIEW OF COMPUTER SYSTEM & MEMORY

By: Sir Ahad

Man is still the most extraordinary computer of all.

John F. Kennedy (1917–1963), President Speech, 21 May 1963

The term computer is derived from the word "compute" which means to calculate. So, a computer is normally considered to be a calculating device that can perform arithmetic operations at enormous speed.

In fact, the original objective for inventing the computer was to create a fast calculating machine. But more than 80% of the work done by computers today is of non-mathematical or non-numerical nature. Hence, to define a computer merely as calculating device is to ignore over 80% of its work.

More accurately, a computer may be defined as a machine that receives data as input, processes that data according to a prescribed sequence of instructions and produces desired output.

Computer System

A system is a group of integrated parts that have the common purpose of achieving some objective(s). So, the following three characteristics are key to a system:

1. A system has more than one element.
2. All the elements of a system are logically related.
3. All the elements of a system are controlled in such a way that the system goal is achieved.

CamScanner

Since a computer is made up of integrated components (input and output devices, storage, CPU, operating system) that work together to perform the steps called for in the program being executed, it is a system. The input or output units cannot function until they receive signals from the CPU. Similarly, the storage unit or the CPU alone is of no use.

So, a *computer system* is a nominally complete computer that includes the hardware, operating system (main software), and peripheral equipment needed and used for full operation. This term may also refer to a group of computers that are linked and function together, such as a computer network or computer cluster.

Examples of Computer Systems

There are many different types of computer systems that can be used by businesses and organizations but also by consumers. The most common type of computer system is the desktop computer, which is a personal computer that sits on a desk or table and can be used by one person at a time.

A laptop computer is similar to a desktop and is also a computer system, but it has an internal battery pack so it can be taken with users when they travel. In an increasingly mobile world, mobile phones and other mobile devices are also considered types of computer systems.

Looking Inside the Machine

The computer itself—the hardware—has many parts. Fig. 1.1 shows the parts of a typical personal computer system. In addition to the CPU, RAM, ROM, and associated control circuits that reside in the computer itself, other components, called *peripheral devices*, are needed to make a complete computer system. Peripheral equipment includes all input, output, and secondary storage devices. In the case of computers, some of the input, output, CPU, memory and storage devices are built into the same physical unit.

A typical digital computer system has five basic functional elements:

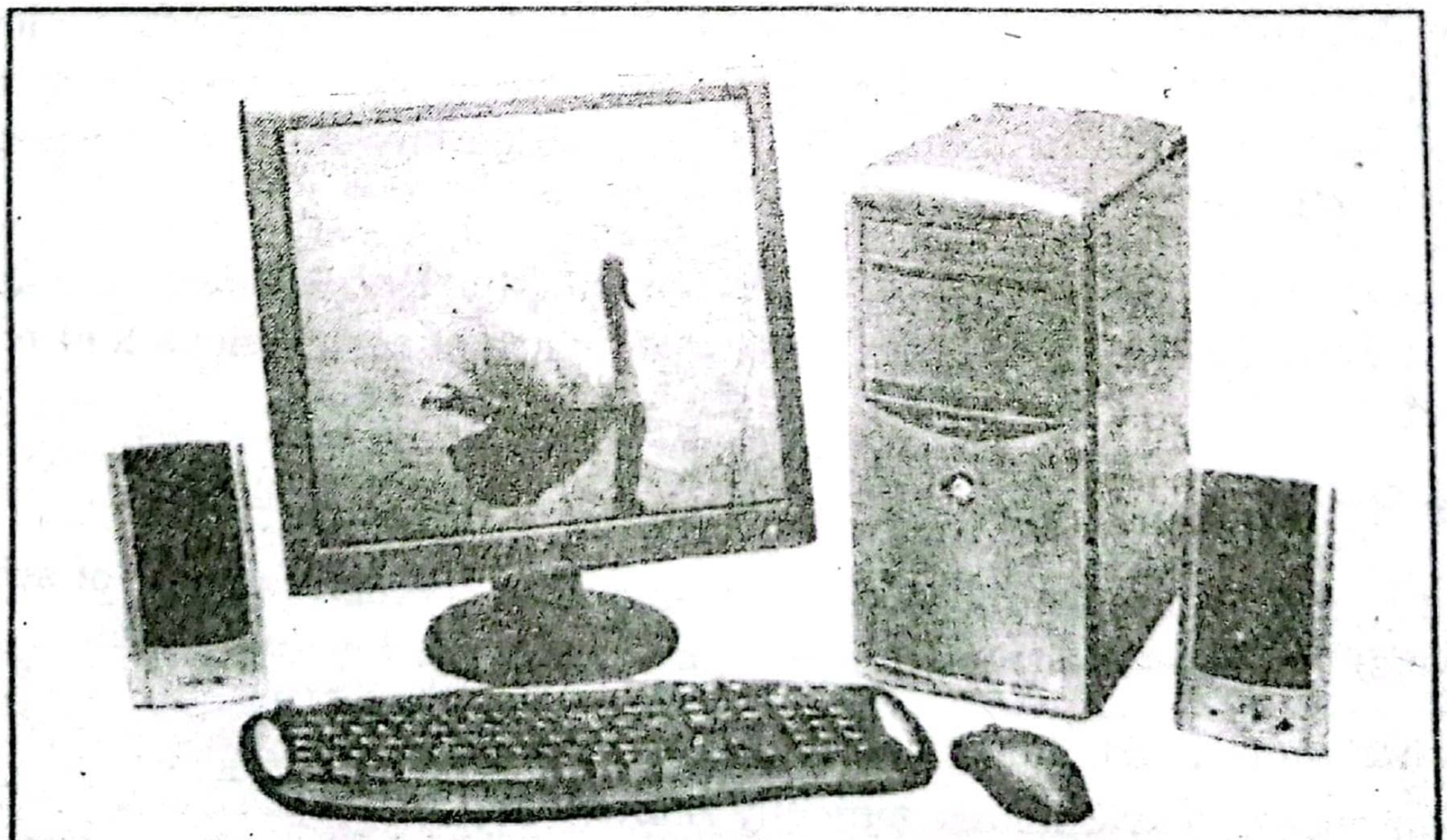


Fig. 1.1. A computer system

CamScanner

1. Input hardware
2. Processing and memory hardware
3. Output hardware
4. Storage hardware; and
5. Software

Let us consider these elements in detail.

Input Hardware

Input and output devices are essentially *converters*. People understand letters, numbers, and punctuation marks. However, computers can understand only the machine language of binary 0s and 1s. Input devices convert what people understand to what computers understand. Output devices do the reverse: they convert machine language to languages people can comprehend.

Input devices consist of devices for entering data into a computer system in a form that can be processed by the computer. There are two kinds of input hardware. They are:

1. Keyboard devices
2. Non-keyboard, or direct-entry, devices

On a personal computer, a keyboard is the primary input device. A keyboard is an input device that converts letters, numbers, and other characters into electrical signals readable by the processor. A computer keyboard looks like a typewriter keyboard, but besides having keys for letters and numbers, it has several keys (such as *Function* keys and *Ctrl*, *Alt*, and *Del* keys) intended for computer-specific tasks.

Keyboard Input Devices

The most common means by which you will input data is through a keyboard.

Keyboards

A keyboard is an input device that looks like a typewriter keyboard but has additional keys for specific purposes. Among personal computers, the most common keyboard layout is shown in Fig. 1.2. There is no single standard dictating the number of keys on a keyboard; most companies use the PC keyboard with a total of 104 alphanumeric keys as a *de facto* standard. The keys arranged in following five groups.

1. Character keys
2. Function keys
3. Special-purpose keys
4. Numeric keys
5. Cursor-movement keys

Character Keys: The character keys are used to type in text (letters and numbers, such as A, B, C and 1, 2, 3), punctuation marks (such as ? and ;), and special characters (such as $, *, and #). These keys are positioned in much the same location as the keys on a typewriter. This layout is called the QWERTY layout because the first six characters on the top row of alphabetic keys are Q, W, E, R, T, Y (see Fig. 1.2).

CamScanner

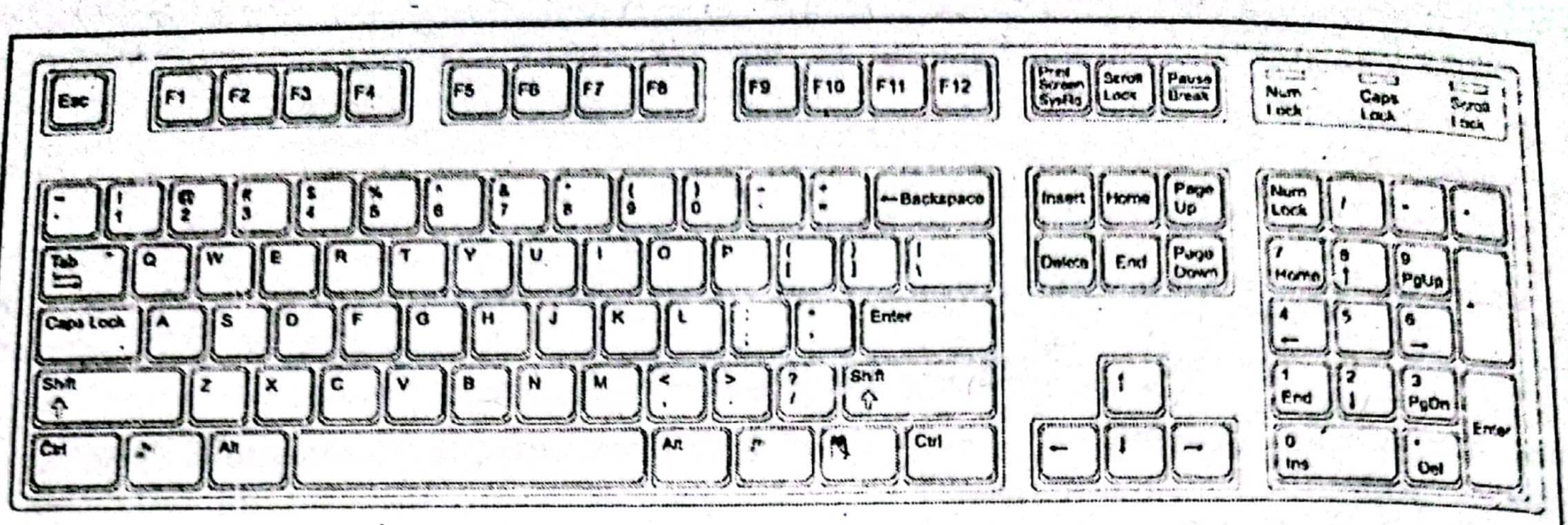


Fig. 1.2. PC keyboard

Function Keys: The function keys are those labelled F1, F2, F3, and so on, and are used to issue frequently used commands (see Fig. 1.2). Today's traditional PC keyboards have 12 function keys.

What the function keys do exactly is determined by the software you are using at the moment. For example, in most software, F1 is commonly the help key. When you press it, a screen displays information about using the program.

Special-Purpose Keys: Special-purpose keys are used to help enter and edit data and execute commands (see Fig. 1.2). These are

1. *Enter—execute commands:* The Enter key, often indicated by a left-facing bent arrow, is used to enter commands into the computer.
2. *Ctrl, Alt, Shift—modifiers:* Three special-purpose keys are *Ctrl* (Control), *Alt* (Alternate), and *Shift*, which are modifier keys. That is, by themselves they can do nothing, but when pressed with another key, they modify the function of the other key.
3. *Del and Ins—editing:* The special-purpose keys *Del* (Delete) and *Ins* (Insert) are used for editing. In word processing, you may use them frequently to delete and insert text.
4. *Caps Lock—uppercase:* The Caps Lock key is used to make all letters keys uppercase—that is, type all CAPITAL LETTERS, as here.

Numeric Keys: The numeric keys, or numeric keypad, which is usually located on the right side of the keyboard, is the part that looks like an adding machine, with 0 through 9 keys and mathematical operators (+, −, *, and /).

Cursor-Movement Keys: The cursor is the blinking symbol on the screen that shows where data may be entered next. The cursor-movement keys, which are represented by directional arrows, move the cursor around the screen.

Terminals

A terminal is an input (and output) device that consists of a keyboard, display screen, and communications connection to a main computer system (see Fig. 1.3).

Terminals are used for inputting data to and retrieving data from a remotely located main computer system. Most terminals are desktop size and are not meant to be carried around. However, some are small enough to be easily portable.

CamScanner

There are two types of terminals:

1. *Dumb Terminals:* A dumb terminal has a display screen and a keyboard and can input and output but cannot process data independently. It is used only to gain access to information from a main computer system. An example might be a terminal used by an airline reservations clerk. For instance, airline reservations clerks use these terminals to access a mainframe computer containing flight information.

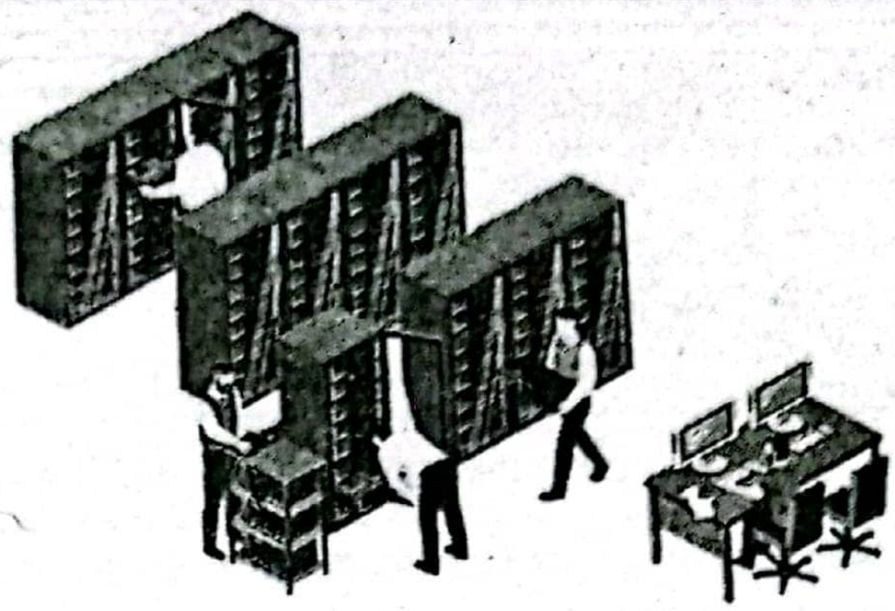


Fig. 1.3. A terminal hooked up to a mainframe system

2. *Intelligent Terminals:* An intelligent terminal has its own processing unit, storage, and software for processing data. It can do independent processing of their own, as well as act as terminals. An intelligent terminal, which is also called a *workstation*, is actually a personal or laptop computer with a communications link to a mainframe. For example, Automated Teller Machine (ATM), the self-service banking machine that is connected through a network to a central computer is an example of intelligent terminal. Another example is the point-of-sale (POS) terminal, used at many stores checkout counters to record purchases, process debit and credit cards, and update inventory.

Non-Keyboard Input Devices

Even though the PC keyboard can be more efficient, it still has several important drawbacks for people using the computer interactively. Many people who don't know how to type feel uncomfortable with a keyboard system. And there are those who don't want to use a keyboard—regardless of its convenient design. Using a keyboard is also slow and often requires that you split your attention between the screen and the keyboard to find special keys.

There are several alternatives to the keyboard that allow a user to point to options or functions on the screen. Direct entry means that data is not entered into the computer through a keyboard. Some common direct-entry devices used to input data into computer-usable form are as follows:

1. Pointing devices: mouse, trackball, joystick, touch screen, and pen-based computer
2. Scanning devices: barcode reader, QR code reader, and image scanner.
3. Image-capture devices: webcam
4. Voice-input devices

Because many such devices cannot input all types of data and instructions, they are often used along with keyboards.

Pointing Devices

The kind of direct-entry input devices known as pointing devices are easy to use—

CamScanner

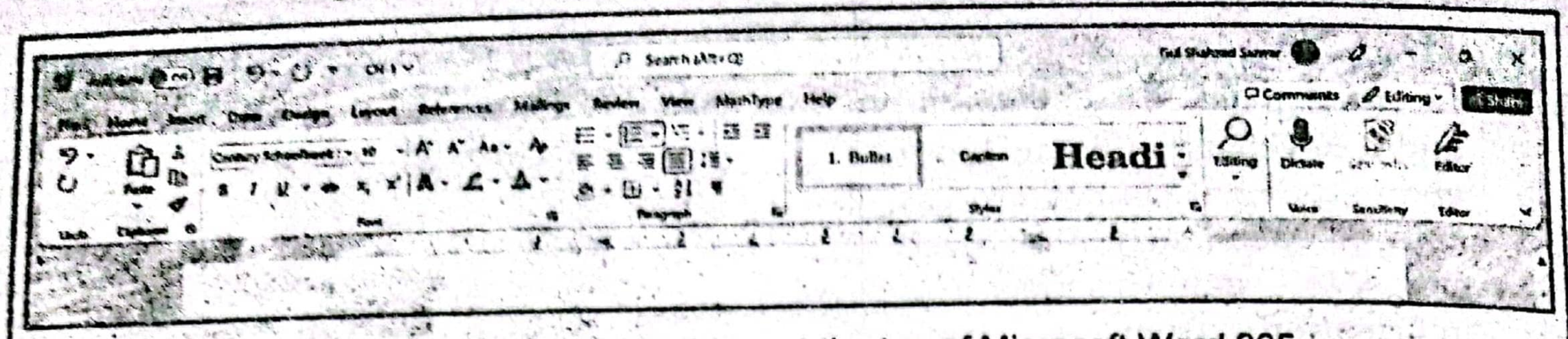


Fig. 1.4. Example of the ribbon at the top of Microsoft Word 365

because there is nothing easier than pointing. Pointing devices are used in menu-driven or ribbon-driven programs—that is, programs that allow the user to select options displayed on the screen (see Fig. 1.4 for ribbon-driven program).

Pointing devices include the following:

Mouse: A mouse is a small, hand-operated device that is connected to the computer wirelessly or by cable and that moves the cursor on the display screen (see Fig. 1.5). As you move the mouse around on the computer, the cursor moves across the screen. When the cursor reaches your desired location, you may push a button on the mouse to signal a command to the computer.

The mouse can also be used like a pen or a paintbrush to draw figures and create patterns directly on the video display screen. A sophisticated mouse enables users to generate very complex and precise images. The keyboard, however, is still used to type in characters and to issue some commands, depending on the software.

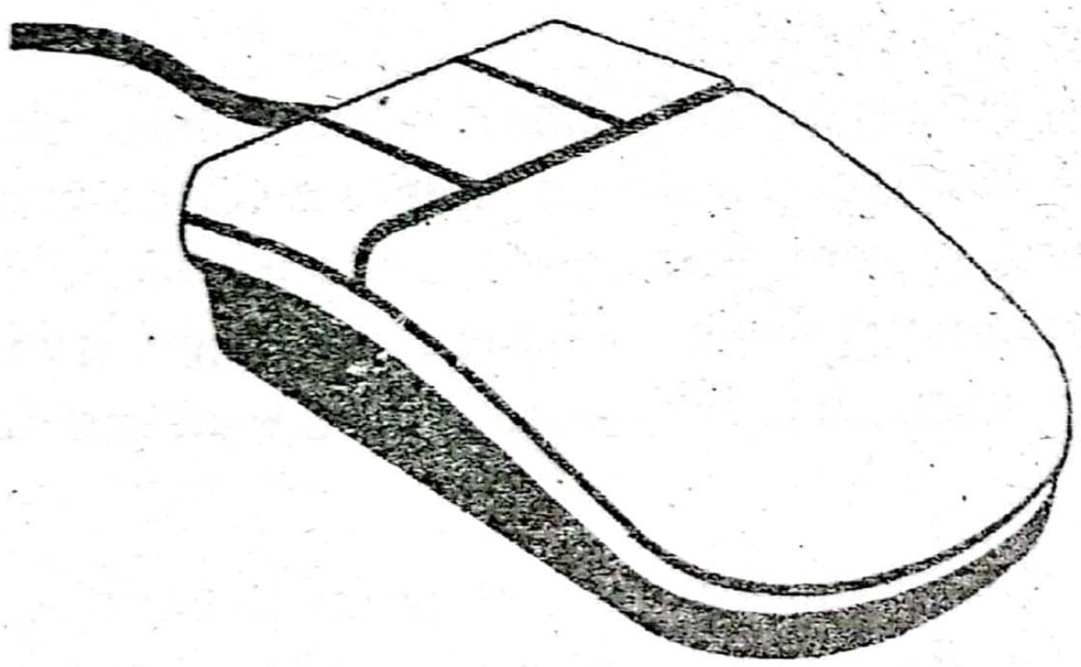


Fig. 1.5. Mouse

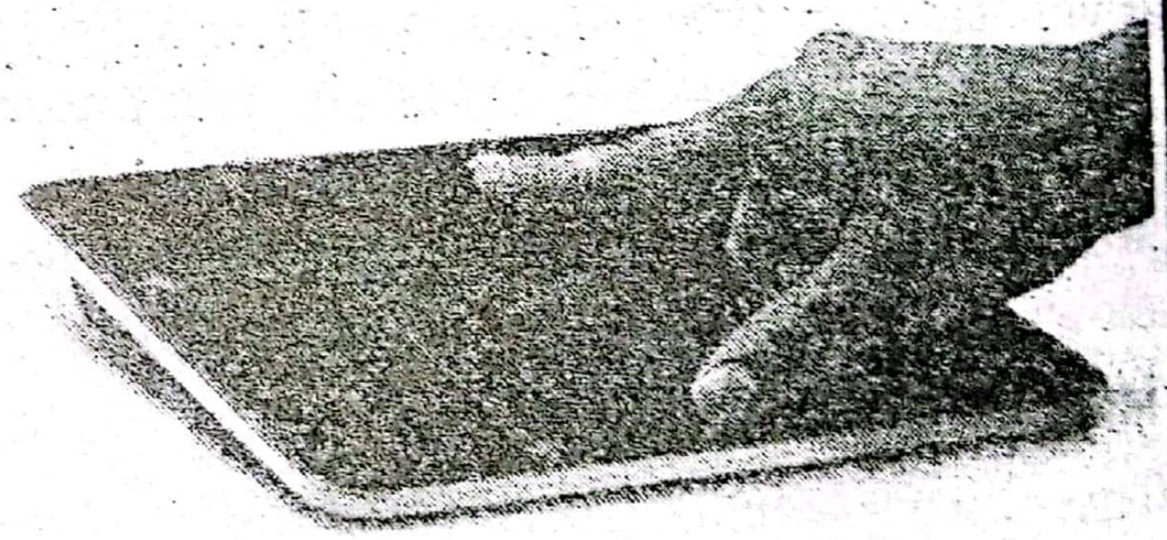


Fig. 1.6. Touchpad or Trackpad

Touchpad or Trackpad: A trackpad is a small, flat surface over which you slide your finger (see Fig. 1.6). The cursor follows the movement of your finger. You "click" by tapping your finger on the pad's surface or by pressing buttons positioned close by the pad. Touchpads are a common feature of laptop computers, but freestanding trackpads are available for use with PCs.

Joystick: A joystick is usually used to play games on computer. It is a device that controls the movement of certain object on the screen (see Fig. 1.7). The buttons on the joystick allow the users to perform different actions depending on the nature of the program.

CamScanner



Fig. 1.7. Joystick

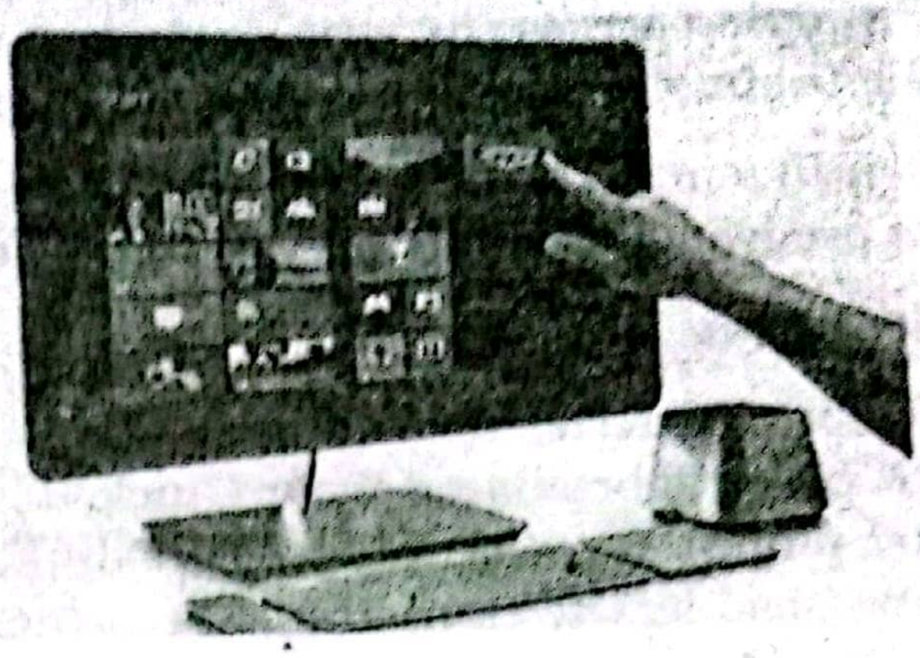


Fig. 1.8. Touch screen

Touch Screen: A touch screen (see Fig. 1.8) is an electronic display screen that is also an input device. A user interacts with the computer, tablet, smartphone or touch-controlled appliance by using hand gestures and fingertip movements to tap pictures, moving elements or type words on the screen. The screens are pressure-sensitive and can be used or manipulated using fingers or a stylus. Today's touch screens are capable of responding to multiple touches at the same time.

Touch screens are a helpful alternative to using a keyboard or mouse while navigating a graphical user interface (GUI). Touchscreens are common in devices such as game consoles, personal computers, electronic voting machines, and point-of-sale (POS) systems. They can also be attached to computers or, as terminals, to networks.

Pen-Based Computers: Pen computing (or stylus) allow users to write on the screen of the computer, which then interprets the handwriting and displays the result on the screen in printed form (see Figs. 1.9-1.10). Users can edit what they have entered and give commands by circling words, checking boxes, and using symbols developed by the manufacturer.

Handwriting recognition refers to the ability of a computer to receive intelligible

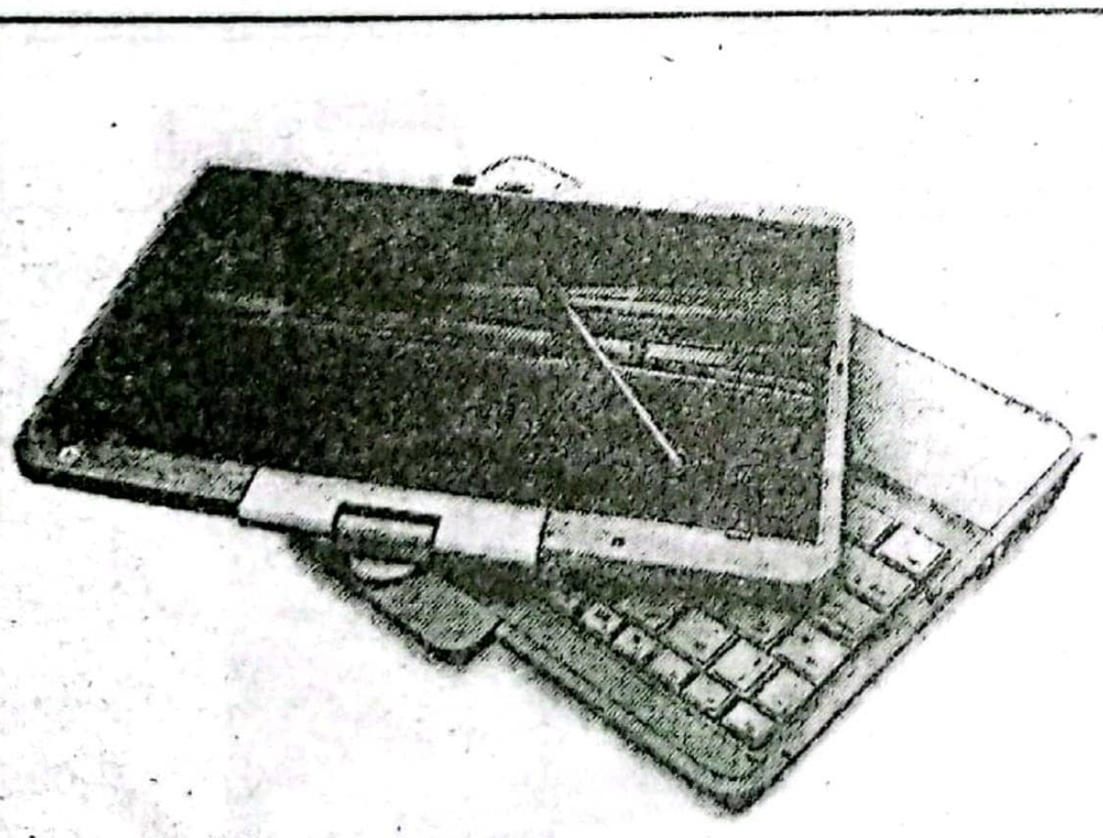


Fig. 1.9. Pen-based computing with and without attached keyboard

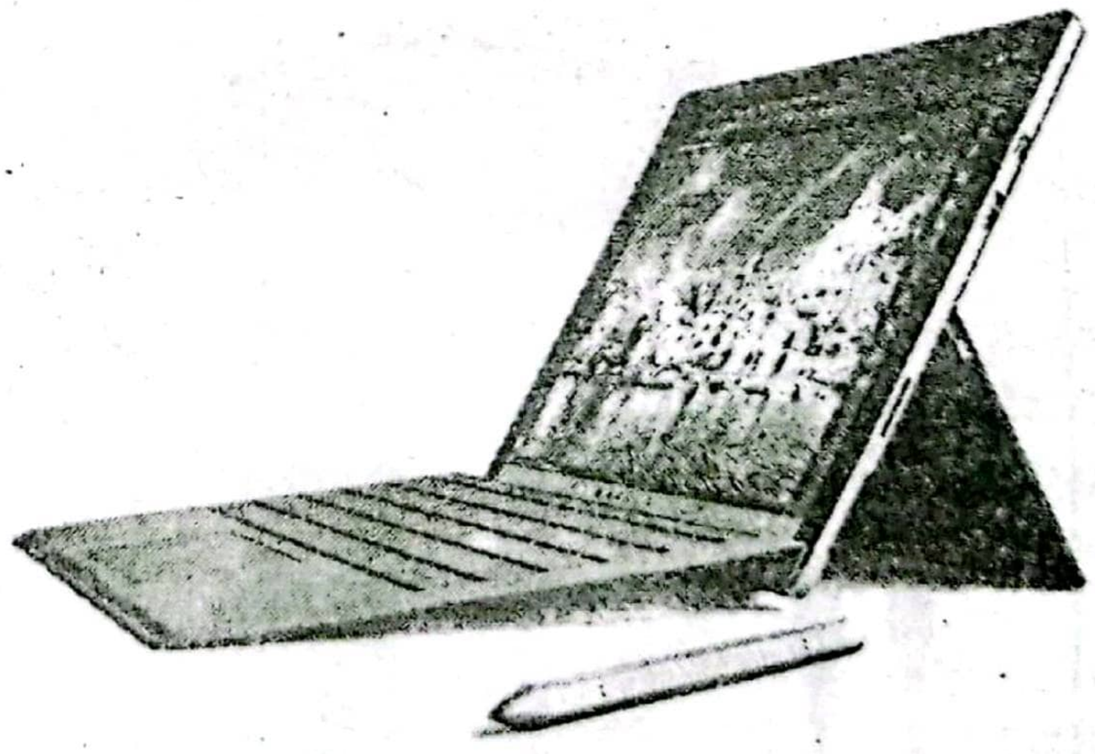


Fig. 1.10. Microsoft's Surface Pro with pen

CamScanner

written input. The system requires special software that interprets the movements of the stylus across the writing surface and translates the resulting cursive writing into digital information.

Pen-like input devices offer increased functionality for computers/laptops/tablets such as programmable buttons, pressure sensitivity and electronic erasers.

Scanning Devices

Scanning devices use light-sensitive equipment to record data in the form of barcodes, QR code (abbreviated from Quick Response Code) or typewritten characters, or even just plain text or graphics. This data is then converted to electronic form that can be processed by the computer. The principal devices include the following:

Barcode Readers: No one nowadays can miss the striped barcodes that appear on food products, cosmetics, and magazine covers (see Fig. 1.11). A barcode is data represented in the parallel lines of a universal coding scheme. An example of such a coding scheme is the Universal Product Code (UPC). The UPC is used by food stores and other retail merchants to represent pricing and other data for various products.

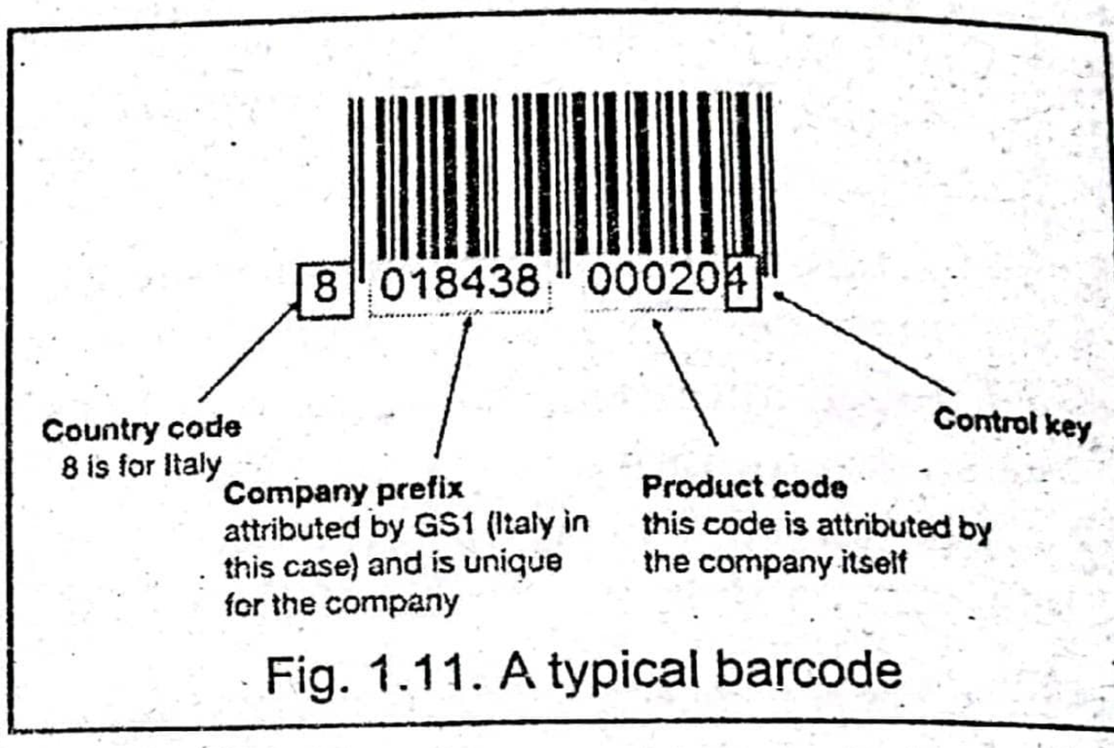


Fig. 1.11. A typical barcode

Barcodes are read by barcode readers. Barcode readers are photo-electric scanners that read barcodes by means of reflected light. Two types of barcode readers are hand-held and stationary:

1. *Hand-held:* Hand-held barcode readers are handgun-like scanning devices that analyze the barcodes and translate them into computer-processable data (see Fig. 1.12).
2. *Stationary:* Stationary barcode readers are built into a countertop, as in the supermarket (see Fig. 1.13).

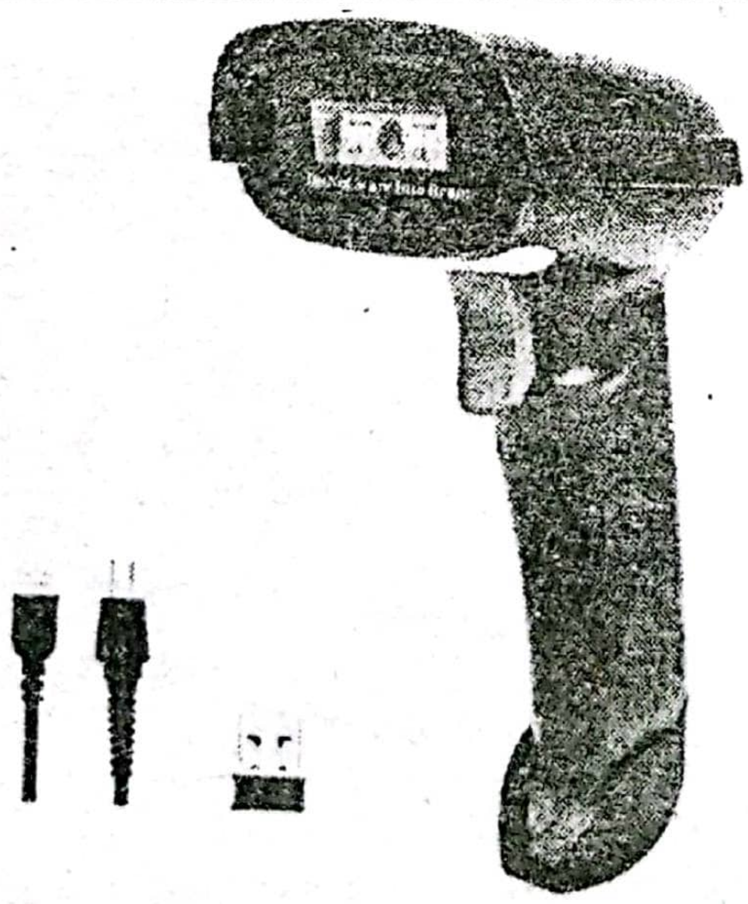


Fig. 1.12. Hand-held barcode reader

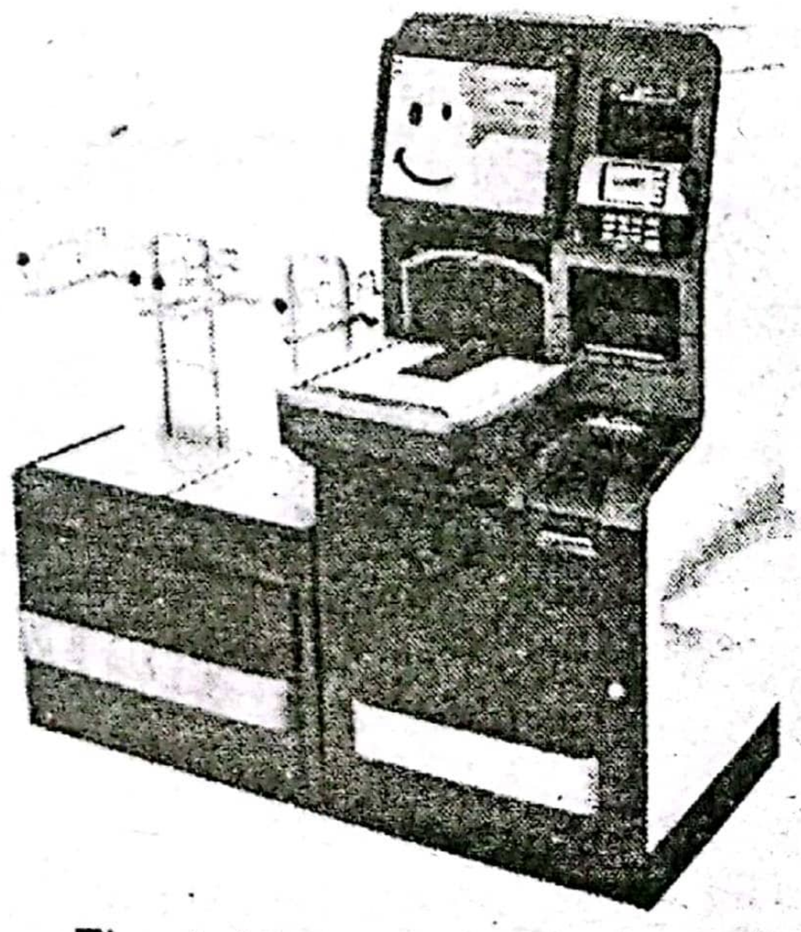


Fig. 1.13. Stationary barcode reader

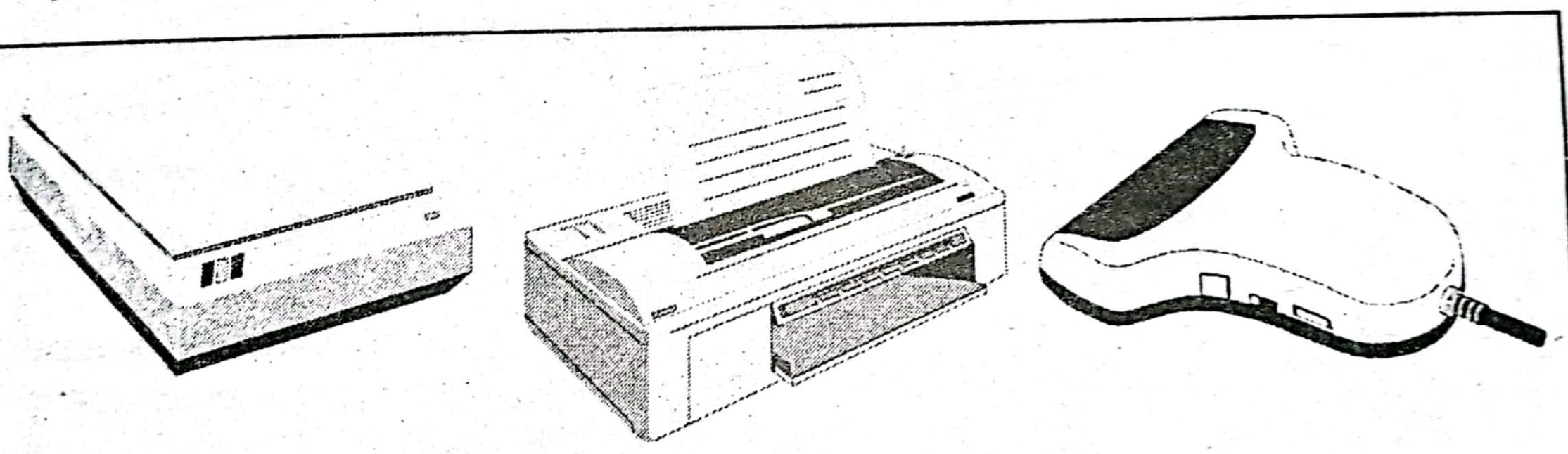
CamScanner

When a supermarket clerk moves your purchase past the countertop barcode scanner, the price and description of the item, which are stored in the store's computer system, are sent to the cashier's terminal (cash register). This price and description are printed out on a receipt for the customer. The information from the terminal is also used by the store for accounting purposes and for restocking the store's products. It is also used for marketing purposes—to analyze which products sell and which don't.

Image Scanners: The scanning devices described above read marks and characters. On the other hand, an *image scanner* is a direct-entry device that scans, or detects, images, drawings as well as text on paper and converts them into computer-usable form. The device scans each image with light and digitizes it. *Digitize* means that the image is converted to computer-processable input data. The images can then be displayed on a monitor, stored on a storage device, printed out, or transmitted to another computer.

Scanners are of several types. They include:

1. *Flatbed:* Flatbed scanners are stationary scanners that hold the image document in place while the scanning mechanism passes over it. A flatbed scanner thus resembles a photocopying machine (Fig. 1.14a).
2. *Sheet-fed:* Sheet-fed scanners are stationary scanners that use mechanical rollers to move the image document past the scanning mechanism (Fig. 1.14b).
3. *Hand-held:* Hand-held scanners are movable scanners that require the scanning mechanism to be moved over the image document (Fig. 1.14c).



(a) Flatbed scanner (b) Sheet-fed scanner (c) Hand-held scanner

Fig. 1.14. Different types of scanners

Image-Capture Devices

Image-capture devices include *webcams* which is designed to record or stream to a computer or computer network. Like scanners, image-capture devices such as webcams have contributed to the rise of the electronic imaging industry, the integration and manipulation of different images.

Webcams are primarily used in videotelephony, livestreaming and social media, and security. Webcams can be built-in computer hardware or peripheral devices, and are commonly connected to a device using USB or wireless protocols.

Nowadays computer hardware manufacturers began building webcams directly into

CamScanner

laptop and desktop screens, thus eliminating the need to use an external USB or wireless webcam. Gradually webcams came to be used more for telecommunications, or videotelephony, between two people, or among several people, than for offering a view on a Web page to an unknown public.

Voice-Input Devices

Voice-input devices, or voice-recognition systems, convert spoken words into computer-usable code (see Fig. 1.15). The devices compare the electrical patterns produced by the speaker's voice with a set of pre-recorded patterns. If a matching pattern is found, the computer accepts this pattern as a part of its standard "vocabulary". Today most such devices have to be specially "trained" to respond to a particular user's voice. In fact, voice input is creating new uses for computers and mobile phones. Due to the many uses of voice recognition software making our lives easier, in just a few years, we brought it into our homes. Today, businesses in a wide array of sectors are tapping into it to make our lives better. Among current uses are:

1. Using virtual assistants like Windows Copilot, Google Assistant, and Apple's Siri or smart speakers at home.
2. Doctors can stop typing while talking to patients.
3. Enhanced security with voice biometry.
4. Voice assistants in the workplace.
5. Using speech recognition to transcribe meetings.
6. E-commerce purchases using voice commands.

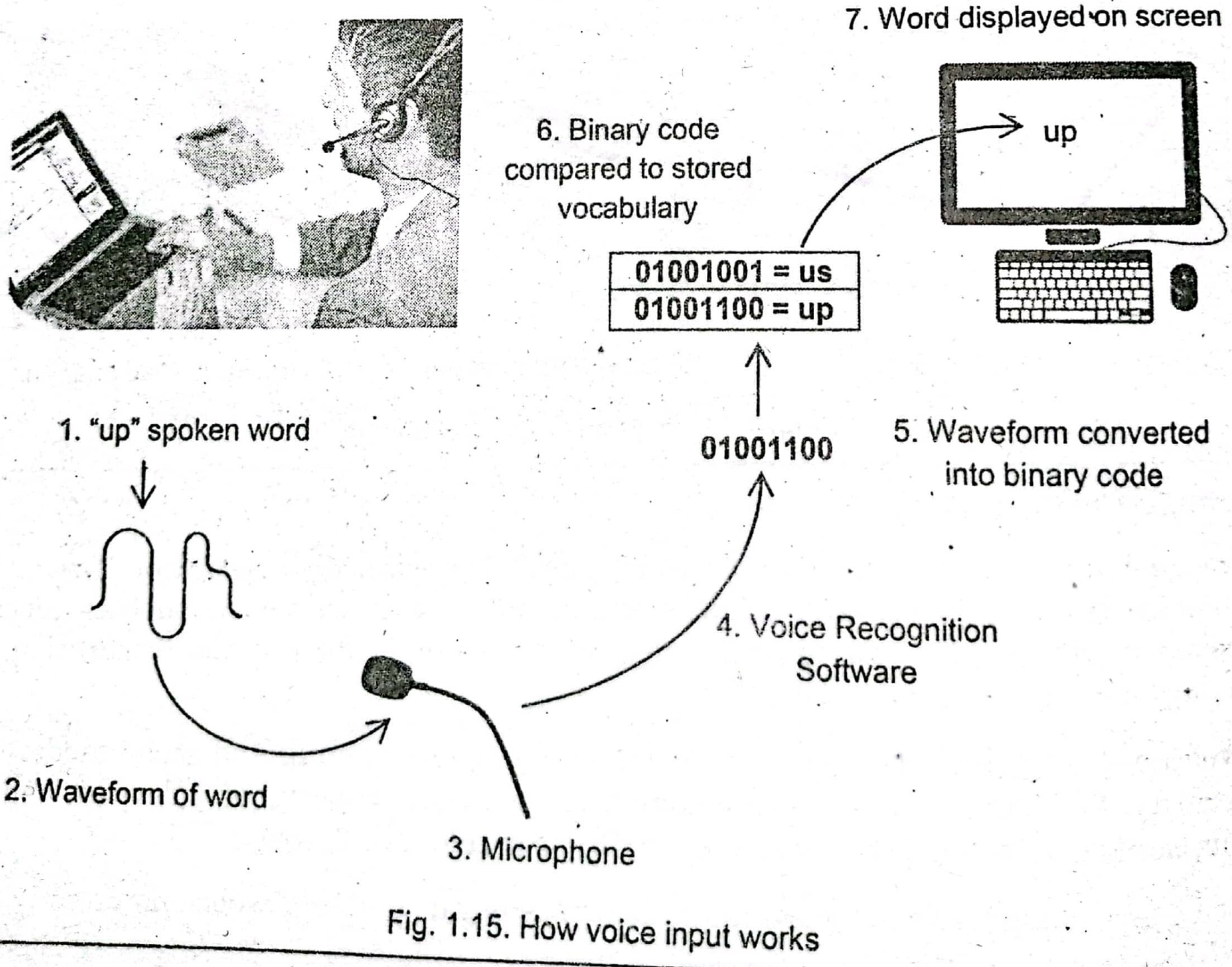


Fig. 1.15. How voice input works

7. Journalists have their interviews transcribed
8. Effortlessly translating and subtitling content

Processing and Memory Hardware

Processing hardware comprises the electronic circuitry inside the cabinet of the computer itself. Two components handle processing in a computer: the central processing unit and the memory. Both are located on the computer's main system board, or motherboard.

Central Processing Unit (CPU)

The central processing unit (CPU) executes program instructions. You can think of it as the computer's brain. In a personal computer, the entire CPU is contained on a tiny chip called a *microprocessor*. The chip is mounted on a piece of plastic with metal leads attached to it. This microprocessor defines type and overall characteristics of a computer. Since 2006, Intel has marketed its chips under "Intel Core" name as Intel Core i3, Intel Core i5, and Intel Core i7 and in 2017 marketed Intel Core i9. It is based on the processor design which was introduced in 1995 with the name Pentium Pro. In general, Core processors are more powerful variants of entry-level Celeron and Pentium processors. Similarly, identical or more capable versions of Core processors are Xeon processors for workstations and low-end servers.

Main Memory

The CPU does contain specialized storage areas called *registers* for data and instructions, but these are small areas that can hold only a few kilo- or mega-bytes at a time. In addition to the registers, the CPU needs to have thousands—or, more often, millions—of bytes of space where it can hold whole programs and the data being manipulated by those programs.

The main memory—also called Random Access Memory (RAM), main storage, primary storage, internal storage or simply memory—holds data and instructions for processing. Although closely associated with the CPU, main memory is technically separate from it. Memory stores program instructions or data for only as long as the program they pertain to is in operation. Keeping these items in memory when the program is not running is not feasible for three reasons:

1. Most types of memory only store items while the computer is turned on—data is destroyed when the machine is turned off.
2. If you share your computer, other people will need the memory space.
3. There may not be room in memory to hold the processed data.

How do data and instructions get from an input device into memory? The control unit sends them. Likewise, when the time is right, the control unit sends these items from memory to the arithmetic/logic unit, where an arithmetic operation or logical operation is performed. After being processed, the information is sent to memory, where it is held until it is ready to be released to an output unit.

The chief characteristic of memory is that it allows very fast access to instructions and data, no matter where the items are within it. We will discuss main memory further in the section that follows.

CamScanner

Output Hardware

Output hardware convert machine-readable information into people-readable form. Output takes two forms:

1. *Softcopy:* Softcopy means the output is in a form that cannot be physically touched. Softcopy output is best for information that needs to be viewed only occasionally or a few moments, such as an airline's departures and arrivals display. The principal softcopy output devices are monitors, projectors and sound or voice output devices.
2. *Hardcopy:* Hardcopy means the output is in a form that can be physically touched. Hardcopy output is best for information that may be used later or that may have greater value over time. The principal hardcopy output devices are printers and plotters.

The output devices we shall describe for computers are monitors, printers, plotters, voice, and sound output devices.

Softcopy Output Hardware

Softcopy output devices are of three principal types:

1. Monitors
2. Speech-Generating devices (SGDs)
3. Sound output devices
4. Multimedia Projectors

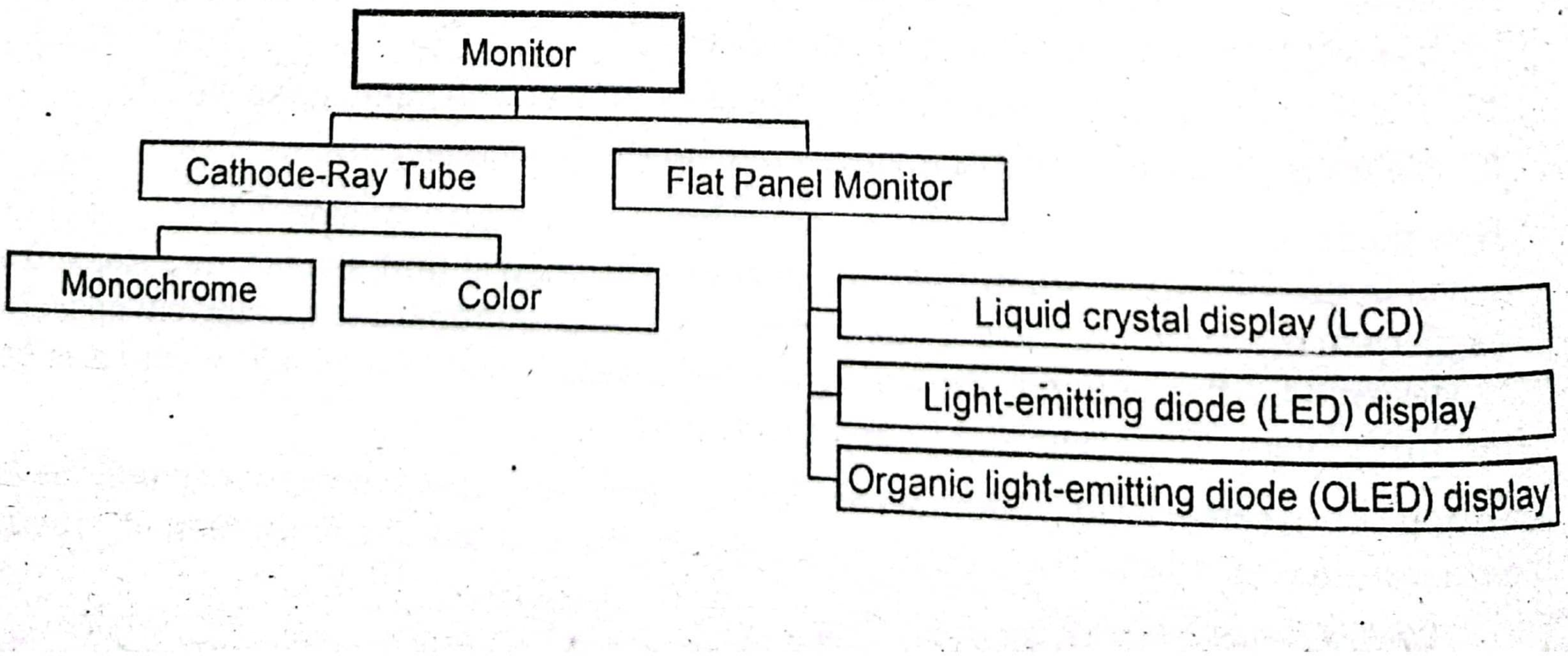
Monitors

This is the most commonly used form of output device when a permanent record is not required. A monitor is a television-like device to display text and graphics from a computer. The monitor is also referred as *screen, console,* and *display*.

There are two main types of monitors:

1. Cathode-ray tube (CRT) monitor.
2. Flat panel monitor.

The following chart shows these types and some common varieties of monitors included in these categories.



CamScanner

Some aspects to consider in buying a monitor are:

1. Resolution—image clarity.
2. Aspect Ratio—proportional relationship between width and height.
3. Refresh Rate—number of times per second the pixels are refreshed.
4. Monochrome/Color—type of color.
5. Color Depth— number of bits used for one pixel.

Resolution: Resolution refers to image clarity. Resolution is determined by the number of pixels in the screen. Pixel is the smallest unit on the screen that can be turned on or off. Each can also be assigned a color. The smaller the pixels and the closer together they are, the more points of an image can be illuminated on the screen, and the better the resolution.

A screen resolution of 1920×1080 means the screen has 1,920 horizontal and 1080 vertical rows. This gives the screen 2,073,600 points, or pixels each forming images (see Fig. 1.16). Examples of higher resolutions are 1920×1440, 2560×1920, and 3000×2000 pixels.

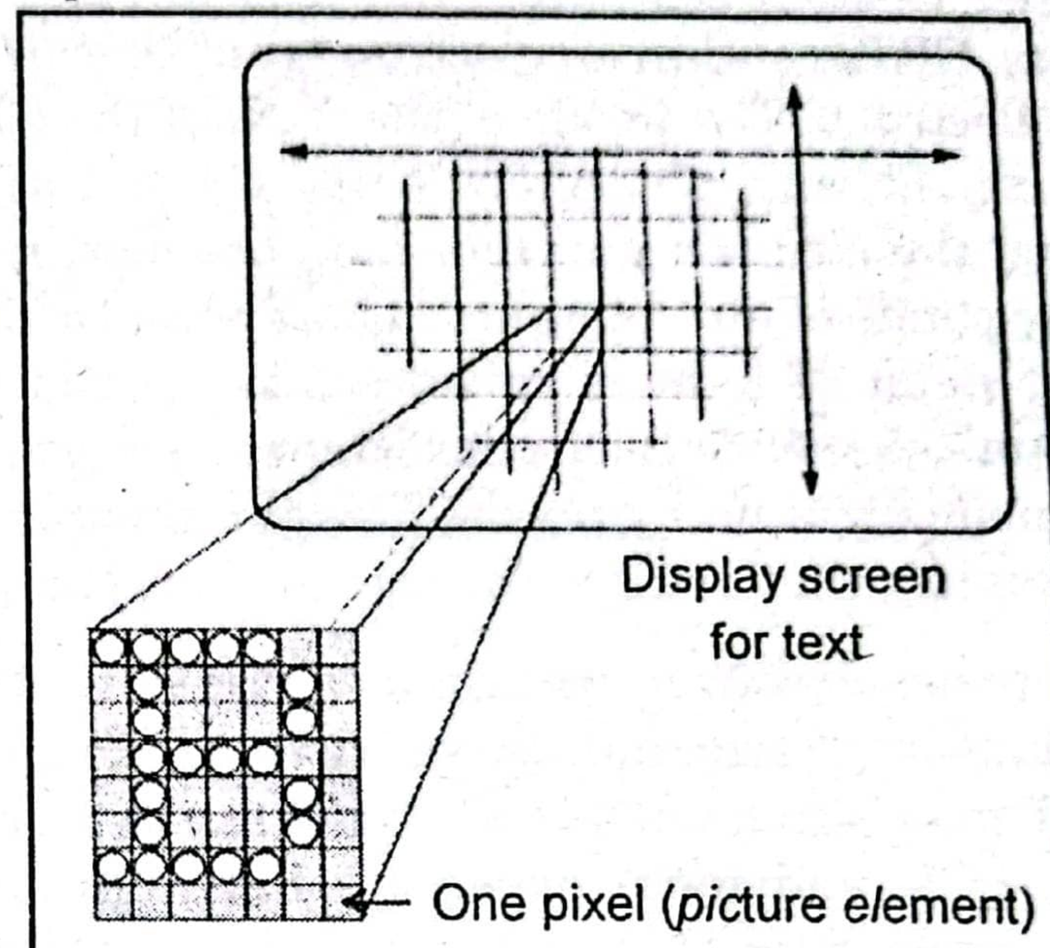


Fig. 1.16. Text displays divide a screen into 1,920 columns by 1080 rows.

Aspect Ratio: The *aspect ratio* defines a display's width relative to its height. A 2:1 aspect ratio, for example, means the display is twice as wide as it is tall. The aspect ratio for widescreen high-definition monitors is 16:9.

Refresh Rate: *Refresh rate* is the number of times per second that the pixels are recharged so that their glow remains bright. That is, refresh rate refers to the number of times that the image on the screen is redrawn each second. The higher the refresh rate, the more solid the image looks on the screen—that is, the less it flickers. In general, displays are refreshed 120, 144, or 240 times per second, or Hertz (Hz). A high-quality monitor has a refresh rate of 240 hertz—the screen is redrawn 240 times per second.

Monochrome/Color Screens: Screens are either monochrome or color. A monochrome monitor displays only a single-color image whereas a color monitor display multiple colors. An RGB (for Red-Green-Blue) monitor or a color monitor works just like a monochrome one, except that each dot on the screen is actually made up of three tiny red, green, and blue dots of compound called phosphors arranged in the shape of a triangle. Different colors can be displayed by combining various intensities of these three dots.

Color Depth: The number of colors a graphics processing unit displays is determined by bit depth. The *bit depth*, also called *color depth*, is the number of bits used to store information about each pixel. For example, a video card with a 24-bit depth uses 24 bits to store information about each pixel. Thus, this video card can display 2^{24} or 16.7

CamScanner

million colors, called *true color*. The greater the number of bits, the better the resulting image.

Cathode-Ray Tube Monitors

The cathode-ray tube (CRT) is an output display device that resembles an old television screen (see Fig. 1.17). This is the type of monitor found not only in older desktop computers but also in older television sets. CRTs contain an electron "gun" that fires a beam of electrons. The inside of the front of the CRT screen is coated with phosphor. When a beam of electrons from the electron gun hits the phosphor, it lights up tiny points of phosphor for a short period of time. Each tiny point of light is called a *pixel* (a contraction of picture element). Each character you see on the screen is made up of many pixels. The gun constantly scans the screen, relighting the pixels so rapidly that you see the screen characters as continuously glowing.

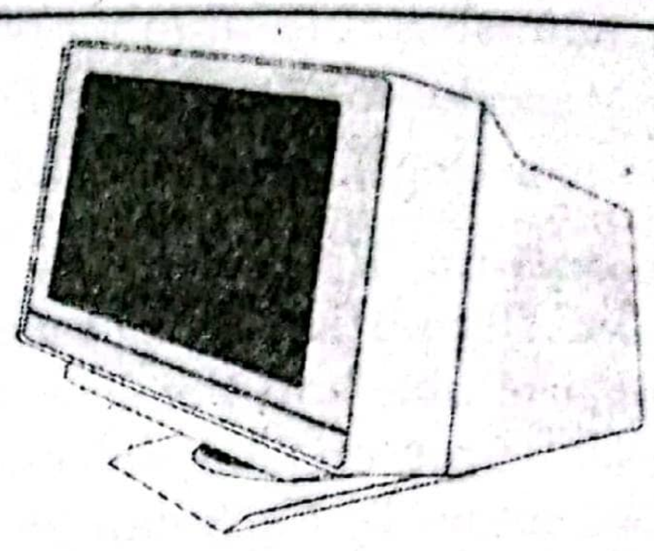


Fig. 1.17. A typical CRT monitor

There are, however, certain major disadvantages associated with CRT monitors. They include large size, high power consumption, fragility, and occasional flickering images. CRT monitors have lost nearly all market share, mostly due to competition from low-cost LCD and more expensive but high-contrast OLED flat-panel displays. These drawbacks have led to the development of flat panel technologies.

Flat Panel Monitors

Since the late 2000s, CRTs have been largely superseded by newer *flat panel* monitor technologies such as LCD, LED display, and OLED displays. LCD and OLED displays have lower manufacturing costs and power consumption, as well as significantly less weight and bulk. Flat panel displays can also be made in very large sizes.

Flat panel technology is particularly useful for laptop computers (Fig. 1.18), which are small enough to fit in a bag and powered by a small battery built into the computer.

There are three common types of flat panel monitors—LCD (liquid crystal display, LED (light-emitting diode) display, and OLED (organic light-emitting diode) display.



Fig. 1.18. A laptop computer, compact enough to fit in a bag.

Liquid Crystal Display (LCD): A liquid-crystal display (LCD) is a flat-panel display that uses the light-modulating properties of liquid crystals. Liquid crystals do not emit light directly, instead using a backlight or reflector to produce images in color or monochrome. The clear liquid crystals trapped in tiny pockets between two pieces of glass. Each pocket of liquid is covered both front and back by very thin wires. When a small amount of current is applied to both wires, a

chemical reaction turns the liquid crystals a dark color—thereby blocking light. The point of blocked light is the pixel.

The principal advantages of LCDs are:

1. Very compact, thin and light.
2. Low power consumption.
3. Little heat emitted during operation.
4. Low cost.

The biggest disadvantages are:

1. LCDs do not emit light; as a result, the image has less contrast.
2. The screen is very susceptible to glare, so the optimum viewing angle is very narrow.
3. Low refresh rate.
4. Black levels may not be as dark as required because individual liquid crystals cannot completely block all of the backlight from passing through.

Light-Emitting Diode (LED) Display: A light emitting diode (LED) display is a flat panel display, which uses an array of light-emitting diodes as pixels for a video display. LEDs are small semiconductors that emit visible light when an electrical current is passed through them. Their brightness allows them to be used outdoors where they are visible in the sun for store signs and billboards. Modern electronic devices such as mobile phones, TVs, tablets, computer monitors, laptops screens, etc., use a LED display to display their output.

The principal advantages of LED display are:

1. Efficient and low-energy consumption.
2. LEDs produce more brilliance and greater light intensity.
3. Rows of LEDs give better control of the light provided to the display and at greater efficiency as it is possible to turn off individual LEDs when they are not needed.

The biggest disadvantages are:

1. LEDs can shift color due to age and temperature.
2. Contrast ratios may not be consistent.
3. LED displays are largely depend on the correct engineering otherwise a lot of deterioration and defects can develop in the products.
4. Much higher price.

Organic Light-Emitting Diode (OLED) display: An organic light-emitting diode (OLED) is a light-emitting diode (LED) in which the emissive electroluminescent layer is a film of organic compound that emits light in response to an electric current (see Fig. 1.19). This layer of organic layers are situated between two electrodes; typically, at least one of these electrodes is transparent. OLEDs are

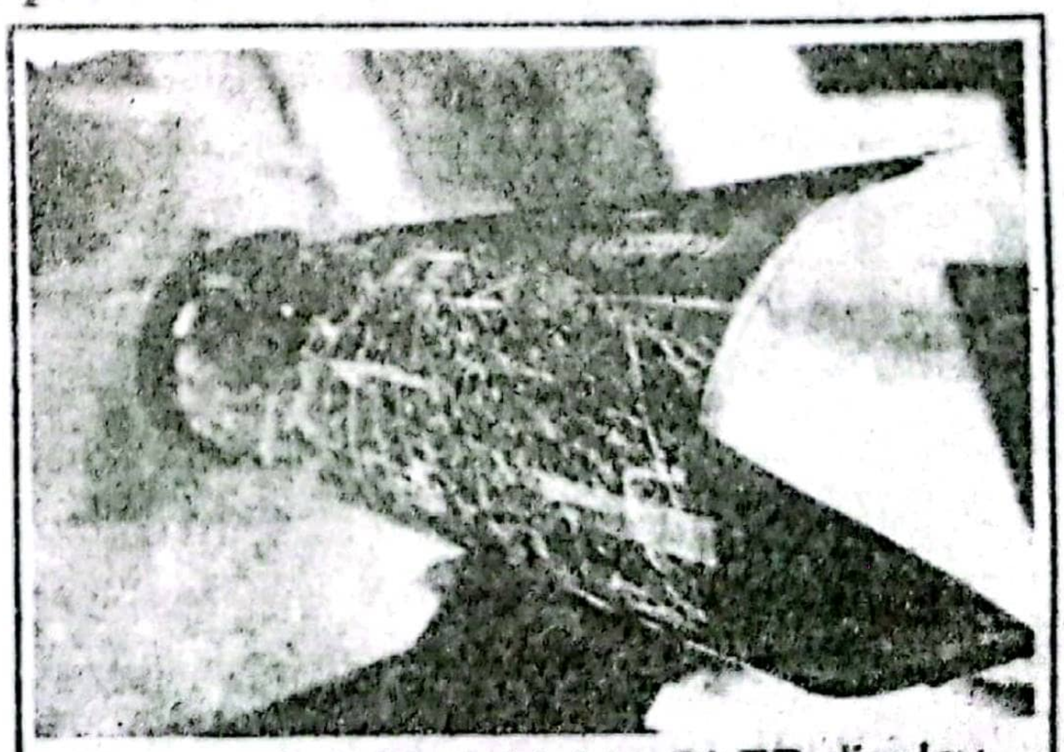


Fig. 1.19. LG's foldable OLED display

CamScanner

used to create digital displays in devices such as television screens, computer monitors, portable systems such as mobile phones, tablets and handheld game consoles.

The principal advantages of OLEDs over flat panel displays made with LCD technology are:

1. Lower cost of production in the future.
2. Lightweight, flexible and foldable.
3. Better picture quality, a greater contrast ratio and wider viewing angle compared to LCDs.
4. Better power efficiency and thickness.
5. Much faster response time than an LCD.

The biggest disadvantages are:

1. An old OLED display shows sign of wear. The biggest technical problem for OLEDs is the limited lifetime of the organic materials.
2. The OLED material produces blue light degradation significantly more rapidly than other colors, which create problem in color balance with time.
3. Water can instantly damage the organic materials of the displays.
4. As an emissive display technology, OLEDs rely completely upon converting electricity to light, enabling the display to be used without any internal light source. This leads to poor readability in bright ambient light such as outdoors.

Voices and Sounds

Sound is one of the more recent areas in which there have been developments for softcopy output. The two categories are speech generating devices and sound output.

Speech-Generating Devices (SGDs)

Speech-generating devices (SGDs), also known as *voice output communication aids*, are typically tablet-like units that allow a person to communicate thoughts by electronic voice generation when he or she is no longer able to speak (Fig. 1.20). Without an SGD, which is highly personalized and uniquely programmed, many people are isolated, trapped inside a body they cannot control, with no ability to communicate.

There are several input and display methods for users of varying abilities to make use of SGDs. Some SGDs have multiple pages of symbols to accommodate a large number of utterances, and thus only a portion of the symbols available are visible at any one time, with the communicator navigating the various pages. Speech-generating devices can produce electronic voice output by using digitized recordings of natural speech or through speech synthesis—which may carry less emotional information but can permit the user to speak novel messages.

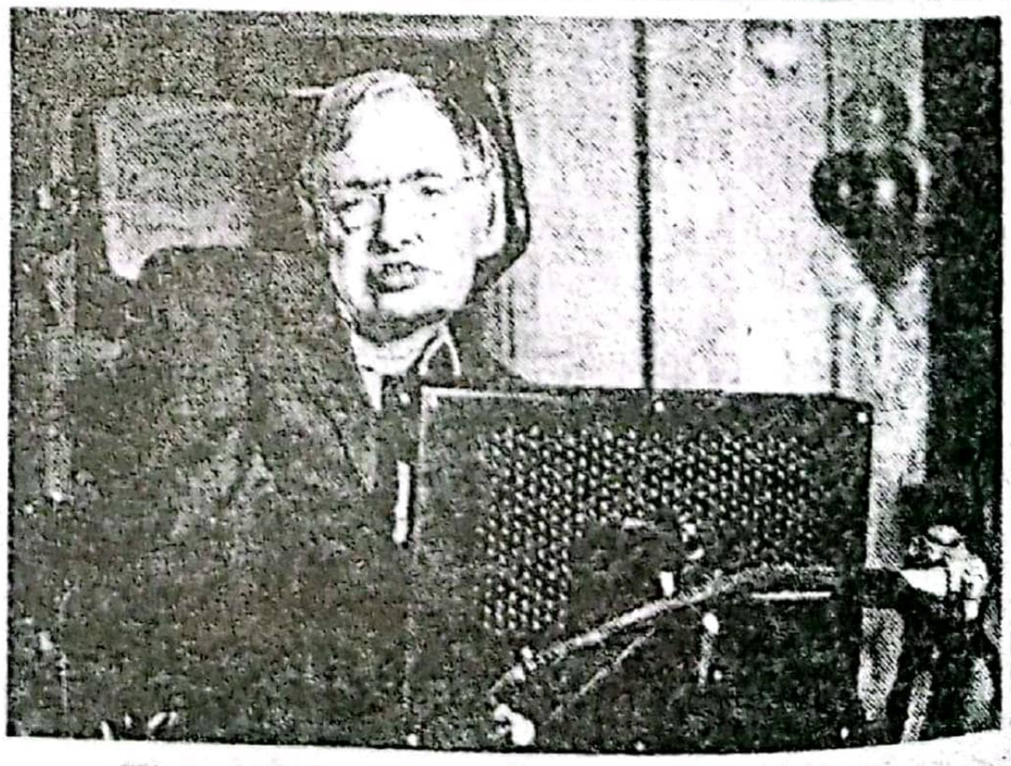


Fig. 1.20. Stephen Hawking, late astrophysicist and prominent SGD user

CamScanner

Using an SGD is different than using a communication app on a mobile device such as an iPad, as SGD devices are especially designed for the varying and specific needs of individuals with complex communication profiles. In some cases, a SGD device may be recommended even if a child has had some success using an app on a mobile device to communicate.

SGDs features are:

- A more robust language system
- More flexible software
- More durable construction
- A larger screen size
- Alternative access options

Sound-Output Devices

Sound-output devices produce digitized sounds, ranging from beeps and chirps to music. Sound capabilities can be added to some computers by inserting a sound card and appropriate software. The sound card plugs into an expansion slot in your computer; on newer computers, it is integrated with the motherboard (see Fig. 1.21).

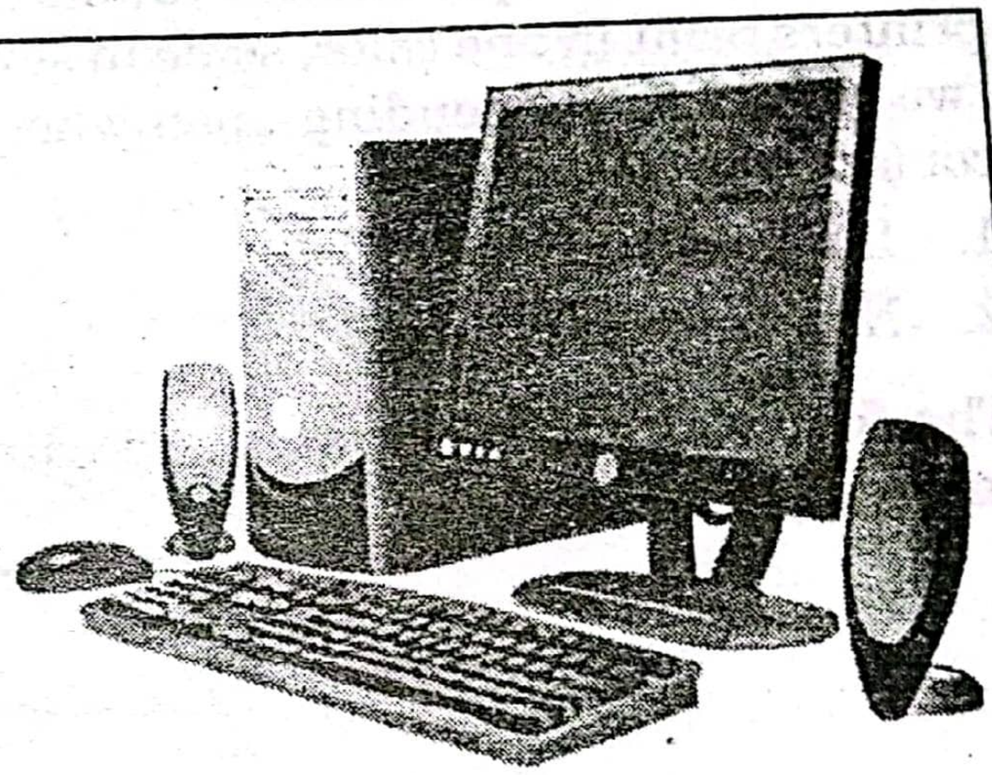


Fig. 1.21. A PC with sound-output device

There has been increasing interest in computer-generated sound with the rise in popularity of *multimedia systems,* which combine uses of text, graphics, video, animations, and sound.

Multimedia Projectors

A multimedia projector (Fig. 1.22) is an optical device that projects an image (or moving images) onto a surface, commonly a projection screen. Most projectors create an image by shining a light through a small transparent lens, but some newer types of projectors can project the image directly, by using lasers.

Multimedia projectors are digital replacements for earlier types of projectors such as slide projectors and overhead projectors. These earlier types of projectors were mostly replaced with multimedia projectors throughout the 1990s and early 2000s, but old analog projectors are still used at some places.

Many teachers find chalkboards to be almost a thing of the past with the advent of multimedia projectors in the classroom.

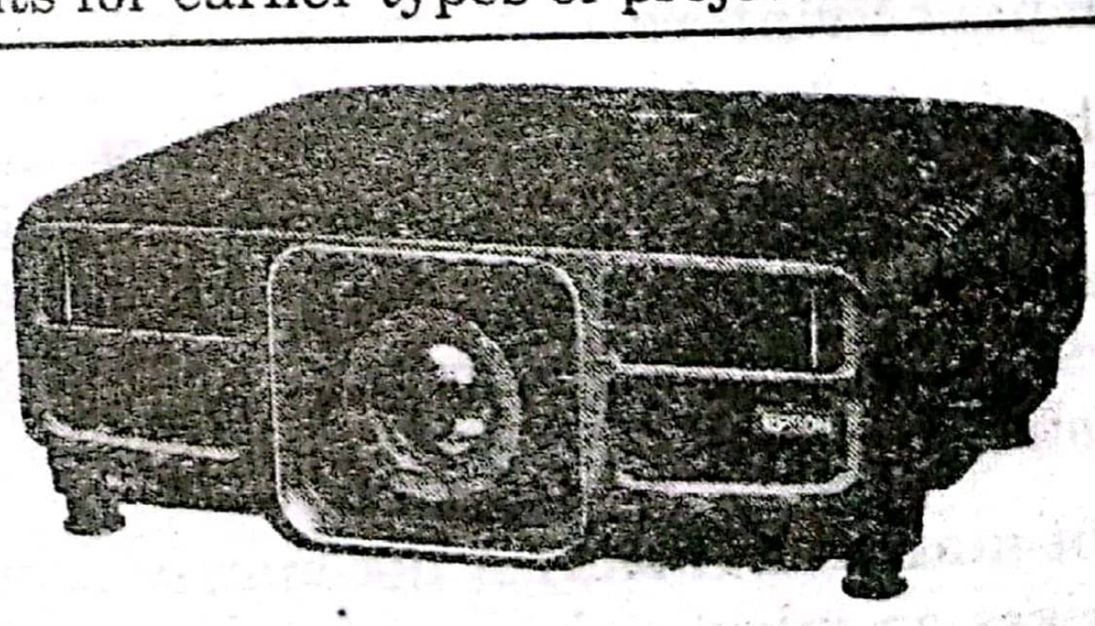


Fig. 1.22. A Multimedia projector

CamScanner

Rather than writing notes across a board, teachers can make use of PowerPoint presentations, images and even film as teaching tools through the use of projectors. Consequently, teachers and students alike find projectors to be useful classroom devices.

Hardcopy Output Hardware

Hardcopy output devices are of two principal types:

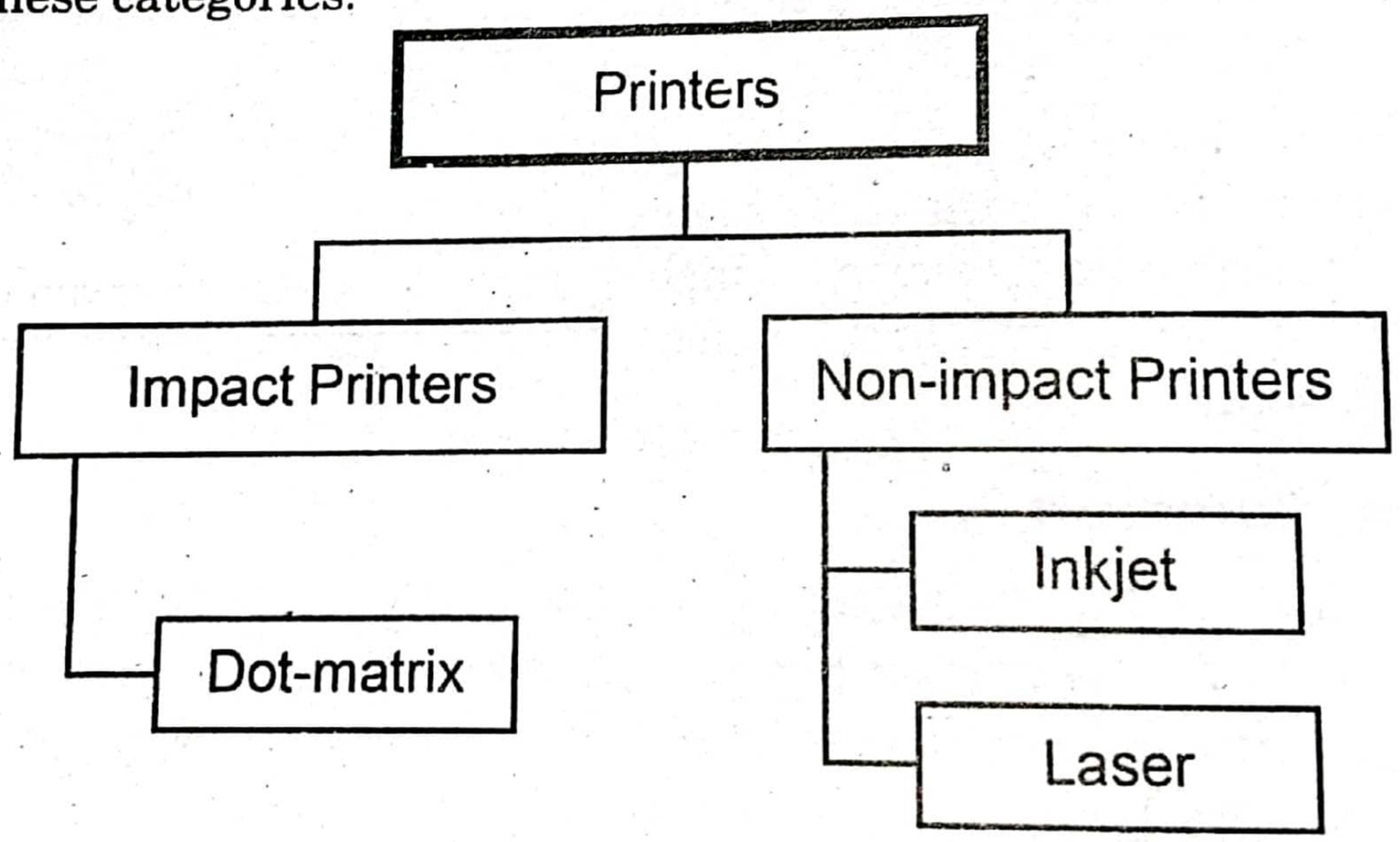
1. Printers
2. Plotters

Printers

Beside the monitor, the other important output device is the *printer*. A printer is a device that produces printed paper output—known in the trade as *hard copy* because it is tangible and permanent (unlike soft copy, which is displayed on a screen). Some printers print in one color, some in several colors. We categorize printers into following two categories, depending upon whether or not the print mechanism makes physical contact with the paper:

1. Impact Printers
2. Non-Impact Printers

The following chart shows these categories and some common varieties of printers included in these categories.



Impact Printers

The term *impact* refers to the use of some sort of physical contact with the paper to produce an image. Hence an impact printer forms images by physically striking an inked ribbon against paper with a hammer-like mechanism. In this sense, impact printers resemble typewriters. The most common form of impact printer is the dot-matrix printer.

Dot-matrix Printer: A dot-matrix printer (Fig. 1.23) uses a matrix of pins, which strikes an inked ribbon against paper, leaving an image in a computer-determined pattern of dots. A typical matrix is 5×7—that is five dots wide and seven dots high is

CamScanner

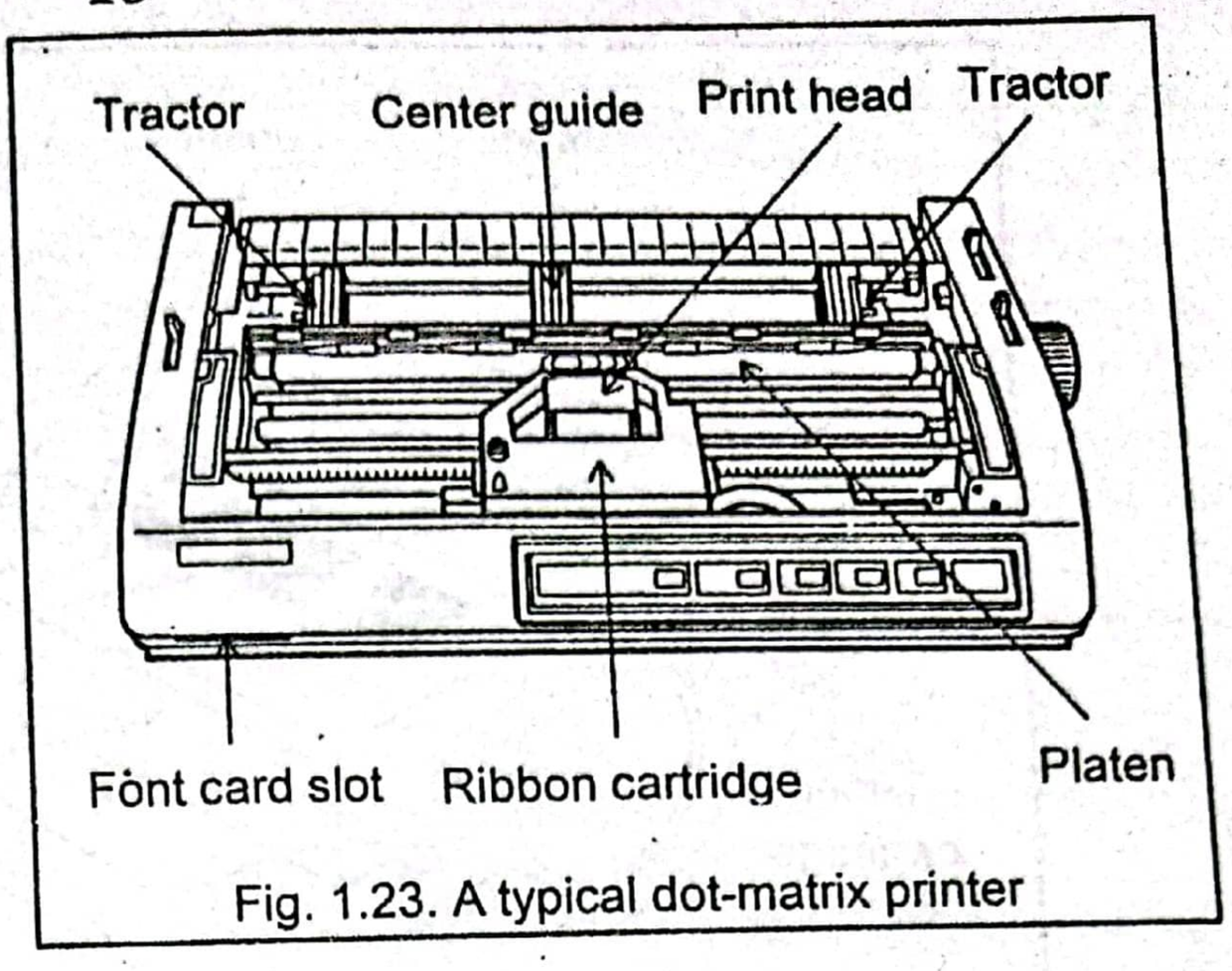


Fig. 1.23. A typical dot-matrix printer

shown in Fig. 1.24. These printers are sometimes called 9-pin printers (see Fig. 1.25), because they have two extra vertical dots for creating the parts of lowercase letters *g*, *j*, *p*, and *y* that go below the text line. The 24-pin dot-matrix printer uses a series of overlapping dots—the more dots, the better the quality of the letter produced.

A character may be produced as *draft quality*, a rough quality, when it is printed once. Or it may be produced as *near-letter-quality* (NLQ), as crisper image, when the character is printed twice. The second time, the print head is positioned slightly to the right of the original image. Some can also print in condensed mode, which allows more words to be fitted on a line. The printers print up to 340 characters per second and can handle graphics as well as text.

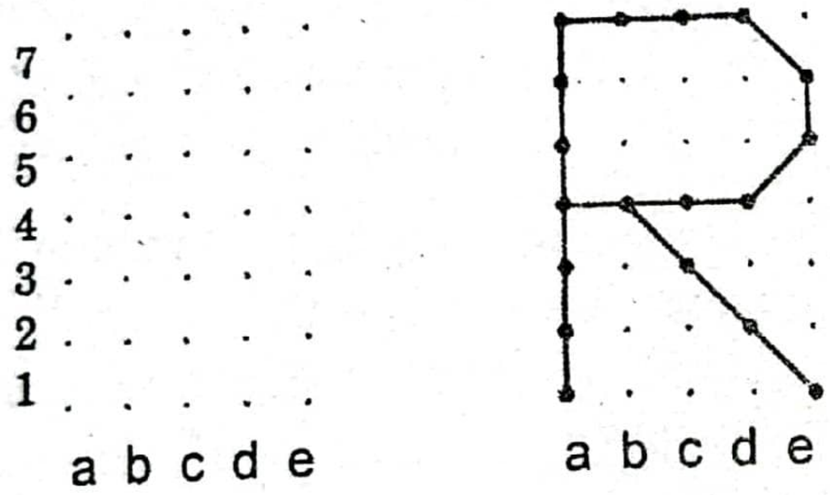


Fig. 1.24. A typical 5×7 matrix

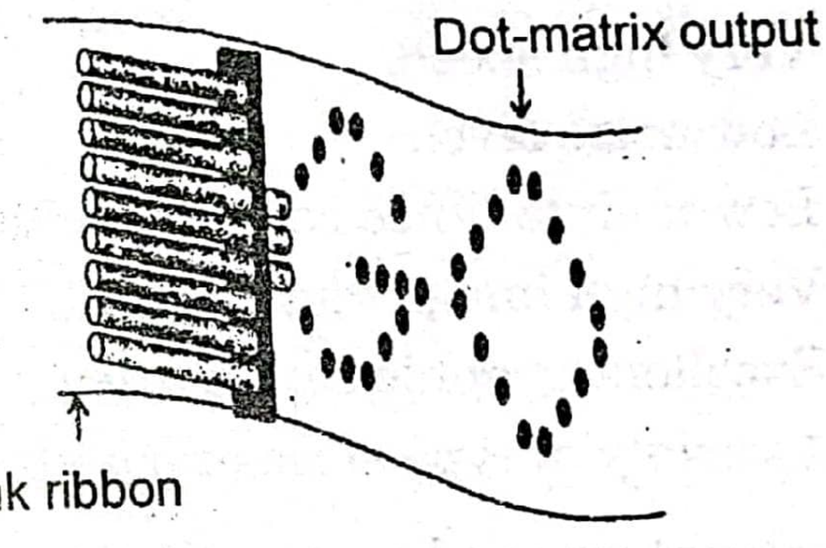


Fig. 1.25. The print head of 9-pin printer

Non-impact Printers

A non-impact printer forms images by using a non-contact process—that is, there is no physical contact between the printing mechanism and the paper. This kind of printer does not generate the kind of hammering noise that an impact printer does, but it cannot be used to print on multipart carbon forms. Because no impact is made on the paper, there can be no carbon copies. However, non-impact printers can produce letter-quality—that is, high-quality—images, which impact dot-matrix printers cannot. Following are some common types of non-impact printers.

Laser Printer: Laser printer technology is much less mechanical than impact printing (that is, no print heads move, no print hammers hit), resulting in much higher speed and quieter operation. The process resembles the operation of a photocopy machine: A laser beam "writes" an image onto the surface of a light-sensitive rotating drum (Fig. 1.26). Then ink-like toner is deposited on the drum; it adheres where the image was "written". The toner is then transferred to paper—a page at a time—in the same fashion as a photocopy machine.

CamScanner

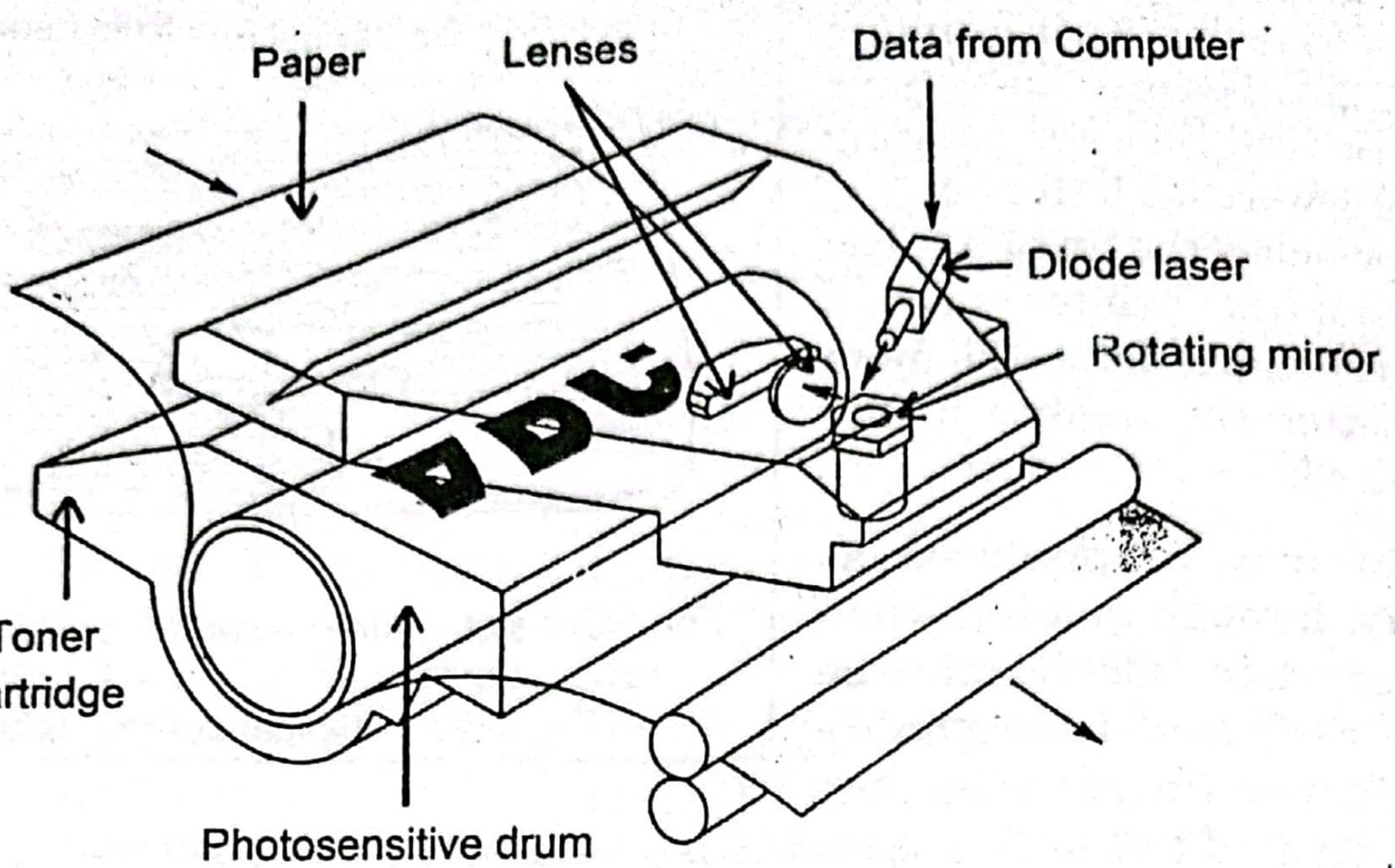


Fig. 1.26. Internal mechanism of a laser printer

The major advantages of laser printers are:

1. Very high speed.
2. Low noise level.
3. Low maintenance requirements.
4. Very high image quality.
5. Excellent graphic capabilities.
6. A variety of type sizes and styles.

Laser printers produce sharp, crisp images of both text and graphics. They are quiet and fast—able to print 11-33 pages per minute (ppm) in color and 10.5-37 black-and-white pages per minute for individual computers and up to 200 pages per minute for mainframes. They can print in different *fonts*—that is, sets of typestyles and type sizes. The more expensive models can print in color.

The resolution of laser printers is measured in dots per inch (DPI). The common laser printers have resolutions of 600-1200 DPI, both horizontally and vertically; some high-end models have resolutions of 2400 DPI. The printing industry stipulates a resolution of at least 2400 DPI for top-quality professional printing. In fact, however, many people cannot readily detect the difference between 1200 and 2400 DPI.

The desktop laser printer (Fig. 1.27) has become a very popular output device for personal computer-based systems. Many practitioners of desktop publishing prefer desktop laser printers because of the high-quality images they produce. The primary limitation of affordable desktop laser printers at this

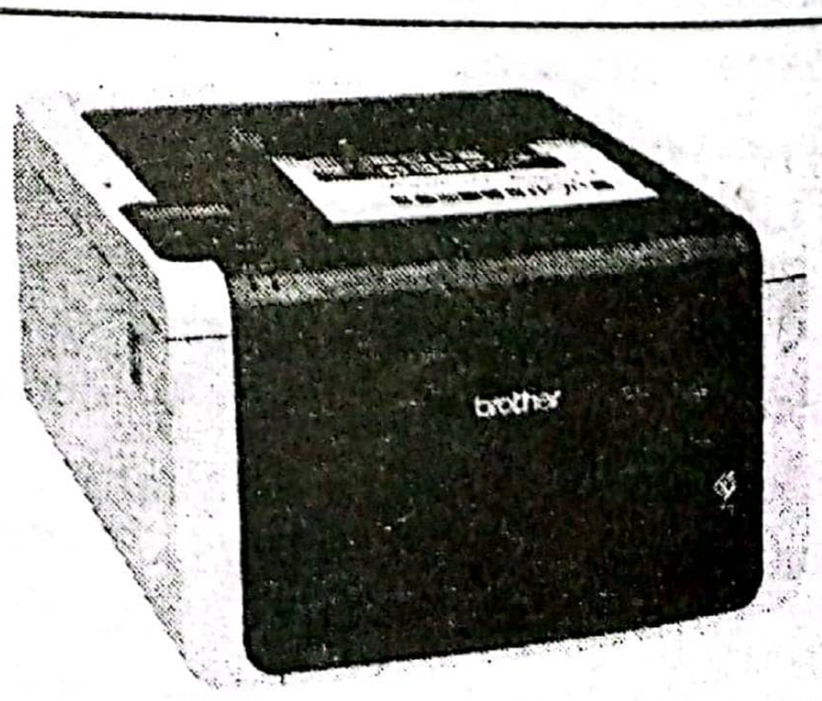


Fig. 1.27. The laser printer

CamScanner

time is restriction to single-color output. And desktop laser printers need a minimum of 16 MB-32 MB main memory (RAM) to build a high-resolution full-page image.

One particular group of laser printers has become known as PostScript printers. *PostScript* is a printer language, or page description language, that has become a standard for printing graphics with laser printers. A *page description language* (software) describes the shape and position of letters and graphics to the printer. PostScript printers are essential if you are printing a lot of graphics or want to generate fonts in various sizes.

Ink-Jet Printer: An ink-jet printer forms images by spraying tiny droplets of ink from jet nozzles onto the paper. The ink, which is charged, passes through an electric field, which deflects it to produce a dot-matrix character. Ink-jet printers, by using multiple nozzles, can print in several different colors of ink (Fig. 1.28). Color ink-jet printers produce excellent graphics.

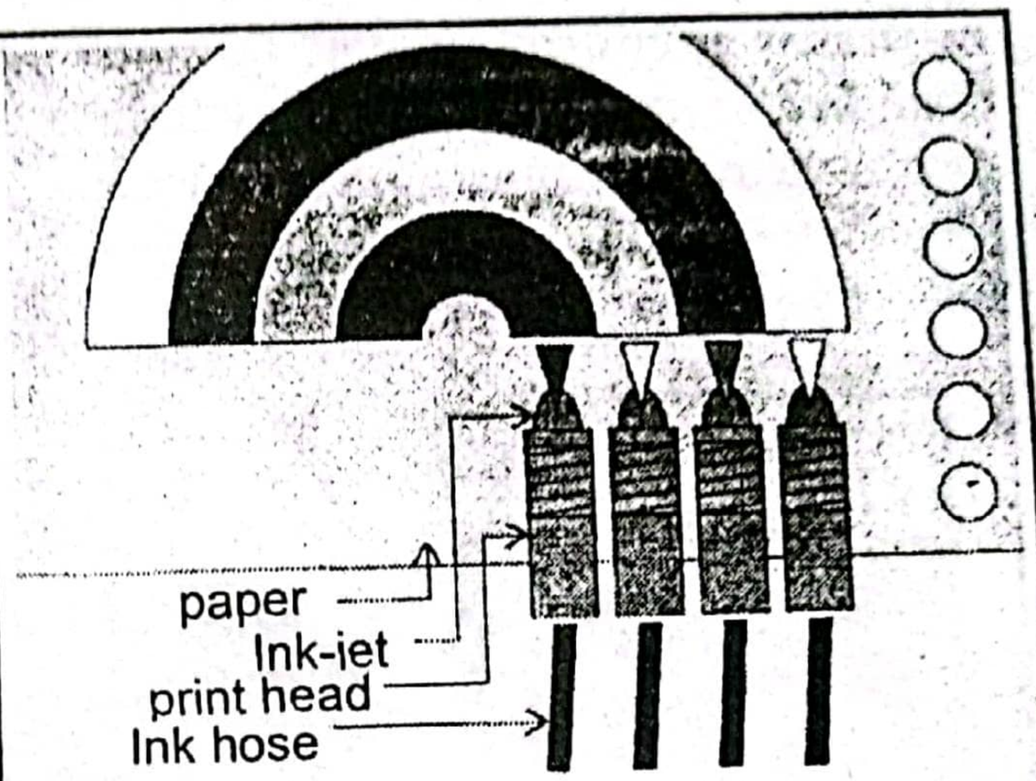


Fig. 1.28. Ink-jet printer. Both free-form graphics and text can be produced—in black and white and in color—using an ink-jet printer

One variation of the ink-jet printer is the *bubble-jet printer*. This printer uses a rising and falling bubble to force a droplet of ink onto the paper.

In general, ink-jet printers offer an excellent middle ground between dot-matrix and laser printers. Inkjet printers commonly have a dpi of 4,800×1,200 (but can be as high as 9,600×2,400). Like laser printers, ink-jet printers are quiet and convenient, but they are not particularly fast. Inkjet cartridges come in various combinations: a single cartridge for black and all color inks, two separate black and color cartridges, or separate cartridges for black and each color. Some cartridges also include the print head, which is apt to wear out before the rest of the machine.

Table 1.1. Comparison of Some Common Printers

Type	Technology	Advantages	Disadvantages	Approximate Speed
Dot-matrix	Impact: 9-, 18-, or 24-pin print head	Inexpensive, fast; can print graphics	Noisy, characters not letter quality; can't print shades of grey; limited color capacity	up to 340 characters per second (cps)
Laser	Laser beam directed onto a drum, "etching" spots that attract toner, which is then transferred to paper.	Quiet; excellent quality output; prints graphics and color; very high speed	Relatively high cost; especially for color	11-33 ppm
Inkjet	Electrostatically charged drops hit paper	Quiet; prints color; less expensive than laser; prints graphics	Relatively slow; clogged jets; output not as crisp as laser	3-13 ppm

CamScanner

The advantages of inkjet printers are that they can print in color, are quiet, and are generally less expensive than color laser printers.

The disadvantages have been that they print a bit less precisely than laser printers do and are slower. Another disadvantage is that inkjet cartridges have to be replaced more often than laser-toner cartridges do and so may cost more in the long run.

Plotters

A plotter, as shown in Fig. 1.29, is a special kind of output device. It is like a printer in that it produces images on paper, but it does so in a different way. Plotters are designed to produce large drawings or images, such as construction plans for buildings or blueprints for mechanical objects.

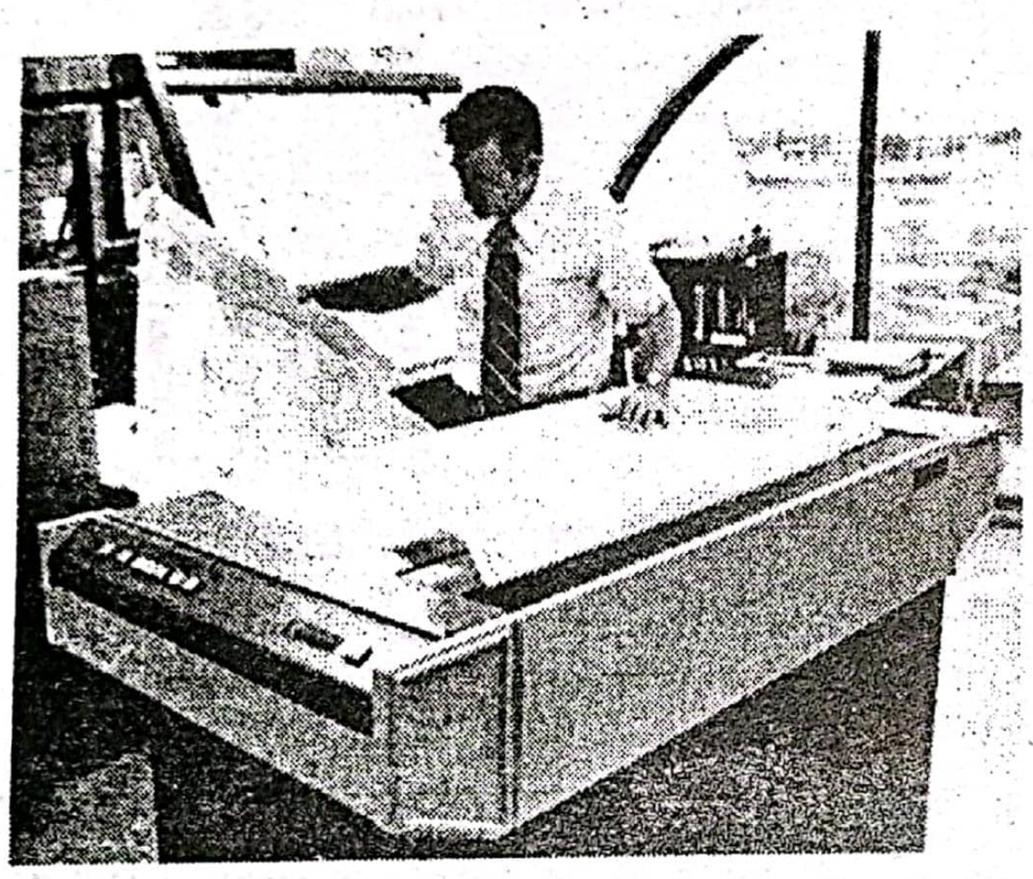


Fig. 1.29. Flatbed plotter

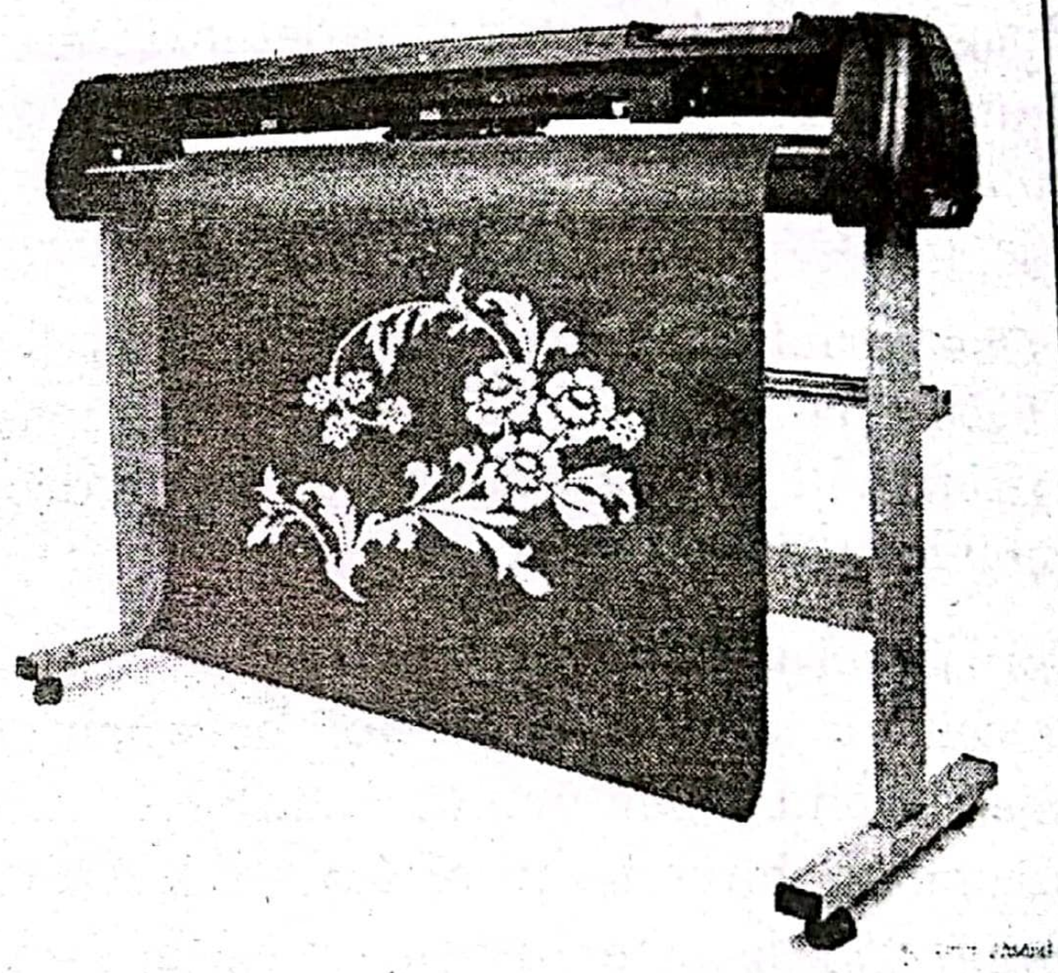


Fig. 1.30. Plotter output example

A plotter uses a robotic arm to draw with colored pens on a sheet of paper. The instructions that a plotter receives from a computer consist of a color, and beginning and ending coordinates for a line. With that information, the plotter picks up the appropriate pen, positions it at the beginning coordinates, drops the pen down to the surface of the paper and draws to the ending coordinates (see Fig. 1.30). Plotters draw curves by creating a sequence of very short, straight lines. Plotters can accommodate paper roll with width up to 42 inches and lengths of 100-150 feet because blueprints, maps, signs, posters and other such drawings and displays can be quite large. Among the types of plotters are:

Pen Plotters: Pen plotters use one or more colored pens to draw on paper. Such plotters may be *drum plotters*, in which the paper is rolled on a drum as pens are held against it. Or they may be *flatbed plotters*, in which the paper is held steady as the pen moves around on the paper.

Electrostatic Plotters: Electrostatic plotters create images by forming electrostatic charges on special paper, then running the paper through a developer. These are faster

CamScanner

than pen plotters and produce high-quality images.

Large-format Plotters: Large-format plotters operate somewhat like an inkjet printer but on a much larger scale. This type of plotter is often used by graphic artists.

Storage Hardware

A limitation of any computer is the size of its primary storage, also called *main memory*. Generally smaller in size, primary storage devices are designed to hold data temporarily and are internal to the computer. They have the fastest data access speed. These types of devices include RAM and cache memory. Moreover, computers lose everything in main memory when the power is turned off, since main memory is *volatile*. These problems created the need for a secondary means of storage that could retain data in a permanent form.

The *secondary storage*, or *storage medium*, is permanent storage. This storage is non-volatile. The contents still exist after the power is turned off. Some types of storage devices are shown in Fig. 1.31.

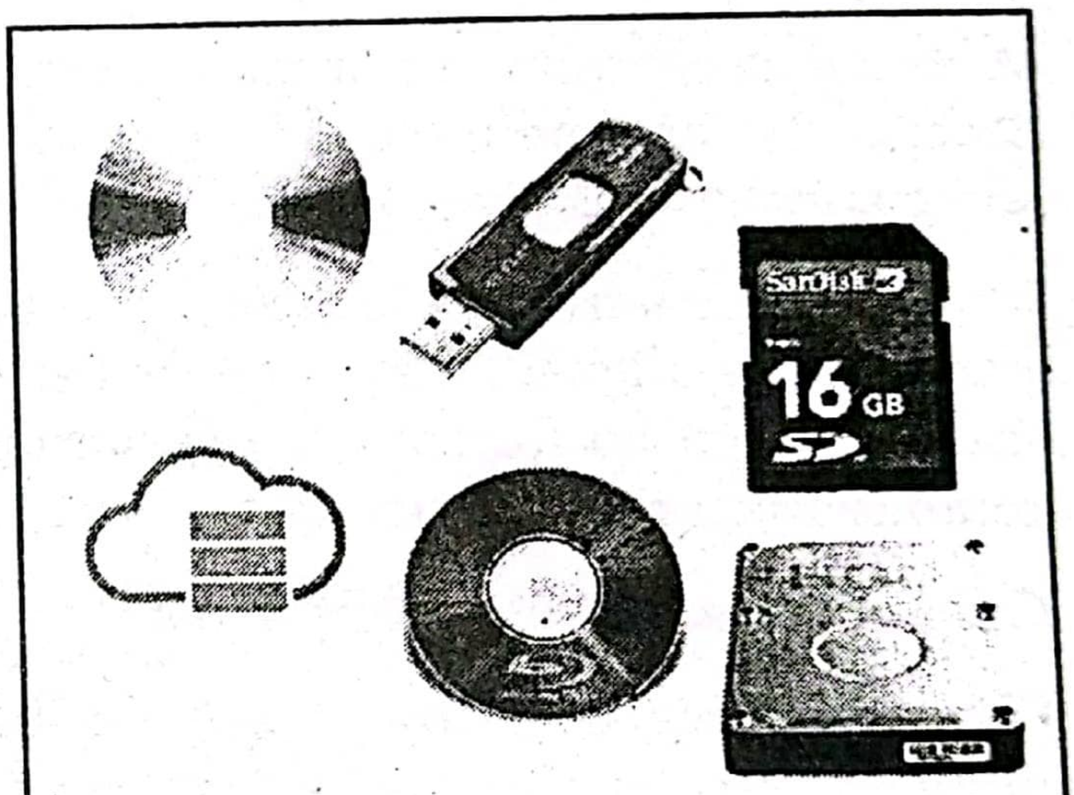


Fig. 1.31. Secondary storage devices

An important feature of a storage device is the speed with which a word can be read out or written. This is termed the *access time* and is the sum of two quantities, *i.e.*, the time taken to locate the particular storage device and the time taken to fill or empty it. In a *random-access* store, *e.g.*, primary store, the time taken to locate word is constant. In a *cyclic access* store, *e.g.*, disk, information circulates in a loop and can be obtained only when the appropriate section of the loop passes the reading device. On average, the access time is equal to half the cycle time for the loop.

Capacity is the number of bytes (characters) a storage medium can hold. For example, a reasonably priced USB flash drive can store up to 256 GB of data (approximately 256 billion bytes) and a typical hard disk has 4 TB (approximately 4 trillion bytes) of storage capacity.

The benefits of secondary storage are:

1. *Economy:* It is less expensive to store data on hard disk, solid state drives, USB flash drive or optical disk, the principal means of secondary storage, than in filing cabinets. Inexpensive storage space permits the storage of vast amounts of data, with increased accuracy in saving and retrieving it.
2. *Reliability:* Data in secondary storage is basically safe, since secondary storage is physically reliable. Also, the data in it is stored in such a way that data tampering is difficult.
3. *Convenience:* With the help of a computer, authorized people can locate and access data quickly.

CamScanner

Types of Storage Devices

Four main technologies are used to store data today: magnetic, optical, solid state and cloud storage. Although devices that store data typically employ one or the other, some combine multiple technologies. Some examples of each type of technology are:

1. Magnetic storage—hard disk
2. Optical storage—CD-ROM, DVD-ROM, Blu-ray disks etc.
3. Flash storage—solid state drive, memory card, and USB flash drive etc.
4. Cloud storage—OneDrive, Google Drive, and iCloud etc.

Magnetic Storage—Hard Disks

Hard disk, also called *fixed disk, hard disk drive* or *hard drive,* is the basic storage device for all computers. Since a hard disk stores so much data, it is sometimes called a *mass-storage device*—along with solid state storage and optical disks, and other media that can store a great deal of data.

A hard disk contains one or more inflexible, circular platters that use magnetic storage technology for reading and writing data. The surfaces of hard disk is coated with a magnetically sensitive material, which reacts to a magnetic field.

The data on a hard disk is recorded as magnetic spots. Depending on how the magnetic particles are aligned, they represent either a 0 bit or a 1 bit. Before a hard disk can be used, it must be formatted. *Formatting*, or *initializing*, is preparing a disk so the computer can record data on it.

Characteristics of a Hard Disk

Characteristics of a hard disk include its capacity, platters, read/write heads, cylinders, sectors and tracks, revolutions per minute, transfer rate, and access time.

The *capacity* of a hard disk is determined from the number of platters it contains, the composition of the magnetic coating on the platters, and whether it uses longitudinal or perpendicular recording.

A *platter* is made of aluminum, glass, or ceramic and is coated with a material that allows data to be recorded magnetically on its surface.

The platters are paired with magnetic *read/write heads*, usually arranged on a moving *actuator arm*, which read and write data to the platter surfaces. It is similar in appearance to the arm of a record player. Data is accessed in a random-access manner, meaning that individual blocks of data can be stored or retrieved in any order and not only sequentially. As the platter spins, the actuator arm adjusts the read/write heads to facilitate the reading, writing, or deletion of information.

The first thing a hard disk does when formatting a disk is to create a set of magnetic concentric circles called *tracks* (see Fig. 1.32). The number of tracks on a hard disk varies with the type/capacity of disk. The tracks on a hard disk are concentric circles and do not form a continuous spiral. Also, these tracks have no visible grooves and present smooth surface, since they are simply composed of magnetically encoded data. Even though tracks get successively smaller toward the hub of the hard disk, each

CamScanner

contains the same amount of data; data is just recorded more densely. In the hard disk, read/write heads move in small increments across the data access area of the disk to find the appropriate track.

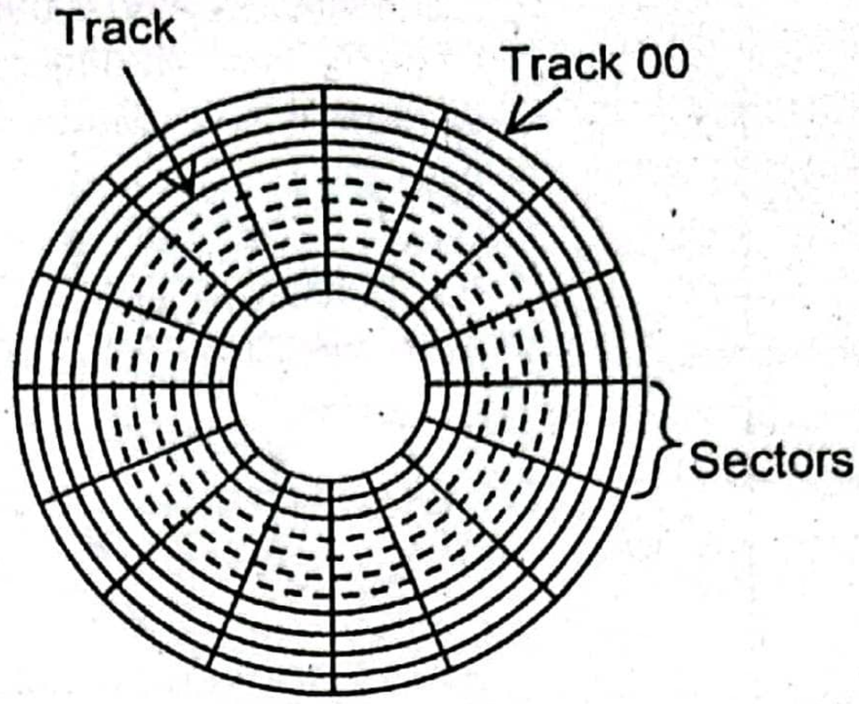


Fig. 1.32. Surface of a hard disk

Each *track* on a hard disk is divided into pie-shaped sections, which break the tracks into small arcs called *sectors*. These sectors are used by the computer system for storage reference purposes. Each sector stores a fixed amount of data, traditionally 512 bytes. Newer hard disks use 4096-byte (or 4 Kilobytes) sectors, which are known as the Advanced Format (AF). Sometimes, a sector has a flaw and cannot write/read data. When you format a disk, the operating system marks these bad sectors as unusable.

For reading and writing purposes, sectors are grouped into *clusters*. A cluster is the smallest unit of disk space that stores data. Each cluster, also called an *allocation unit*, consists of two to eight sectors (the number varies depending on the operating system). Even if a file is very small and consists of only a few bytes, it uses an entire cluster. Each cluster holds data from only one file. One file, however, can span many clusters.

While the computer is running, the platters in the hard disk rotate at a high rate of speed. This spinning, which usually is 7200 to 15,000 RPM (revolutions per minute), allows nearly instant access to all tracks and sectors on the platters. The platters may continue to spin until the computer is turned off, or more commonly today, they stop spinning or slow down after a specified time to save power.

Depending on the type of hard disk, *transfer rates* range from 15 MBps to 320 MBps. *Access time* for today's hard disks ranges from about 3 to 12 ms (milliseconds). Hard disks improve their access time by caching, specifically using disk cache. Disk cache (pronounced cash), sometimes called a *buffer*, consists of a memory chip(s) on a hard disk that stores frequently accessed data.

Construction of a Hard Disk

A hard disk contains one or several rigid platters on which data may be recorded (Fig. 1.33). There may be as many as 12 platters in a unit, all connected to a central spindle or shaft. The hard disk drive, or hard drive, is the whole unit which encloses the disk and drive. It includes the hard disk, the motor that spins the platters, and read/write heads. You generally cannot remove the hard disk from its drive; the two terms are used interchangeably to mean the whole unit, both disk and drive.

Hard disks have become the primary storage device for personal computers because they are incredibly convenient. Today, hard disks with the capacity range of tera-bytes (TB) are very common. With a hard disk, most people can store all their programs and files in one location. Still hard disks do not eliminate the need for a USB flash drive.

The hard disks are addressed as data located on surface number, sector number, and track number. Remember that hard disks consist of several platters stacked one atop

CamScanner

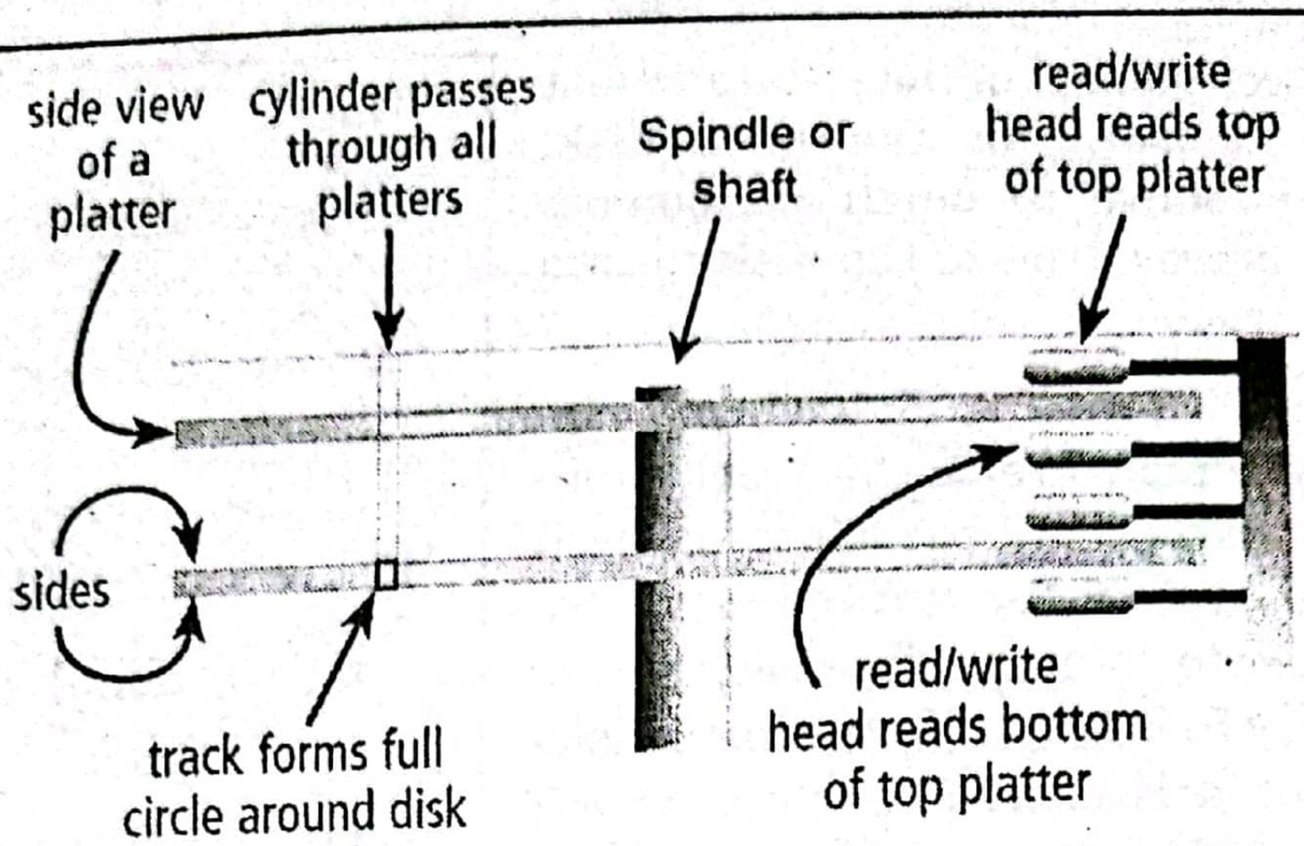


Fig. 1.33. A disk having 2 platters. Read/write heads are positioned over both sides of each platter

another (Fig. 1.33). To the disk-drive controller, this configuration just means that the disk has more than two sides; in addition to a side 0 and side 1, there are sides 2, 3, 4, and so on. Some hard-disk drives hold as many as 12 disks, but both sides of disks are not always used.

With hard disks, the number of sides that the disk uses are specified by the number of heads. For example, a particular hard disk drive might have six disk platters (that is, 12 sides), but only ten heads, indicating that two sides are not used to store data. Often, these are the top side of the top disk and the bottom side of the bottom disk.

Because hard disks are actually a stack of platters, we use a new term, *cylinder*, to refer to the same track across all the disk sides, as shown in Fig. 1.33. For example, track 150 on every disk is cylinder 150. A single movement of the read/write head arms accesses all the platters in a cylinder. If a hard disk has two platters (four sides), each with 1,000 tracks, then it will have 1,000 cylinders with each cylinder consisting of 4 tracks (2 tracks for each platter).

The computation of a hard disk's capacity is very straightforward. Here is the breakdown for a disk that is sold as a 50 GB disk:

12,024 cylinders × 22 heads (sides)	= 264,528 tracks
264,528 tracks × 369 sectors/track	= 97,610,832 sectors
97,610,832 sectors × 512 bytes/sector	= 49,976,745,984 bytes

With a brand-new hard disk, the computer will try to place the data in clusters that are *contiguous*—that is, that are adjacent (next to one another). Thus, data would be stored on track 1 in sectors 1, 2, 3, 4, and so on. However, as data files are updated and the disk fills up, the operating system stores data in whatever free space is available. Thus, files become fragmented. *Fragmentation* means that a data file becomes spread out across the hard disk in many non-contiguous clusters.

Fragmented files cause the read/write head to go through extra movements to find data, thus slowing access to the data. This means that the computer runs more slowly than it would if all the data elements in each file were stored in contiguous locations.

CamScanner

To speed up the disk access, you must defragment the disk. *Defragmentation* means that data on the hard disk is reorganized so that data in each file is stored in contiguous clusters. Programs for defragmentation are available on some operating systems or as separate software utilities.

Example 1.1

Consider a disk unit which is containing 7 platters. Each platter has 80 tracks on one side and each track is divided into 9 sectors. The storage capacity of each sector is 256 bytes or characters. Storage facility cannot be accessed to top and bottom surface of housing unit. Compute the total number of characters that can be stored on this disk.

Solution

Housing unit has 7 platters, *i.e.*, number of surfaces = $(7 \times 2) - 2 = 12$ surfaces are available for storing data.

Number of sectors on each surface = number of tracks × number of sectors
= 80 × 9 = 720 sectors

but 1 sector = 256 bytes

∴ Storage capacity of each surface = number of sectors × number of bytes
= 720 × 256 = 184320 bytes

Storage capacity of disk unit = number of surfaces × storage capacity of each surface
= 12 × 184320 = 2211840 bytes

but 1024 bytes = 1 KB

∴ Storage capacity of disk unit $= \frac{2211840}{1024} = 2160$ KB

Optical storage—CD-ROM, DVD-ROM, Blu-ray Disks

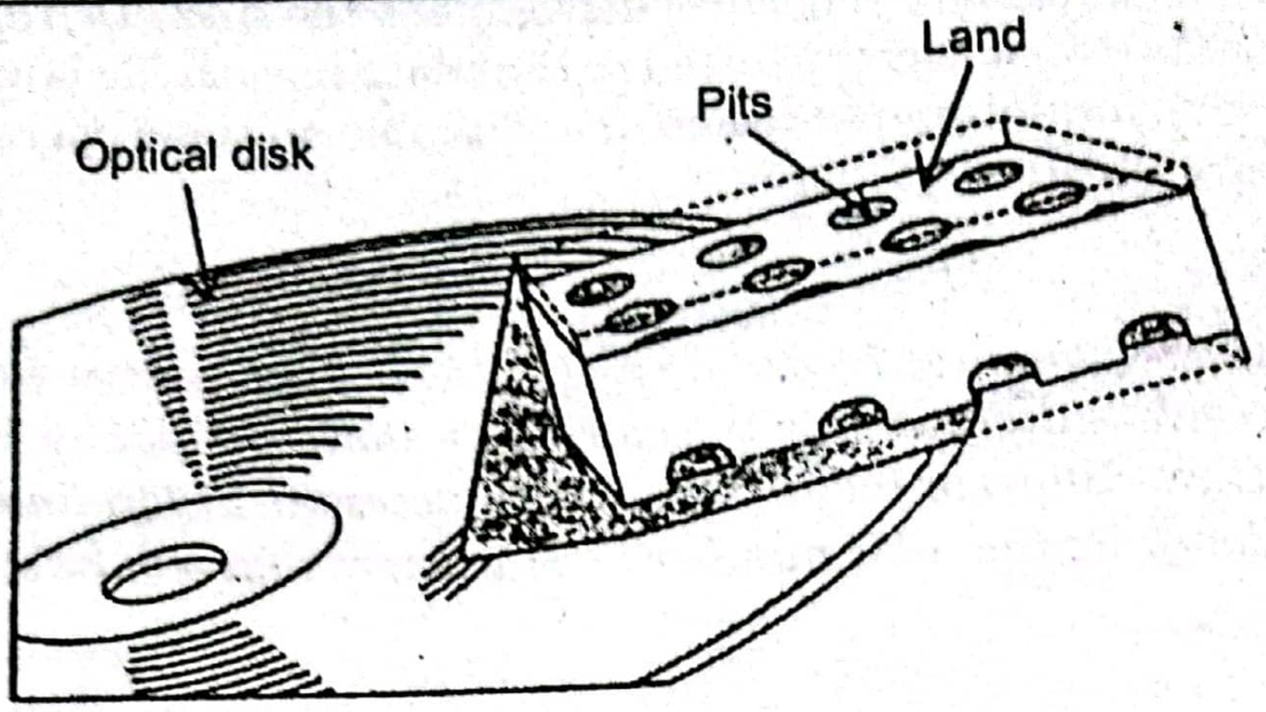
The need for ever greater storage capacities drives hardware manufacturers to improve the available technologies and to develop alternative storage media. Today, *optical storage* is the main alternative to magnetic storage.

Optical disks used in personal computers are removable disks, 4.75 inches in diameter and less than one-twentieth of an inch thick. Smaller computers, game consoles, and mobile devices, however, often use mini disks that have a diameter of 3 inches or less.

Optical storage techniques make use of the laser beams to write and read data—which can consist of text, graphics, audio clips, or video images. A single optical disk of the type called CD-ROM can hold up to about 700 megabytes of data. This works out to about 269,000 pages of text, or more than 7500 photos or graphics, or 20 hours of speech, or 77 minutes of video. Although some disks are used strictly for digital data storage, many combine text, visuals, and sound.

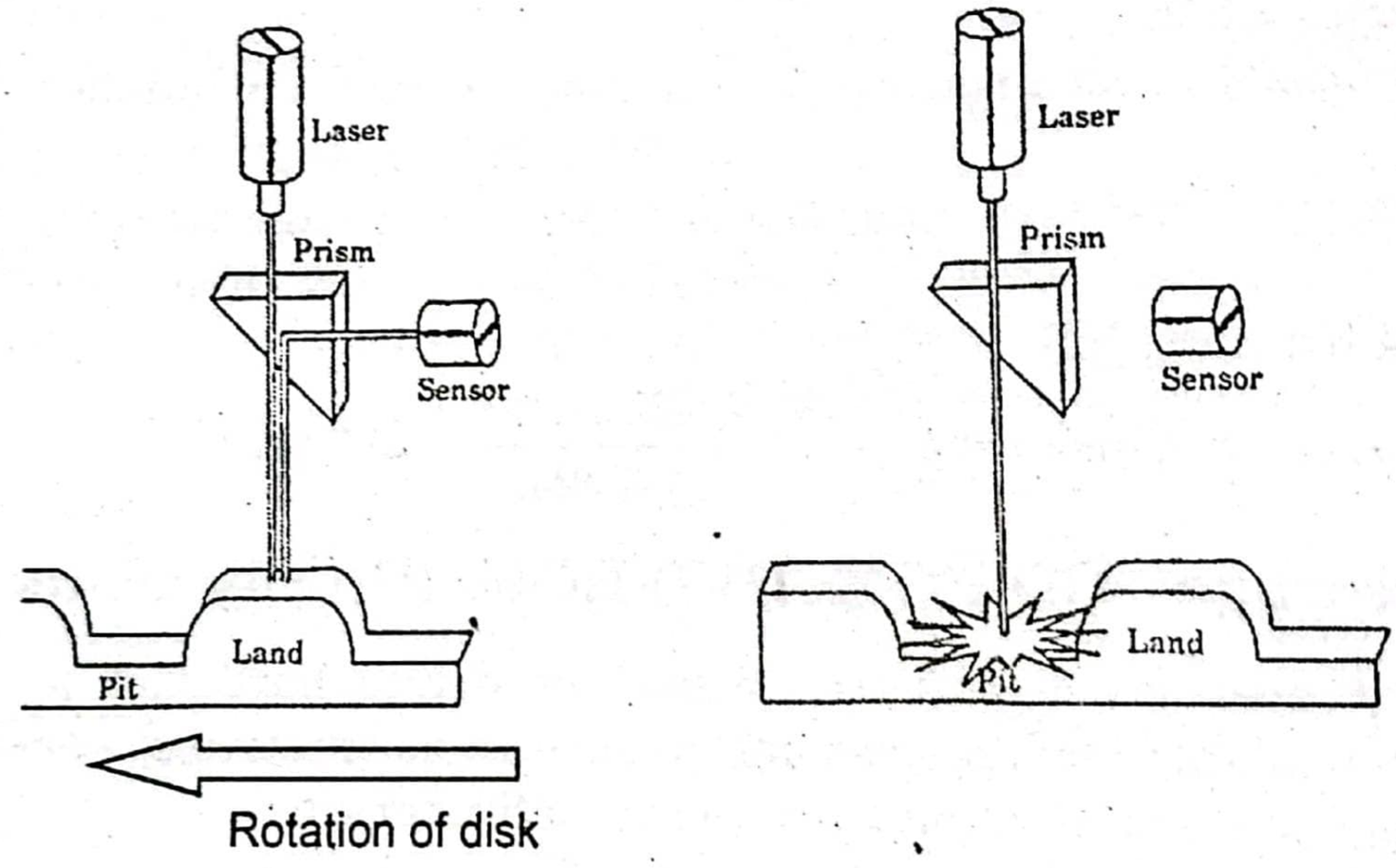
In the principal types of optical-disk technology, a high-power laser beam is used to represent data by burning tiny *pits* into the surface of a hard plastic disk. To read the data, a low-power laser light scans the disk surface: pitted areas are not reflected and are interpreted as 0 bits; smooth areas, called *land*, are reflected and are interpreted as 1 bits (see Fig. 1.34). Because the pits are very tiny, much more data can be represented than is possible in the same amount of space on a magnetic hard disk.

CamScanner



Reading "1":
The laser beam reflects off the smooth surface, which is interpreted as a 1 bit.

Reading "0":
The laser beam enters a pit and is not reflected, which is interpreted as a 0 bit.



(b) Reading data

Fig. 1.34. The surface of an optical disk is etched with tiny lands and pits that are the equivalent of binary 1s and 0s. Lands reflect light of a sensor; pits scatter light.

The cost of optical media has proven to be about 1/100 that of magnetic disk. In addition, optical disks are more durable. The data recorded on optical media is not damaged by stray magnetic fields and surface problems like finger-prints, dust, and so on. Moreover, nothing touches the disk's surface except a beam of light, they are less at risk for deterioration or contamination. A current disadvantage, however, is that it takes longer to retrieve data from an optical disk than from a magnetic hard disk.

Many types of optical disks are available for computer users. Among them the commonly available optical disks are:

1. CD-ROM Disks
2. CD-R Disks
3. CD-RW Disks

CamScanner

4. DVD-ROM Disks
5. DVD-R Disks
6. DVD-RW, DVD-RAM, DVD +RW Disks
7. Blu-Ray Disks

CD-ROM Disks

CD-ROM, which stands for *compact disk–read-only memory*, is an optical-disk format that is used to hold prerecorded text, graphics, sound and videos (Fig. 1.35). Data on disk is imprinted by the disk manufacturer and cannot be altered or erased by the user. *Read-only* means that data can be read but not altered or erased by the user. This type of optical disk is used to store huge volume of data that rarely change. For example, dictionaries, encyclopedias, medical, legal, or other professional reference libraries, music and video all required tremendous amounts of data that you wouldn't normally want to alter.

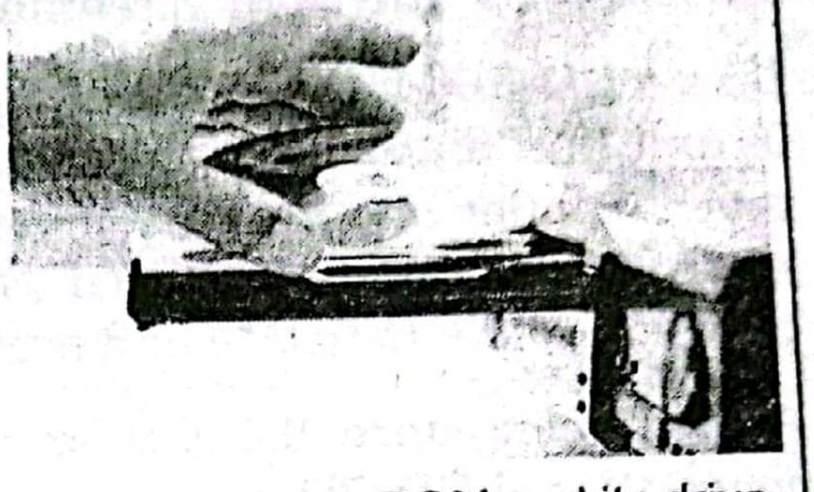


Fig. 1.35. A CD-ROM and its drive

At one time a CD-ROM drive was only a single speed drive. A single-speed drive access data at 150 kilobytes per second. Now most CD-ROM disk drives available in the market are labelled as 40X, 48X, 52X, 56X and so on. These labels convey the speeds of the drives. For instance, 48X indicates forty-eight times the speed of the 'single-speed drive' that was originally manufactured.

CD-R Disks

CD-R (*compact disk–recordable*) disks can be written to only once but can be read many times. This allows users to make their own CD disks. Once recorded, the information cannot be erased. CD-R is often used by companies for archiving—that is, to store vast amounts of information.

CD-R disks have much greater storage capacities—650 MB to 5 GB—than CD-ROMs. This quality makes it ideal medium for making a permanent record of important data. Many banks use CD-R disks to store a record of each day's transactions. The transactions are written to an optical disk and become a permanent record that can be read but never altered.

CD-RW Disks

CD-RW (*compact disk–rewritable*) disks are both readable and writable; their contents can be erased or altered by the user. These in effect resemble magnetic disks, although they store huge amount of data. The CD-RW, also known as *magneto-optical (MO) disk*, uses aspects of both magnetic-disk and optical-disk technologies. Special CD-RW drives and software are required.

CD-RW disks are useful for people who need to store and work with great amounts of data. Examples are those who work in desktop publishing, with video and sound, or with database of photographs. Because they are relatively slow, they are no substitute for a hard disk. CD-RW disks commonly have a capacity of 650-700 megabytes.

CamScanner

DVD-ROM Disks

The acronym DVD-ROM (for *digital versatile disk–read-only memory*) refers to a high-capacity CD storage format that was initially developed to store the full contents of a standard three-hour movie. The acronym DVD originally stood for *digital video disk*. Proponents of the technology later felt that the word video was too limiting—the disks can also store text, graphics, and audio data—so the longer name is often dropped or, alternatively, the word versatile substituted for video.

Like a CD-ROM, the surface of a DVD contains microscopic pits, which represent the 0s and 1s of digital code that can be read by a laser. The pits on the DVD, however, are much smaller and grouped more closely together than those on a CD-ROM, allowing more information to be stored. DVD is designed to be backward compatible, so you can play your old CD-ROMs and music CDs on DVD peripheral devices.

A DVD can store 9.4 GB or more gigabytes—more than a dozen times what a conventional CD-ROM can carry-on a single side (Actually, DVD uses both sides of the disk). Therefore, these two-sided disks can contain as much as 18.8 GB of data. They support a faster data-transfer rate than CD-ROMs, as well. Most importantly, a single DVD has enough capacity to hold a three-hour movie—one with a full-screen picture and digital-TV quality, not the grainy quarter-screen video you see with most CD-ROMs today. The DVD standard includes support for Dolby surround sound and multichannel digital sound, multiple language tracks, multiple subtitle channels, copy protection, and parental lockout capabilities.

DVD-R Disks

DVD-R (DVD-recordable) disks allow onetime recording by the user. That is, they cannot be reused—written on more than once.

DVD-RW, DVD-RAM, DVD+RW Disks

Three types of reusable disks are DVD-RW (DVD-rewritable), DVD-RAM (DVD-random access memory), and DVD+RW (DVD+rewritable), all of which can be recorded on and erased (except for video) many times. DVD-R disks have a capacity of 4.7 (single-sided) to 9.4 (double-sided) gigabytes.

Blu-Ray Disks

Blu-ray (see Fig. 1.36), also known as Blu-ray Disk (BD), is an optical-disk format jointly developed by the Blu-ray Disk Association (BDA), a group of consumer electronics and several PC companies. The Blu-ray optical format was developed to enable recording, rewriting, and playback of high-definition video, as well as storing of large amounts of data.

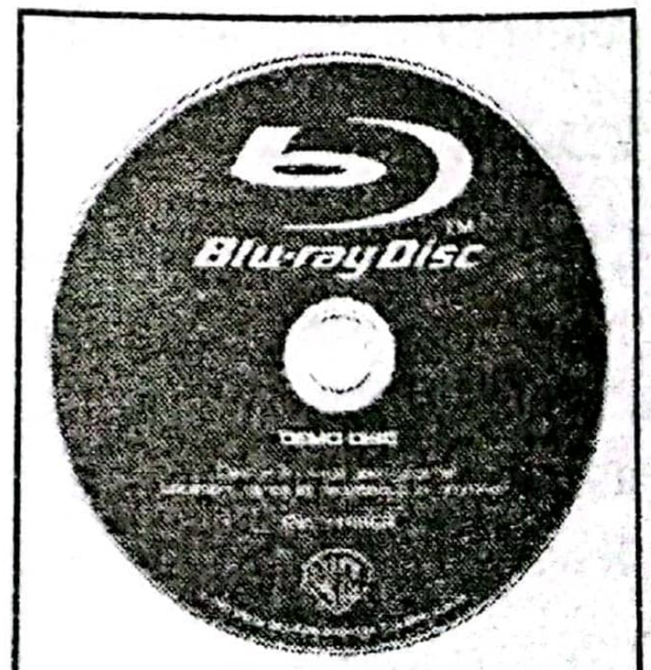


Fig. 1.36. A Blu-ray Disk

A single-layer Blu-ray Disk can hold 25 gigabytes, which can be used to record over 2 hours of HDTV (or more than 13 hours of standard-definition TV). There are also multi-layer versions of the disks. A dual layer disks, which can hold 50 gigabytes, is the industry standard for feature-length video disks. Triple-layer disks (100 GB) and quadruple-

CamScanner

layer disks (200/300 GB) are also available with re-writer drives.

While optical-disk technologies such as DVD, DVD-R, DVD-RW, and DVD-RAM use a red laser to read and write data, the Blu-ray Disk uses a blue-violet laser instead, hence the name Blu-ray. The benefit of using a blue-violet laser is that it has a shorter wavelength than a red laser, which makes it possible to focus the laser spot with even greater precision. This allows data to be packed more tightly and stored in less space, so it's possible to fit more data on the disk even though it's the same size as a CD/DVD.

Flash Storage—Solid State Drive, Memory Card, USB Drive

Flash storage, also known as flash memory, is a type of non-volatile memory that can be erased and rewritten electronically. Flash storage chips are solid state media, which means they consist entirely of electronic components and contain no moving parts. The lack of moving parts makes flash memory storage more durable and shock resistant than other types of media such as magnetic hard disks or optical disks.

Flash memory was invented by Fujio Masuoka while working for Toshiba during 1980s. According to Toshiba, the name "flash" was suggested by Masuoka's colleague, Shoji Ariizumi, because the erasure process of the memory contents reminded him of the flash of a camera.

Types of flash memory include solid state drives, memory cards, and USB flash drives.

Solid State Drives

A solid state drive (SSD) is a storage device that typically uses flash memory to store data (see Fig. 1.37). Instead of hard disk drives, SSDs are used in all types of newer computers including servers, desktop computers, and mobile computers and devices such as portable media players and digital video cameras. SSDs are lighter and use less power than magnetic disk drives and can better withstand bumps and bangs. As the price of SSDs drops, experts estimate that increasingly more users will purchase computers and devices that use this media.

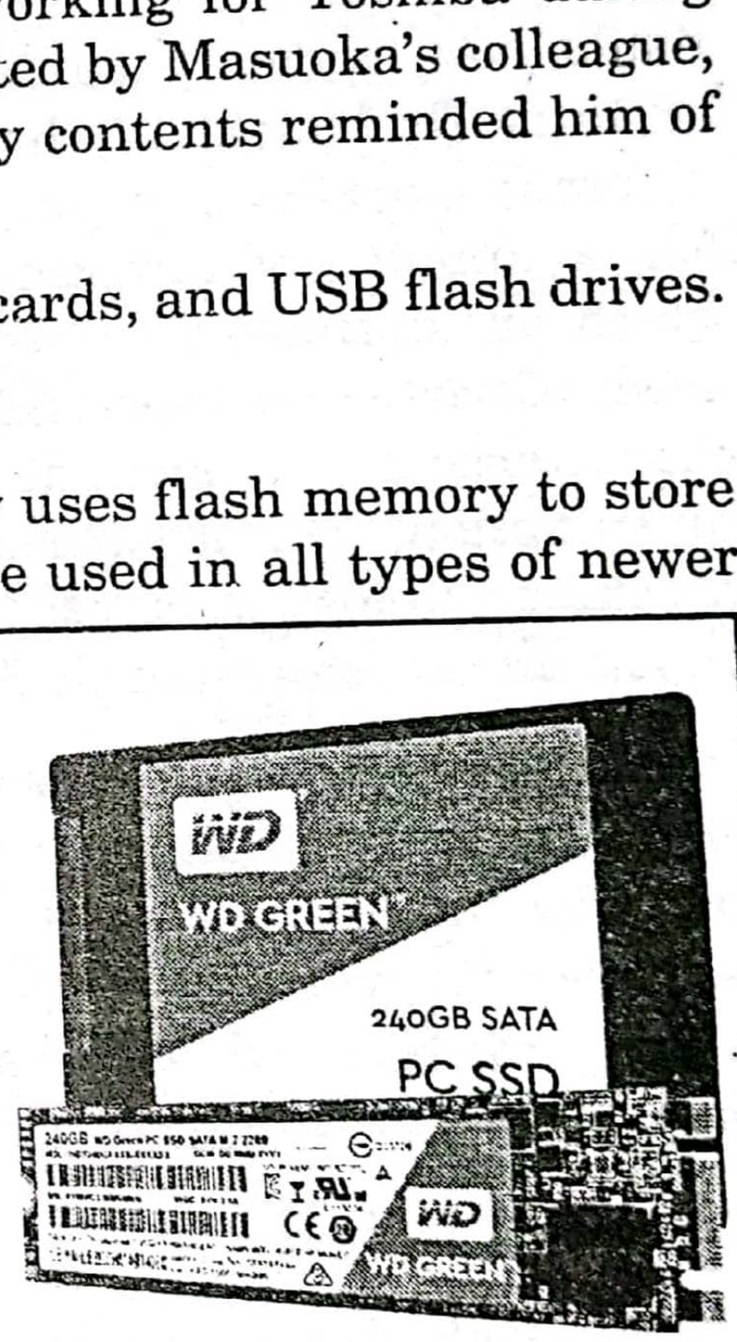


Fig. 1.37. A solid state drive

SSD storage continues to evolve and shrink in size, increasing the potential for storage density. The largest SSD hard drives currently available have a storage capacity of 30.72 TB with sequential read speeds of up to 2100 MB/s.

SSDs have several advantages over magnetic hard disks.

1. As there are no mechanical components to prepare, the start-up time of SSD is very less. May need a few milliseconds to come out of an automatic power-saving mode.
2. Random access time of SSD is typically under 0.1 ms, which is more than 80 times faster than a hard disk.
3. Transfer rates of SSDs are faster than comparable hard disks. For example, SSD

CamScanner

transfer rates typically range from about 500 MB/s to 600 MB/s, while transfer rates of a typical 2.5-inch hard disk is about 60 MB/s.

4. SSDs generate less heat and consume less power than magnetic hard disks.
5. Manufacturers claim that SSDs will last more than 50 years, which is much greater than the 3 to 5 years hard disk stated lifespan.

The disadvantages of SSDs are they currently have a higher failure rate than hard disks, and they are much more expensive than hard disk drives.

Memory Card

A memory card, or flash card, is an electronic flash memory data storage device used for storing digital information (see Fig. 1.38). These are commonly used in portable electronic devices, such as digital cameras, mobile phones, laptop computers, and tablets etc. Memory cards are quite expensive compared to other storage media with equivalent capacity. Depending on when the product was manufactured, the transfer rates of memory cards speed can range from 10 MB/s to 80 MB/s.

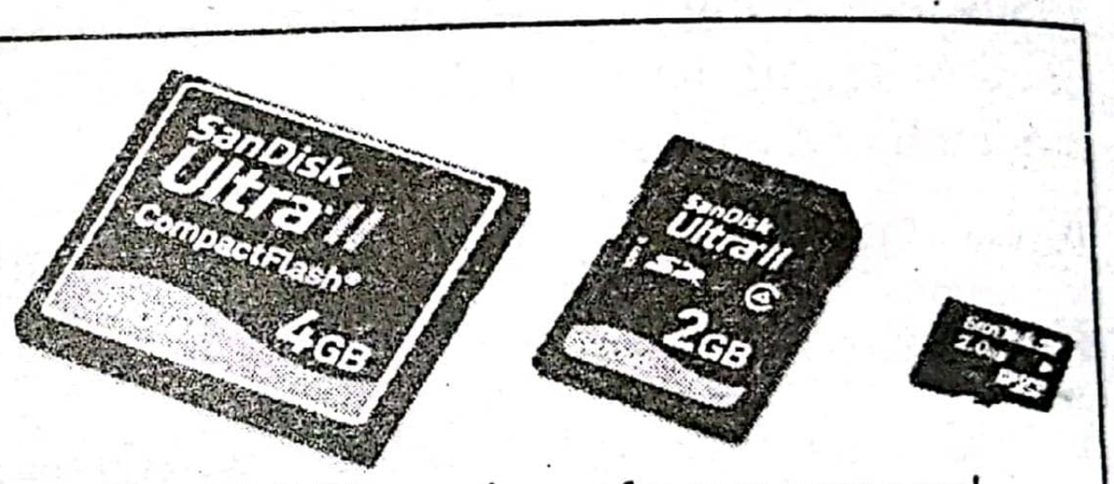


Fig. 1.38. Three sizes of a memory card

While both memory cards and most SSDs use flash memory, they serve very different markets and purposes. Each has a number of different attributes which are optimized to best meet the needs of particular users. Some of these characteristics include power consumption, performance, size, and reliability. Most memory cards are physically smaller than SSDs, and designed to be inserted and removed repeatedly. To view, edit, or print images and information stored on memory cards, you can use your personal computer. Some printers have slots to read memory cards. If your computer or printer does not have a built-in slot, you can purchase a card reader/writer, which is a device that reads and writes data and information stored on memory cards. Card reader/writers usually connect to the USB port on a computer. The type of card you have will determine the type of card reader/writer needed.

USB Flash Drive

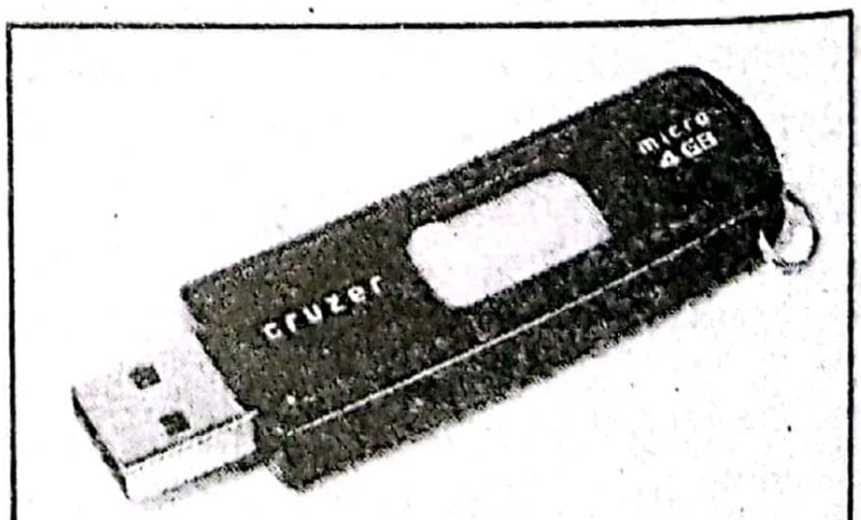


Fig. 1.39. USB Flash Drive

A USB flash drive, sometimes called a thumb drive, is a flash memory storage device that plugs in a USB port on a computer or mobile device (see Fig. 1.39). It is typically removable, rewritable and much smaller than an optical disc. USB flash drives are convenient for mobile users because they are small and lightweight enough to be transported in a pocket. With a USB flash drive, users easily transfer documents, photos, music, and videos from one computer to another. When you plug the device into your USB port, it shows up as an external drive on the computer screen. As of September 2023, flash drives with anywhere from 16 to 256 GB are frequently sold; less frequent are

CamScanner

512 GB and 1 TB units. USB flash drives generally last between 10 and 100 years under normal circumstances. Some USB flash drives include fingerprint readers, restricting access to authenticated users.

USB 3.2 (Gen 2) flash drives reached 1,000 MB/s read speed. Most USB flash drives are unable to read at such high speeds. Practically, current USB flash drives have data read speed of about 480 MB/s, depending on the exact type of memory chip used.

There are three factors that effect efficiency and speed in USB flash drives:

1. **Type and Age:** As they get older, they tend to lose swiftness. The operating components become weaker, perhaps covered in dust, and less responsive than when they were brand new. Also, the capabilities of USB flash drives depend on their type (USB 3.1 or 3.2).
2. **Type and Size of Files:** Small files like MP3, Word documents and lower resolution images usually transfer faster than larger files like videos and high-resolution images.
3. **USB Port Version:** With a newer port version, like USB Port 3.2, data will transfer faster.

Cloud Storage—OneDrive, Google Drive, iCloud

A user may choose cloud storage instead of storing data locally on a hard disk or other media. Cloud Storage is an online file storage web service for storing and accessing data (Fig. 1.40). The physical storage spans multiple servers (sometimes in multiple locations), and the physical environment is typically owned and managed by a hosting company. These cloud storage providers are responsible for keeping the data available and accessible, and the physical environment protected and running. The service combines the performance and scalability with advanced security and sharing capabilities. People and organizations buy or lease storage capacity from the providers to store user, organization, or application data.

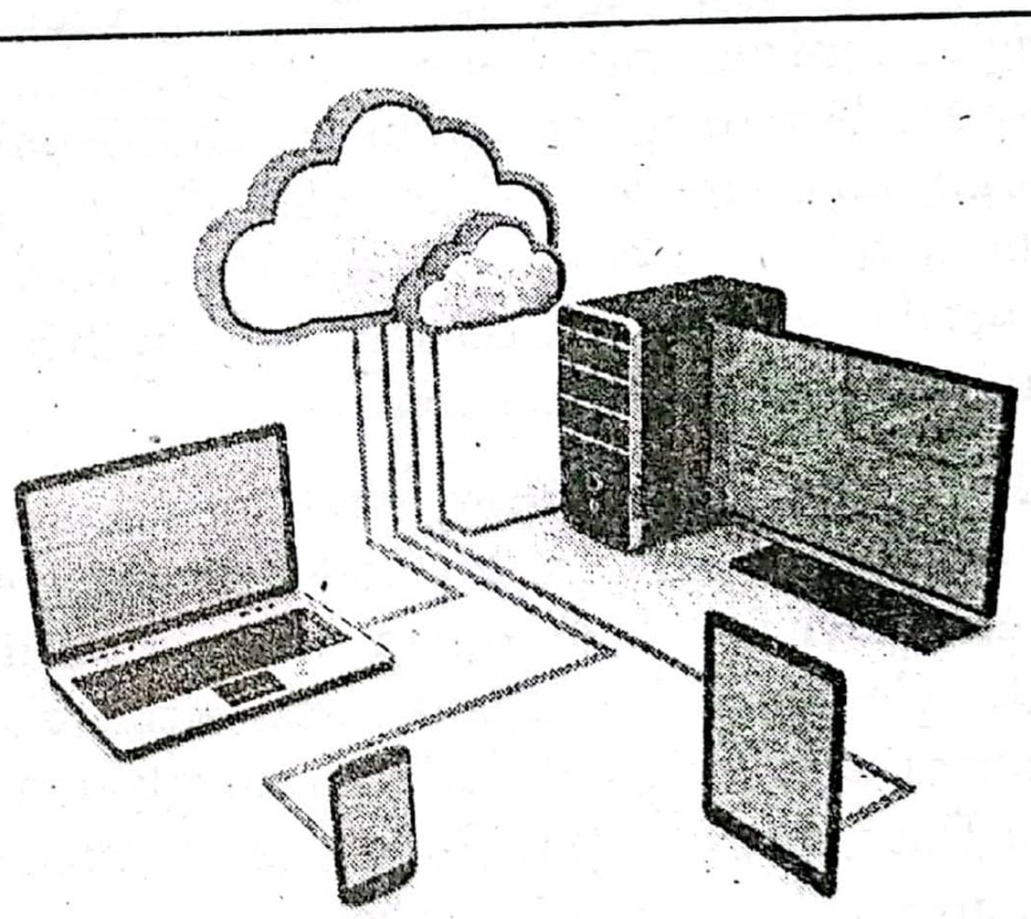


Fig. 1.40. An example of services provided by cloud storage

Cloud storage services may be accessed through a cloud computing service, a web service application programming interface (API) or by applications that utilize the API, such as cloud desktop storage, a cloud storage gateway or Web-based content management systems.

Types of services offered by cloud storage providers vary. Some provide storage for specific types of files, such as photos or email messages, whereas others store any type of file. Many cloud storage providers offer additional services such as encryption,

CamScanner

passwords, Web applications, and Web services. Table 1.2 identifies a variety of cloud storage providers.

Table 1.2. Some of the more widely used cloud storage providers

Web Site Names	Type of Storage Provided
Dropbox, iCloud, Google Drive, OneDrive	Backup or additional storage for any type of file
Google Photos, Flickr, Photobucket	Digital photos, photo editing and photo management
YouTube	Digital videos
Facebook, Twitter	Digital photos, digital videos, messages, and personal information. Social networking
Google Docs, Microsoft Office Online	Documents, spreadsheets, presentations
Gmail, Outlook, Yahoo! Mail	Email messages

Cloud storage is available for all sizes of users, with various degrees of storage services available for home and business users. Cloud storage fee vary, depending on the user's storage requirements. For example, Microsoft OneDrive storage service provides 5 GB of storage free to registered users; additional storage can be added either separately or through subscription of Microsoft 365. It allows users to store files as well as other personal data like Windows settings or BitLocker recovery keys in the cloud. Files can be synced to a PC and accessed from a web browser or a mobile device, as well as shared publicly or with specific people.

Google Drive is another example of cloud storage. It is a file storage and synchronization service developed by Google. Google Drive allows users to store files on their servers, synchronize files across devices, and share files. In addition to a website, Google Drive offers apps with offline capabilities for Windows and macOS computers, and Android and iOS smartphones and tablets. Google Drive encompasses Google Docs, Sheets and Slides, an office suite that permits collaborative editing of documents, spreadsheets, presentations, drawings, forms, and more. Files created and edited through the office suite are saved in Google Drive. Google Drive offers users 15 gigabytes of free storage, with additional storage through optional paid plans.

Users subscribe to cloud storage for a variety of reasons:

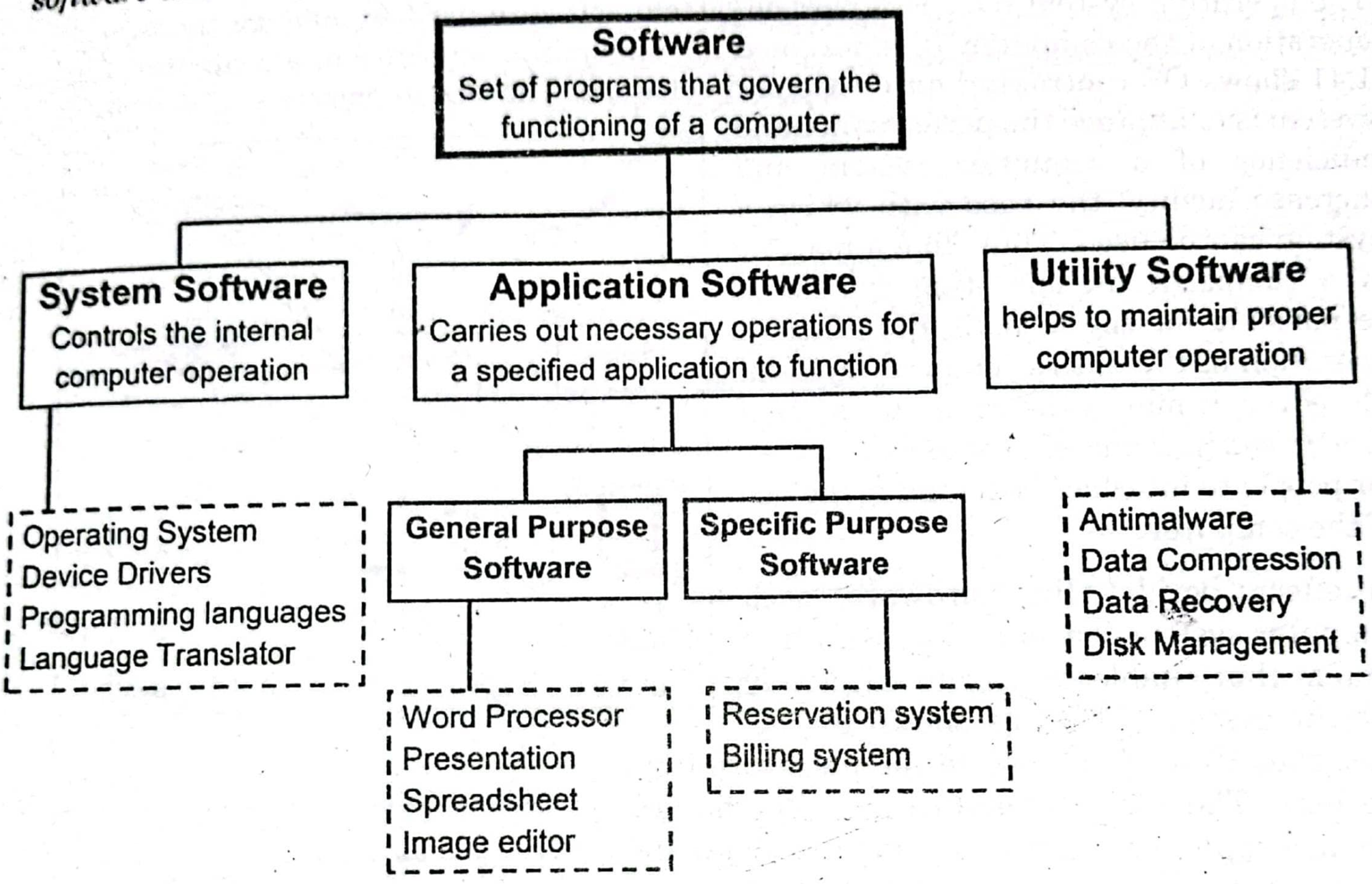
1. To access files on the Internet from any computer or device that has Internet access.
2. To store large audio, video, and graphics files on the Internet instantaneously, instead of spending time downloading to a local hard disk or other media.
3. To allow others to access their files on the Internet so that others can listen to an audio file, watch a video clip, or view a photo—instead of emailing the file to them.
4. To view time-critical data and images immediately while away from the main office or location.
5. To store offsite backups of data.

CamScanner

Software

The instructions that tell a computer what to do is called a *software*. It is the opposite of hardware, which describes the physical aspects of a computer. Software comprises the entire set of applications, scripts and programs associated with the operation of a computer system.

Software can be broadly classified into three types; *system software, application software* and *utility software*. The types are illustrated below:



System software controls a computer's internal functioning, mainly through an operating system, and handles the more basic operations, such as hardware interfaces, disk storage, and integration with the application software. It ensures that the technology tools involved in the information system all work together smoothly, though most of the time you need pay no attention to its activities. It also controls the order in which jobs are undertake, and many other activities that let the computer efficiently process a series of tasks. Three principal parts of system software are:

1. Operating system,
2. Device drivers, and
3. Language translator.

Application software, by contrast, supports all the work (or play) you want to do on a PC. The term *application software* comes from the fact that such software *applies* the computer to real-world user tasks. General purpose software and specific purpose software which are the categories of application software can be found on personal computers.

CamScanner

General purpose application software includes word processors, spreadsheets, database management systems, video editing, email, web browsing, game playing, and many other applications. On the other hand, specific purpose software are designed to perform specific tasks. This type of application software generally has one purpose to execute. Common examples of specific purpose software are inventory management system and purchasing system, payroll management system, hotel management system, reservation system, report card generator, and billing system.

The Operating System

The operating system (OS) is a part of system software used to manage the overall operation of the computer. It is designed to support the activities of a computer. Fig. 1.41 shows OS controlling hardware and software. The prime objective of operating system is to improve the performance and efficiency of a computer system and increase facility, the ease with which a system can be used. Thus, like a manger of a company, an operating system is responsible for the smooth and efficient operation of the entire computer system. Moreover, it makes the computer system *user friendly*. That is, it makes it easier for people to interface with and make use of the computer.

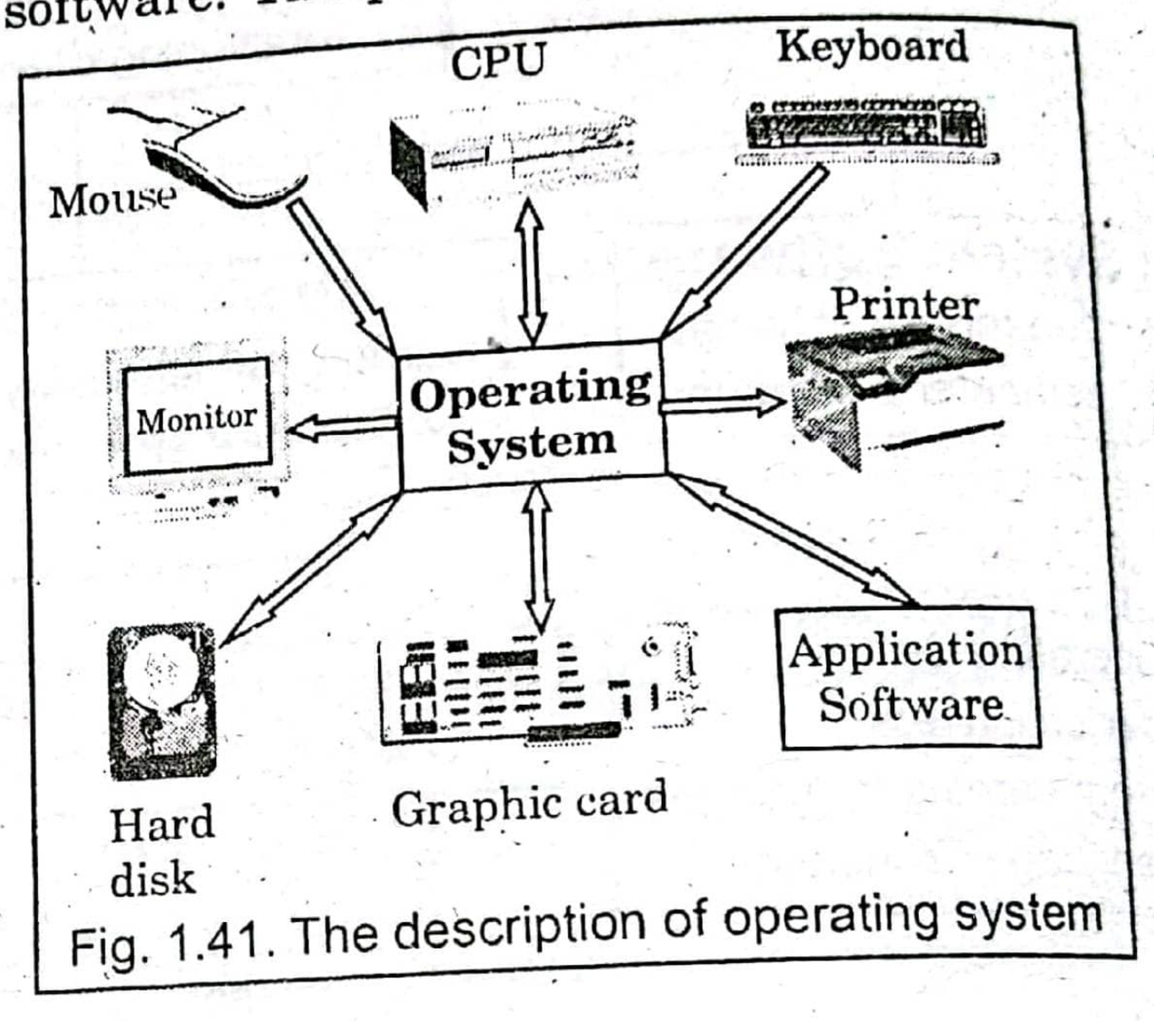


Fig. 1.41. The description of operating system

In effect, besides the hardware, each computer system consists of an operating system that enables a user to effectively use the system. Thus, as shown in Fig. 1.42, the OS tends to isolate the hardware from the user. The user communicates with the OS, supplies application software and input data, and receives output results. However, it is interesting to known that all the tasks performed by the OS are performed automatically. The functions of the OS are performed automatically. The functions of the OS are transparent to the user—he/she really is not at all concerned with what the OS is doing or how the OS directs the hardware to handle certain tasks.

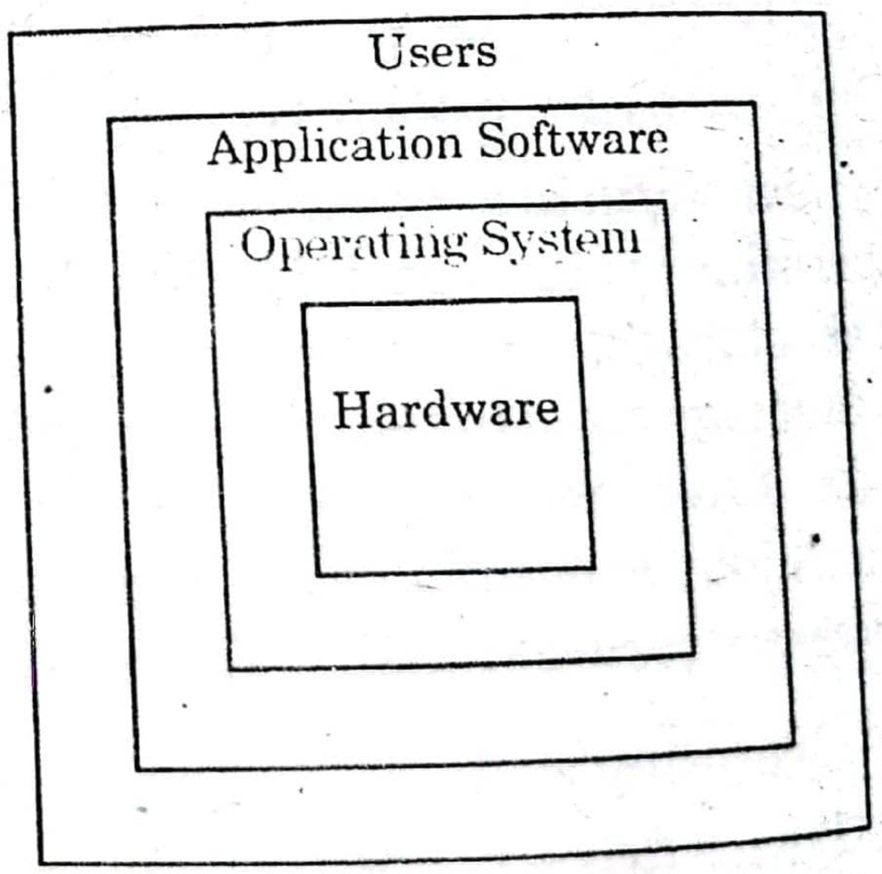


Fig. 1.42. An overview of a system showing the position of the Operating System

Examples of operating system (OS) software for the PC include various Microsoft OS such as MS-DOS (Microsoft Disk Operating System), Windows 10, Windows 11, Unix, Linux, macOS, and Chrome OS. The Macintosh uses macOS.

Device Drivers

Device drivers are specialized software programs that allow a piece of hardware to

CamScanner

communicate with the OS running your computer. Each device's brand and model are supported by a different driver that works with only one operating system. Many basic device drivers come with (are preinstalled in) the system software when you buy a computer, and the system software will guide you through choosing and installing the necessary drivers. If, however, you buy a new peripheral device, such as a mouse, a scanner, or a printer, the drivers are usually packaged with the hardware or available for download over the Internet.

Most new operating systems recognize many new hardware devices on their own and automatically install them. If your OS does not recognize your new hardware, it will display a message and ask you to install the driver.

Language Translators

Computer cannot execute programming instructions of a programming language. They work electronically and, therefore, must represent data and instructions in binary (1s and 0s) form. Language Translators are programs those converts or translates programming language (high level language) into 1s and 0s, or machine language (low level language). Language translators use two methods to translate high level language. They either compile or interpret it. These methods give language translators their more common names of *compilers* and *interpreters*.

Compiler: A compiler is a program that translates a high-level programming language into machine language. It translates whole high level language program at once into machine language before it is executed.

A program written by a programmer in a high-level language is called a *source program* or *source code*. After this source program has been converted into machine language by a compiler, it is referred to as an *executable* or *executable file*. The executable is saved as a program file ready for execution at any time desired. As shown in Fig. 1.43 the input to a compiler (program) is a source program written in a high-level language and its output is an executable file which consists of machine language instructions. Note that the source program and the executable file are the same program, but at different stages of development.

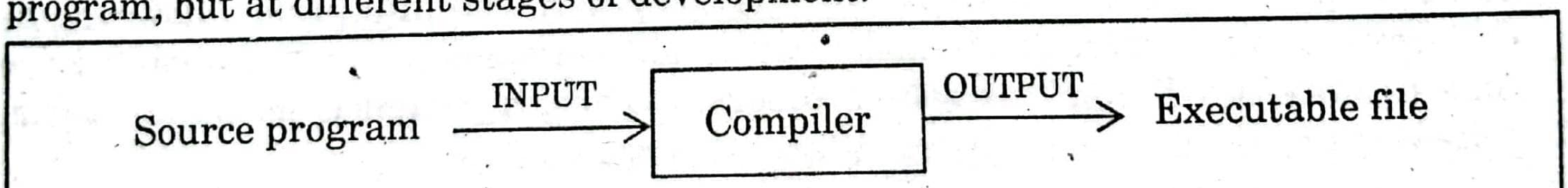


Fig. 1.43. Illustrating the translation process of a compiler

A compiler can translate only those source programs which have been written in the language for which the computer is meant. For example, a C++ compiler is only capable of translating source programs which have been written in C++ and, therefore, computer requires a separate compiler for each high-level language.

While translating a given program, the compiler analyses each statement in the source progra[illegible]nd generates a sequence or machine instructions which, when executed, will precisely carry out the computation specified in the statement. As the compiler analyses each statement, it uncovers certain types of errors. These are referred to as *diagnostic errors*. The errors listed are *syntax* (or "language") *errors*. The compiler can

CamScanner

diagnose the following kinds of errors in a source program.

1. Illegal characters.
2. Illegal combinations of characters.
3. Improper sequencing of instructions in a program.

A source program containing an error diagnosed by the compiler will not be compiled into an executable file. The compiler will print out a suitable message indicating this, along with a list of error message which indicate the type of errors committed. The error diagnostics is an invaluable aid to the programmer.

Interpreter: An interpreter is a program that translates a high-level programming language into machine language during the actual step-by-step execution of a program. Translation and execution alternate for each statement encountered in the high-level language program. In other words, an interpreter translates one instruction, and the control unit executes the resulting machine language, the next instruction is translated, and the control unit executes the machine language instruction, and so on. If the program is run seven times a day, the programming language is reinterpreted seven times.

In case of a compiler, the whole source program is translated into an equivalent machine language program. The executable file, thus obtained, is permanently saved for future use and is used every time the program is to be executed. So repeated compilation is not necessary for repeated execution of a program. However, in case of an interpreter, no executable file is saved for future use because the translation and the execution processes alternate. The next time an instruction is used, it must once again be interpreted and translated into machine language. For example, during the repetitive processing of the instructions within a loop, each instruction in the loop will have to be reinterpreted every time the loop is executed.

The advantage of an interpreter over a compiler is fast response to changes in the source program. The interpreter eliminates the need for a separate compiling run after each program change to add features or correct errors. Moreover, a compiler is a complex program compared to an interpreter. Interpreters are easy to write and they do not require large memory space in the computer. The interpreter, however, is a time-consuming translation method because each statement must be translated every time it is executed from the source program. Thus, a compiled machine language program runs much faster than an interpreted program.

Utility Software

The *utility software* helps to maintain the proper and smooth functioning of a computer system. It assists the operating system to manage, organize, maintain, and optimize the functioning of the computer system. The best utility software provides key functionality such as data compression, data recovery, disk defragmentation, computer resources and files management, system diagnosis, and more.

Most of the operating systems have built-in utility tools, but additional software like antivirus tools, disk tools and data backup tools further ensure improved functionality. Utility software focuses on computer components such as hardware, software,

CamScanner

operating system, and storage parts. Some of the important utility programs are as follows:

Antimalware: Malware (or malicious software) is any software intentionally designed to cause damage to a computer, server or computer network. Malware does the damage after it is introduced or implanted in some way into a target's computer and can take the form of executable code, active content, and other software. Malware has a malicious intent, acting against the interest of the computer user.

One strategy for protecting against malware is to prevent the malware software from gaining access to the target computer. For this reason, *antivirus software, firewalls* and other strategies are used to help protect against the introduction of malware, in addition to checking for the presence of malware and malicious activity and recovering from attacks. *Antivirus software* is a utility program that scans your computer and notifies you of possible viral behavior or destroys the virus on the spot. Because new viruses are constantly being created, you need the type of antivirus software that can constantly update itself against newly discovered viruses.

Malware can be divided into several categories. These include computer viruses, worms, Trojan horses, ransomware, spyware, adware, and scareware, among other terms.

Data Compression: With the increasing use of large gråphic, sound, and video files, data compression has become an increasingly important part of computing, for both to reduce the storage space required on your hard-disk drive and to reduce the time required to transmit such large files across the Internet. Data compression utilities encode information using fewer bits than the original representation.

Any particular compression is either lossless (exact) or lossy (inexact). Lossless compression can be reversed to yield the original data, while lossy compression loses detail or introduces small errors upon reversal. Lossless compression is necessary for text, where every character is important, while lossy compression may be acceptable for videos, images or voice.

Most operating systems include tools for compressing and uncompressing files. For example, latest versions of Microsoft Windows include a compression utility for creating and extracting .zip files and Linux has tools for both .tar.gz and .zip. Examples of a third-party compression tools are 7-Zip, WinZip, and WinRAR. (Inspired by the name "Zip," people now refer to compressing a file as "zipping" and decompressing a file as "unzipping.)

Data Recovery: A data-recovery utility is used to restore data that has been deleted, inaccessible, lost, corrupted, damaged, or formatted data from secondary storage or removable media. Data can be damaged by viruses, bad software, hardware failure, and power fluctuations that occur while data is being written/recorded.

The data recovery process varies based on the circumstances under which it was lost. Data recovery software can be used to retrieve files that were not backed up and accidentally deleted from a computer's file system, but still remain on the hard disk in fragments. Data recovery is possible because a file and the information about that file are stored in different places. For example, the Windows operating system uses a file

CamScanner

allocation table to track which files are on the hard drive and where they are stored. The allocation table is like a book's table of contents, while the actual files on the hard drive are like the pages in the book. When data needs to be recovered, it is usually only the file allocation table that is not working properly. The actual file to be recovered may still be on the hard drive in flawless condition. If the file still exists—and it is not damaged or encrypted—it can be recovered. If the file is damaged, missing or encrypted, there are other ways of recovering it. If the file is physically damaged, it can still be reconstructed.

Disk Management: Disk Management is a utility used to manage disks and data on disks. Disk Management is a Microsoft Windows utility. It enables users to view and manage any internal and external disk drives installed in their computer and the partitions associated with those drives. While in Disk Management, you can format and reformat hard drives, create partitions, and view various information such as the status, capacity, and percentage of free space available. You also can rename hard drives, assign them specific disk letters, and change the size of partitions on that drive.

Fig. 1.44 shows Disk Management utility included in Microsoft Windows 11. Each drive is displayed followed by the layout, type, file system, status, capacity, free space, and % free.

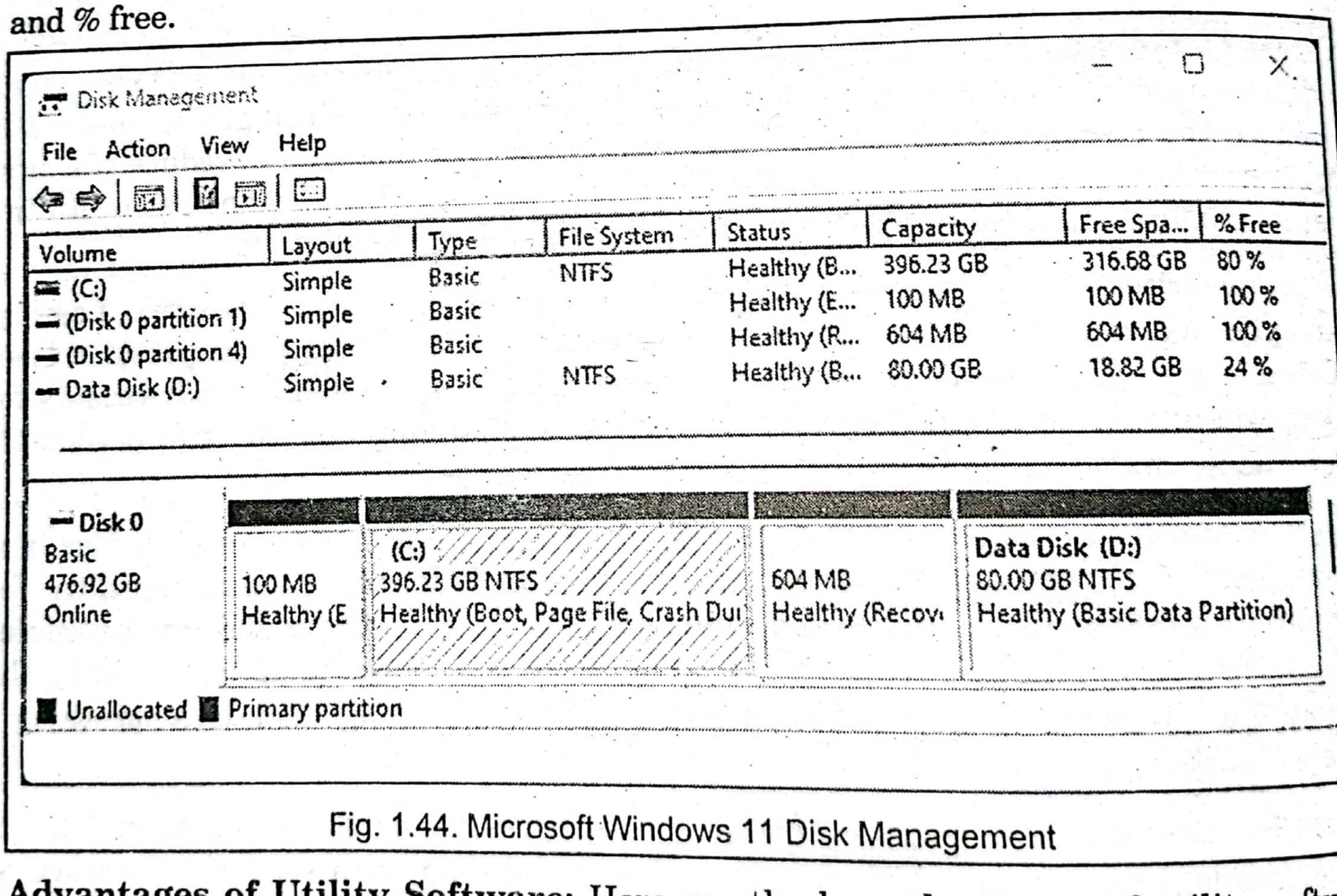


Fig. 1.44. Microsoft Windows 11 Disk Management

Advantages of Utility Software: Here are the key advantages of utility software that help a computing device run smoothly and efficiently:

1. Virus scanners and disk cleaners to get rid of unwanted files.
2. Fast scanning with advanced privacy controls.
3. Real-time device monitoring for keeping a tab on online threats.
4. Backup utilities with file compression for retrieving the lost data.
5. Quick file and registry clean-ups for optimizing a computer's performance.

CamScanner

6. System repair for fixing problems.
7. Disk defragmentation helps correct errors to be found in hard disk drives.
8. System status and profile checks for protecting a computer system's status.
9. Inbuilt device performance advisor with a modern interface.

Introduction to Artificial Intelligence

Artificial intelligence is a subfield of computer science that studies the thought processes of humans and recreates the effects of those processes through information systems. We define artificial intelligence (AI) the simulation of human intelligence processes by machines, especially computer systems. Specific applications of AI include expert systems, natural language processing, speech recognition and machine vision (see Fig. 1.45).

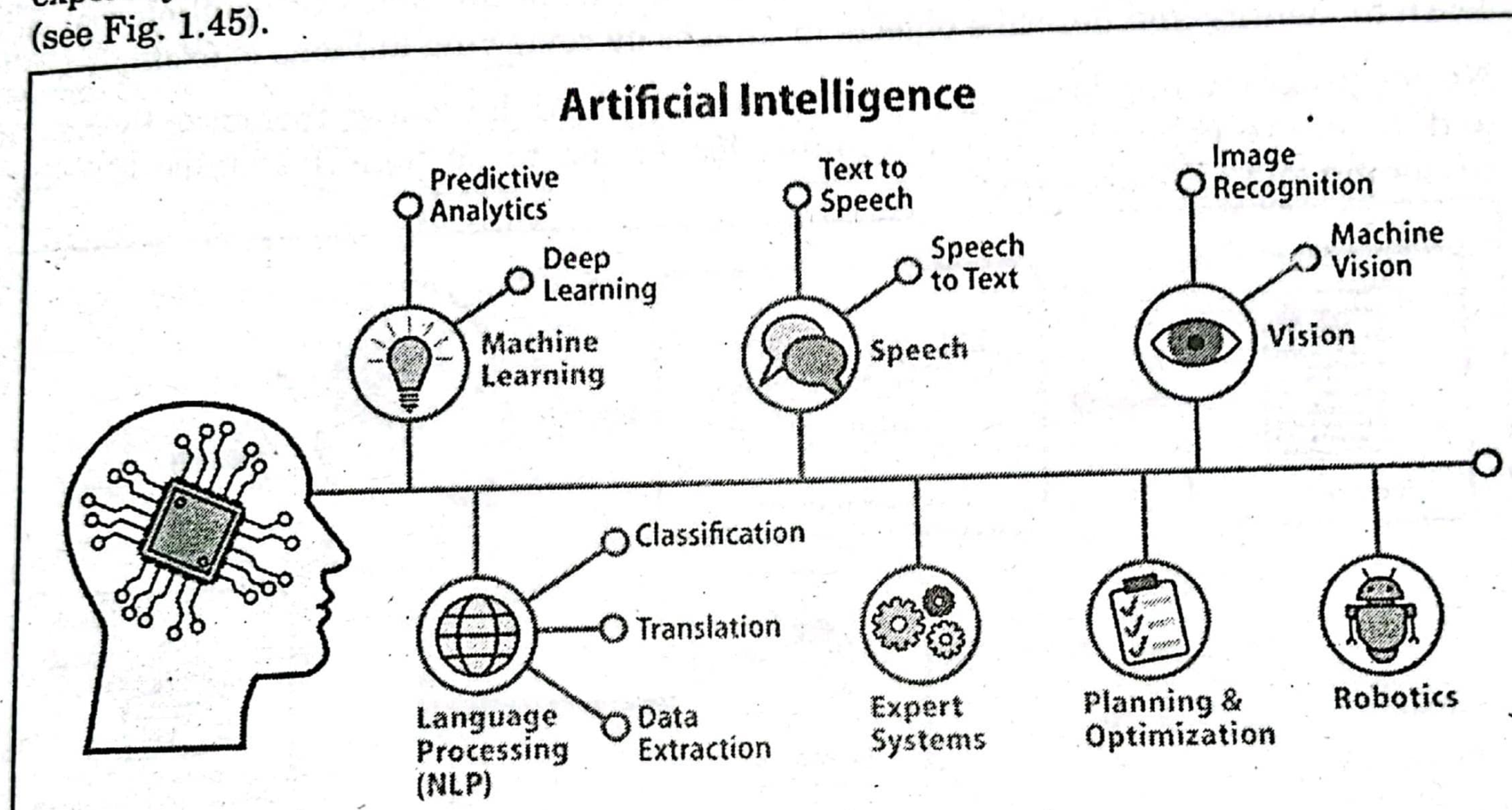


Fig. 1.45. Artificial Intelligence

Several technological advancements have led to advancements in artificial intelligence. We take a brief look at each of them here:

1. **Advancements in chip technology:** An AI chip, also known as an artificial intelligence chip or AI accelerator, is a specialized semiconductor component designed to perform high-speed computations for artificial intelligence tasks. These chips are optimized to efficiently handle the complex and data-intensive computations required in machine learning and deep learning algorithms.
2. **Big Data:** Big Data consists of diverse, high-volume, high-velocity information assets that require new types of processing to enable enhanced decision making, insight discovery, and process optimization. Big Data is now being used to train deep learning software.
3. **The Internet and cloud computing:** The Internet and cloud computing make Big Data available to AI systems, specifically neural networks, and provide the computational capacity needed for AI systems.

CamScanner

4. **Improved AI algorithms:** AI algorithms refer to a set of mathematical and computational instructions that enable machines, particularly artificial intelligence systems, to process data, learn from it, and make decisions or predictions based on patterns and patterns within the data. These algorithms are the backbone of AI systems, allowing them to perform tasks that typically require human intelligence, such as recognizing images, understanding natural language, playing games, driving cars, and more.

How does AI work?

In general, AI systems work by storing large amounts of training data, analyzing the data for correlations and patterns, and using these patterns to make predictions about future states (see Fig. 1.46). In this way, a *chatbot* that is fed examples of text chats can learn to produce lifelike exchanges with people, or an image recognition tool can learn to identify and describe objects in images by reviewing millions of examples.

No one programming language is synonymous with AI, but a few, including Python, R and Java, are popular. AI programming focuses on three cognitive skills: learning, reasoning and self-correction.

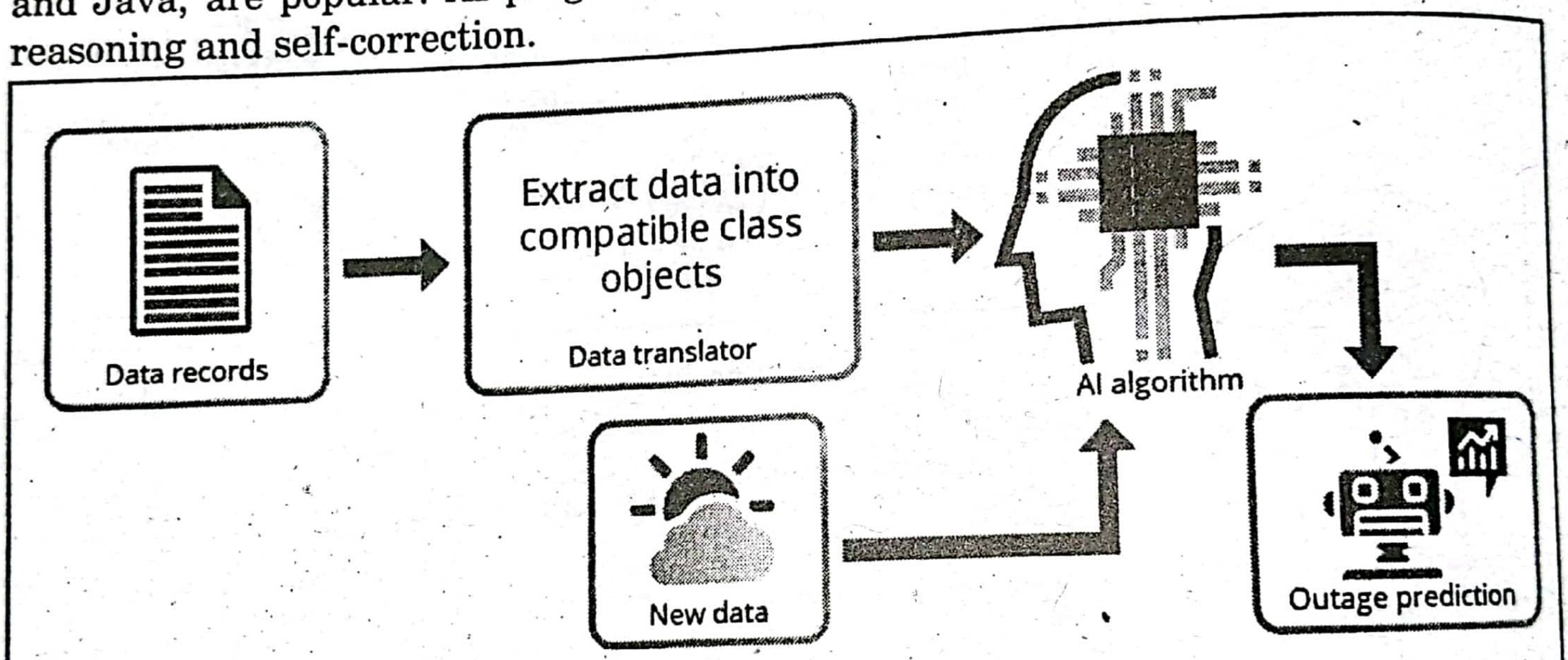


Fig. 1.46. How does Artificial Intelligence work?

Learning processes: This aspect of AI programming focuses on acquiring data and creating rules for how to turn the data into actionable information. The rules, which are called *algorithms*, provide computing devices with step-by-step instructions for how to complete a specific task.

Reasoning processes: This aspect of AI programming focuses on choosing the right algorithm to reach a desired outcome.

Self-correction processes: This aspect of AI programming is designed to continually fine-tune algorithms and ensure they provide the most accurate results possible.

Importance of Artificial Intelligence

AI is important because it can give enterprises insights into their operations that they may not have been aware of previously and because, in some cases, AI can perform tasks better than humans. Particularly when it comes to repetitive, detail-oriented tasks like analyzing large numbers of legal documents to ensure relevant fields are

CamScanner

filled in properly, AI tools often complete jobs quickly and with relatively few errors.

This has helped increase in efficiency and opened the door to entirely new business opportunities for some larger enterprises. Prior to the current wave of AI, it would have been hard to imagine using computer software to connect riders to taxis, but today Uber has become one of the largest companies in the world by doing just that. It utilizes sophisticated machine learning algorithms to analyze real-time demand and supply data, the system adjusts fares, ensuring the availability of drivers during peak hours and incentivizing them to meet high demand. AI assists Uber drivers by recommending the most efficient routes based on traffic conditions and other relevant factors. This leads to shorter travel times, lower fuel consumption, and better overall efficiency. As another example, ChatGPT, based on OpenAI's GPT technology, is an exemplary AI language model. It utilizes natural language processing (NLP) algorithms to comprehend and generate human-like text. The impact of ChatGPT is evident in various applications:

1. Content Creation: Content creators can leverage ChatGPT to generate high-quality articles, blogs, and social media posts. It assists in generating creative ideas, improving writing efficiency, and tailoring content for specific audiences.
2. Customer Support: Companies integrate ChatGPT-powered chatbots into their websites to provide instant customer support. These chatbots handle customer inquiries 24/7, reducing response times and enhancing user experiences.
3. Language Translation: ChatGPT enables real-time language translation, breaking down language barriers and promoting global communication and understanding.

Today's largest and most successful enterprises have used AI to improve their operations and gain advantage on their competitors.

Advantages and Disadvantages of Artificial Intelligence

Artificial neural networks (a form of artificial intelligence that mimics the human brain to find relationships in data.) and deep learning (a machine learning technique that teaches computers to do what comes naturally to humans) artificial intelligence technologies are quickly evolving, primarily because AI processes large amounts of data much faster and makes predictions more accurately than humanly possible.

While the huge volume of data being created on a daily basis would bury a human researcher, AI applications that use machine learning can take that data and quickly turn it into actionable information.

<u>Advantages:</u>

1. AI is good at detail-oriented jobs;
2. It drives down the time taken to perform a task. It enables multi-tasking and eases the workload for existing resources.
3. Enables the execution of complex tasks without significant cost outlays.
4. AI operates 24x7 without interruption or breaks and has no downtime.
5. Augments the capabilities of differently abled individuals.
6. AI has mass market potential; it can be deployed across industries.
7. Facilitates decision-making by making the process faster and smarter.

CamScanner

Disadvantages:

1. The primary disadvantage of using AI is that it is expensive to process the large amounts of data that AI programming requires.
2. Requires deep technical expertise.
3. Limited supply of qualified workers to build AI tools.
4. Only knows what it's been shown; and
5. Lack of ability to generalize from one task to another.

Applications of AI Technology

AI applications have a significant impact on various fields and industries, transforming the way we work, live, and interact. Here are some AI applications in different domains:

1. **Healthcare:**
   (a) *Medical Imaging Analysis:* AI systems can analyze medical images, such as X-rays and MRIs, to assist doctors in diagnosing diseases and conditions more accurately and efficiently.
   (b) *Personalized Medicine:* AI algorithms analyze patient data to develop personalized treatment plans based on individual characteristics and medical history.
   (c) *Drug Discovery:* AI accelerates drug development by analyzing molecular interactions and predicting potential drug candidates.
2. **Finance:**
   (a) *Fraud Detection:* AI technology can identify fraudulent transactions by analyzing patterns and anomalies in financial data, enhancing security for banking and e-commerce transactions.
   (b) *Credit Risk Assessment:* AI evaluates creditworthiness by analyzing an individual's financial history and behavior.
   (c) *Algorithmic Trading:* AI-driven algorithms analyze market data to make automated trading decisions, increasing efficiency and responsiveness in financial markets.
3. **Autonomous Vehicles:**
   (a) *Self-Driving Cars:* AI technologies, such as computer vision and machine learning, enable self-driving cars to perceive the environment, navigate roads, and make driving decisions without human intervention.
   (b) *Drones:* AI-powered drones can be used in various applications, including aerial photography, surveillance, search and rescue operations, and delivery services.
4. **Natural Language Processing (NLP):**
   (a) *Virtual Assistants:* Examples include Apple's Siri, Amazon's Alexa, and Google Assistant, which use NLP to understand and respond to user commands and queries.
   (b) *Language Translation:* AI-powered language translation services, like Google Translate, use NLP to translate text and speech between different languages.

CamScanner

(c) *Sentiment Analysis:* AI algorithms analyze text data to determine the sentiment or emotions expressed in social media posts, customer reviews, and feedback.

5. Image and Video Recognition:

(a) *Facial Recognition:* AI-powered facial recognition systems, like those used in smartphones and security systems, can identify and authenticate individuals based on their facial features.

(b) *Object Detection:* AI algorithms can identify and locate specific objects within images and videos, used in applications like autonomous vehicles, surveillance, and medical imaging.

(c) *Image Captioning:* AI can generate descriptive captions for images, improving accessibility and content understanding.

6. Manufacturing:

(a) *Predictive Maintenance:* AI monitors equipment data to predict maintenance needs, reducing downtime and optimizing production efficiency.

(b) *Quality Control:* AI-powered systems inspect products in real-time, ensuring consistent quality and detecting defects.

7. Education:

(a) *Intelligent Tutoring:* AI-based tutoring systems personalize learning experiences for students, adapting the curriculum to individual strengths and weaknesses.

(b) *Automated Grading:* AI can assess and grade assignments, quizzes, and exams, streamlining the evaluation process for educators.

8. Social Media, Advertising and Content Recommendation:

(a) *Targeted Advertising:* AI analyzes user behavior and preferences to deliver personalized ads, increasing ad relevance and conversion rates.

(b) *Content Recommendation:* AI algorithms analyze user preferences and behaviors to recommend personalized content on social media platforms and streaming services like Netflix and YouTube.

9. Gaming:

(a) *AI Opponents:* In video games, AI is used to create computer-controlled opponents with varying levels of difficulty, providing challenging gameplay experiences for players.

10. Agriculture:

(a) *Precision Farming:* AI technologies monitor and analyze data from sensors and drones to optimize irrigation, fertilization, and crop health.

(b) *Crop Disease Detection:* AI-powered image analysis identifies crop diseases and pests, enabling timely intervention and reducing losses.

11. Human Resources:

(a) *Talent Acquisition:* AI screens job applications, shortlisting candidates based on qualifications and experience.

(b) *Employee Engagement:* AI-powered tools gauge employee sentiment and engagement, helping organizations improve workplace dynamics.

CamScanner

Cloud Computing

Another emerging technology trend that computer users should be aware of is cloud computing. Cloud computing is a style of computing in which dynamically scalable and often virtualized resources are provided over the Internet. Users need not have knowledge of, experience in, or control over the technology infrastructures in the cloud that supports them. These services are divided into three main categories or types of cloud computing: *infrastructure as a service* (IaaS), *platform as a service* (PaaS) and *software as a service* (SaaS).

A cloud can be private or public. A *public cloud* sells services to anyone on the internet. A *private cloud* is a proprietary network or a data center that supplies hosted services to a limited number of people, with certain access and permissions settings. Private or public, the goal of cloud computing is to provide easy, scalable access to computing resources and IT services.

Cloud infrastructure involves the hardware and software components required for proper implementation of a cloud computing model. Cloud computing can also be thought of as on-demand computing.

The name *cloud computing* was inspired by the cloud symbol that is often used to represent the internet in flowcharts and diagrams.

Types of Cloud Computing Services

Cloud computing can be divided into three general service delivery categories or forms of cloud computing:

1. **IaaS (infrastructure as a service):** IaaS providers, such as Amazon Web Services (AWS), supply a virtual server instance and storage, as well as application programming interfaces (APIs) that let users migrate workloads to a virtual machine (VM). Users have an allocated storage capacity and can start, stop, access and configure the VM and storage as desired. IaaS providers offer small, medium, large, extra-large, and memory- or compute-optimized instances, in addition to enabling customization of instances, for various workload needs. The IaaS cloud model is closest to a remote data center for business users.

2. **PaaS (platform as a service):** In the PaaS model, cloud providers host development tools on their infrastructures. Users access these tools over the internet using APIs, web portals or gateway software. PaaS is used for general software development, and many PaaS providers host the software after it is developed. Common PaaS products include Salesforce's Lightning Platform, AWS Elastic Beanstalk and Google App Engine.

3. **SaaS (software as a service):** SaaS is a distribution model that delivers software applications over the internet; these applications are often called *web services*. Users can access SaaS applications and services from any location using a computer or mobile device that has internet access. In the SaaS model, users gain access to application software and databases. One common example of a SaaS application is Microsoft 365 for productivity and email services.

CamScanner

Table 1.3. Cloud Computing Service Categories

	SaaS Software as a service	PaaS Platform as a service	IaaS Infrastructure as a service
Definition	A software distribution model in which a third-party provider hosts applications and makes them available to customers over the internet.	A model in which a third-party provider hosts application development platforms and tools on its own infrastructure and makes them available to customers over the internet.	A model in which a third-party provider hosts servers, storage and other virtualized compute resources and makes them available to customers over the internet.
Examples	Salesforce, NetSuite and Concur	AWS Elastic Beanstalk, Google App Engine and Heroku	AWS, Microsoft Azure and Google Compute Engine

Cloud Computing Deployment Models

Private cloud services are delivered from a business's data center to internal users. With a private cloud, an organization builds and maintains its own underlying cloud infrastructure (see Fig. 1.47). This model offers the versatility and convenience of the cloud, while preserving the management, control and security common to local data centers. Common private cloud technologies and vendors include VMware and OpenStack.

In the *public cloud* model, a third-party cloud service provider (CSP) delivers the cloud service over the internet. Public cloud services are sold on demand, typically by the minute or hour, though long-term commitments are available for many services. Customers only pay for the central processing unit cycles, storage or bandwidth they consume. Leading public CSPs include AWS, Microsoft Azure, IBM and Google Cloud Platform, as well as IBM, Oracle and Tencent.

A *hybrid cloud* is a combination of public cloud services and an on-premises private cloud, with collaboration and automation between the two. Companies can run mission-critical workloads or sensitive applications on the private cloud and use the public cloud to handle workload spikes in demand. The goal of a hybrid cloud is to create a unified, automated, scalable environment that takes advantage of all that a public cloud infrastructure can provide, while still maintaining control over mission-critical data.

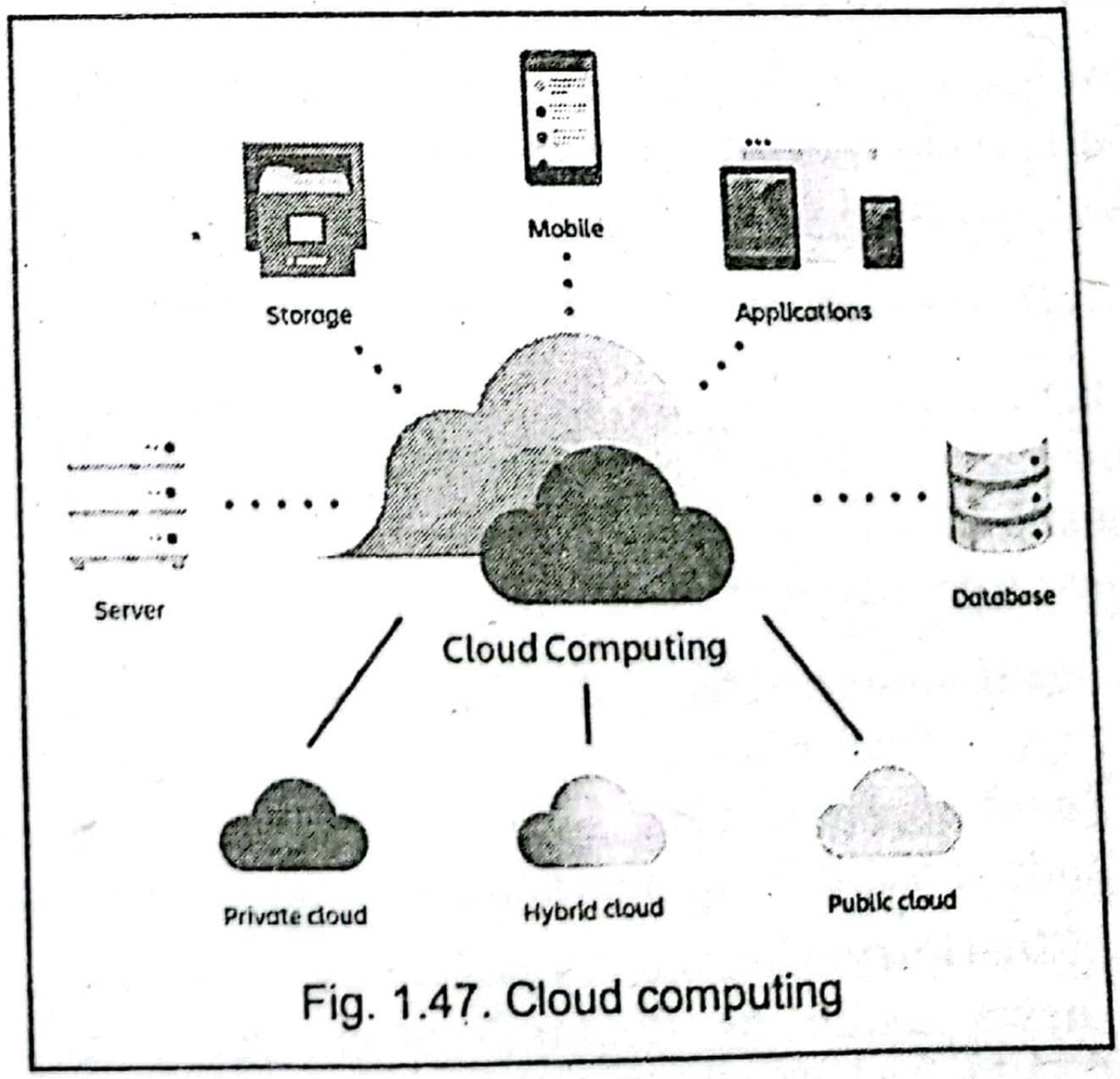


Fig. 1.47. Cloud computing

CamScanner

Table 1.4. The Three Main Cloud Deployment Models

	Private	Public	Hybrid
Definition	A cloud computing model in which an enterprise uses a proprietary architecture and runs cloud servers within its own data center.	A cloud computing model in which a third-party provider makes compute resources available to the general public over the internet. With public cloud, enterprises do not have to set up and maintain their own cloud servers in house.	A cloud computing model that includes a mix of on-premises, private cloud and third-party public cloud services with collaboration between the two platforms.
Characteristics	Single-tenant architecture. On-premises hardware Direct control of underlying cloud infrastructure.	Multi-tenant architecture. Pay-as-you-go pricing model	Cloud bursting capabilities. Benefits of both public and private environments.
Top vendors	HPE, Dell EMC, IBM/Red Hat, Microsoft, OpenStack	AWS, Microsoft Azure, Google Cloud	A combination of both public and private cloud providers

In addition, organizations are increasingly embracing a *multi-cloud model*, or the use of multiple IaaS providers. This enables applications to migrate between different cloud providers or to even operate concurrently across two or more cloud providers.

Organizations adopt multi-cloud for various reasons. For example, they could do so to minimize the risk of a cloud service outage or to take advantage of more competitive pricing from a particular provider. Multi-cloud implementation and application development can be a challenge because of the differences between cloud providers' services and APIs.

Multi-cloud deployments should become easier, however, as providers' services and APIs converge and become more standardized through industry initiatives such as the Open Cloud Computing Interface.

Characteristics and Advantages of Cloud Computing

Cloud computing has been around for several decades now, and today's cloud computing infrastructure demonstrates an array of characteristics that have brought meaningful benefits for businesses of all sizes. Some of the main characteristics of cloud computing are the following:

1. **Self-service Provisioning:** End users can use compute resources for almost any type of workload on demand. An end user can provision computing capabilities, such as server time and network storage, eliminating the traditional need for IT administrators to provision and manage compute resources.
2. **Elasticity:** Companies can freely scale up as computing needs increase and scale down again as demands decrease. This eliminates the need for massive

CamScanner

investments in local infrastructure, which might or might not remain active.

3. **Pay per use:** Compute resources are measured at a granular level, enabling users to pay only for the resources and workloads they use.
4. **Workload Resilience:** Cloud service providers (CSPs) often implement redundant resources to ensure resilient storage and to keep users' important workloads running—often across multiple global regions.
5. **Migration Flexibility:** Organizations can move certain workloads to or from the cloud—or to different cloud platforms—as desired or automatically for better cost savings or to use new services as they emerge.
6. **Broad Network Access:** A user can access cloud data or upload data to the cloud from anywhere with an internet connection using any device.
7. **Multi-tenancy and Resource Pooling:** Multi-tenancy lets numerous customers share the same physical infrastructures or the same applications yet still retain privacy and security over their own data. With resource pooling, cloud providers service numerous customers from the same physical resources. The resource pools of the cloud providers should be large and flexible enough so they can service the requirements of multiple customers.

These characteristics support a variety of important benefits and advantages for modern business, including the following:

1. **Cost Management:** Using cloud infrastructure can reduce capital costs, as organizations don't have to spend massive amounts of money buying and maintaining equipment. This reduces their capital expenditure costs—as they don't have to invest in hardware, facilities, utilities or building large data centers to accommodate their growing businesses. Additionally, companies don't need large IT teams to handle cloud data center operations because they can rely on the expertise of their cloud providers' teams. Cloud computing also cuts costs related to downtime. Since downtime rarely happens in cloud computing, companies don't have to spend time and money to fix any issues that might be related to downtime.
2. **Data and Workload Mobility:** Storing information in the cloud means that users can access it from anywhere with any device with just an internet connection. That means users don't have to carry around USB drives or an external hard drive to access their data. Users can access corporate data via smartphones and other mobile devices, enabling remote employees to stay up to date with co-workers and customers. End users can easily process, store, retrieve and recover resources in the cloud. In addition, cloud vendors provide all the upgrades and updates automatically, saving time and effort.
3. **Business Continuity and Disaster Recovery:** All organizations worry about data loss. Storing data in the cloud guarantees that users can always access their data even if their devices, *e.g.*, laptops or smartphones, are inoperable. With cloud-based services, organizations can quickly recover their data in the event of emergencies, such as natural disasters or power outages.

Disadvantages of Cloud Computing

Despite the clear upsides to relying on cloud services, cloud computing carries its own

CamScanner

challenges for IT professionals:

1. **Cloud Security:** Security is often considered the greatest challenge facing cloud computing. When relying on the cloud, organizations risk data breaches, hacking of APIs and interfaces, compromised credentials and authentication issues. Furthermore, there is a lack of transparency regarding how and where sensitive information entrusted to the cloud provider is handled. Security demands careful attention to cloud configurations and business policy and practice.
2. **Cost Unpredictability:** Pay-as-you-go subscription plans for cloud use, along with scaling resources to accommodate fluctuating workload demands, can make it tough to define and predict final costs.
3. **Lack of Capability and Expertise:** With cloud-supporting technologies rapidly advancing, organizations are struggling to keep up with the growing demand for tools and employees with the proper skill sets and knowledge needed to architect, deploy, and manage workloads and data in a cloud.
4. **Compliance with Industry Laws:** When transferring data from on-premises local storage into cloud storage, it can be difficult to manage compliance with industry regulations through a third party. It's important to know where data and workloads are actually hosted in order to maintain regulatory compliance and proper business governance.
5. **Management of Multiple Clouds:** Every cloud is different, so multi-cloud deployments can be a challenge for IT professionals.
6. **Cloud Performance:** Performance—such as latency—is largely beyond the control of the organization contracting cloud services with a provider. Network and provider outages can interfere with productivity and disrupt business processes if organizations are not prepared with an alternate plan.
7. **Building a Private Cloud:** Architecting, building and managing private clouds—whether for its own purpose or for a hybrid cloud goal—can be a challenging task for IT departments and staff.
8. **Cloud Migration:** The process of moving applications and other data to a cloud infrastructure often causes complications. Migration projects frequently take longer than anticipated and go over budget. The issue of workload and data repatriation—moving from the cloud back to a local data center—is often overlooked until unforeseen cost or performance problems arise.
9. **Vendor Lock-in:** Often, switching between cloud providers can cause significant issues. This includes technical incompatibilities, legal and regulatory limitations and substantial costs incurred from sizable data migrations.

CamScanner

Questions

1.1. What is a system? Why do we refer to a computer as a system?
1.2. Name three categories of peripheral devices.
1.3. What is the purpose of input and output devices?
1.4. What are the two main categories of input hardware?
1.5. What determines what the function keys on a key-board do?
1.6. What is the difference between dumb terminals and smart terminals?
1.7. List and describe two hand-held input devices.
1.8. What is an scanning device?
1.9. What are the three most common types of scanners? What type of data does each scan?
1.10. What are the main uses of voice-input devices?
1.11. Explain how a voice recognition system works?
1.12. What is the main difference between a mouse and a trackball?
1.13. Why is the process of formatting (initializing) necessary?
1.14. What is the function of primary storage, and how does secondary storage differ from it.
1.15. Describe why having more main memory, or RAM, in your computer (as opposed to less) is useful.
1.16. What is the difference between volatile and non-volatile storage? Give an example of each of these types of storage.
1.17. Differentiate between RAM and ROM.
1.18. What are registers?
1.19. What is a cache memory? How is it different from a primary memory?
1.20. What is the difference between hardcopy and softcopy?
1.21. What are the two types of display screen?
1.22. Define the following terms related to screen display: CRT, pixel, resolution, and monochrome.
1.23. List and describe four types of flat-screen technologies.
1.24. List some relative advantages and disadvantages of CRT and flat panel monitors.
1.25. Describe four hardware devices that produce hardcopy output.
1.26. What is a printer? What are the two main types of printers?
1.27. What is a letter-quality printer? A dot-matrix printer?
1.28. What is an impact printer? A non-impact printer?
1.29. Differentiate between impact and non-impact printers. Give one example of each.
1.30. What advantages does a laser printer have over other types of printers?

CamScanner

1.31. What is a plotter? What is its principal use?
1.32. Why do many people find it necessary to use a plotter to output information?
1.33. Distinguish between primary and secondary storage.
1.34. What is secondary storage? How does it differ from a primary storage?
1.35. Distinguish between a sequential access, a cyclic access, and a random-access device. Give one example of each.
1.36. What factors determine the storage capacity of hard disks?
1.37. What is access time?
1.38. A hard-disk consists of 8 platters. Each platter has 300 tracks and there are 50 sectors per track. If 256 bytes can be stored per sector then calculate the total number of bytes that can be stored in this hard-disk.
1.39. What is the purpose of backing up the files on cloud storage?
1.40. What are the most convenient forms of backup storage? Why?
1.41. How are data recorded on and read from an optical disk? What are the major advantages of an optical disk storage?
1.42. What is Blu-ray used for?
1.43. What does "read-only" mean?
1.44. As they apply to disk storage, what is meant by the terms *fragmentation* and *defragmentation*?
1.45. Why do users use memory cards? Describe some of the characteristics of each card.
1.46. What is an operating system? Why is it necessary for a computer system?
1.47. List out the various functions normally performed by an operating system.
1.48. "The operating system tends to isolate the hardware from the users". Discuss the statement.
1.49. What are the principal functions performed by an operating system?
1.50. Distinguish between 'System Software' and 'Application Software'.
1.51. What is lossy compression?
1.52. What is a compiler? Why is it required? A computer supports five high level languages, how many compilers will this computer have?
1.53. What is an interpreter? How does it differ from a compiler?
1.54. What is the difference between a source program and an executable program?
1.55. Define the term 'low level language'. Why are they used?
1.56. Give some examples of high-level languages.
1.57. What is an assembly language? What are its advantages over machine language?
1.58. How have IT architectures changed over time as new technologies have emerged?
1.59. What is the basic meaning of cloud computing?
1.60. What is cloud computing? How does it support business objectives?

CamScanner

Objectives and MCQs

Each of the following questions has four alternative answers, of which one or more may be correct. Pick up the correct alternative/alternatives.

1.1. What does a dumb terminal consist of?
- (a) Microprocessor and monitor
- (b) Keyboard and printer
- (c) Keyboard and monitor
- (d) Keyboard and microprocessor

1.2. A barcode reader is an example of a(n)
- (a) processing device
- (b) storage device
- (c) input device
- (d) output device

1.3. Which of the following can be output by a computer?
- (a) graphics and text
- (b) voice
- (c) computer usable information
- (d) All of the above

1.4. Printers and screens are common forms of
- (a) input units
- (b) output units
- (c) storage units
- (d) processing units

1.5. An example of peripheral equipment is the
- (a) CPU
- (b) spreadsheet
- (c) printer
- (d) personal computer

1.6. A monitor looks like a television set but it does not
- (a) give a steady picture
- (b) receive television signals
- (c) give a clear picture
- (d) display graphics

1.7. It is a non-impact printer that can produce very high-quality letter printing,
- (a) daisy wheel printer
- (b) chain printer
- (c) Laser printer
- (d) dot matrix printer

1.8. The advantage of the laser printer is that
- (a) it is very slow
- (b) the output is of inferior quality

CamScanner

(c) it cannot produce a wide range of type fonts
(d) None of the above

1.9. To produce high quality graphics in color one would like to use
(a) plotter
(b) inkjet printers
(c) laser printer
(d) All of the above

1.10. Laser beam technology is used for
(a) terminals
(b) optical disks
(c) keyboards
(d) hard disk

1.11. Users subscribe to a cloud storage service to
(a) save time by storing large files instantaneously
(b) allow others to access their files
(c) store offsite backups of data
(d) all of the above

1.12. Step-by-step instructions that run the computer are
(a) hardware
(b) documents
(c) CPUs
(d) software

1.13. A software that enables users and applications to interact with the computer hardware is called
(a) operating system
(b) application software
(c) client software
(d) database management system

1.14. A source program is a
(a) program written in machine language
(b) program written in a high-level language
(c) program required to run a computer
(d) None of these.

1.15. Once a program has been translated into the machine language, it is called
(a) a system program
(b) logical program
(c) an executable file
(d) an application program

1.16. Machine language is a language
(a) understood directly by a computer
(b) needs to be translated before computer can understand it
(c) in which programs were written first

CamScanner

(d) which is machine independent

1.17. Which of the following languages is most efficient
(a) Assembly language
(b) High level languages
(c) Machine language
(d) All are equally efficient

1.18. A compiler is a
(a) part of a computer
(b) program to convert a high level language into machine language
(c) source program
(d) executable file

Objectives

1.1. The term computer is derived from the word compute.
1.2. Data means names, numbers, facts, or anything needed to work out a problem.
1.3. Input and output devices—as well as storage devices—are categorized as peripheral.
1.4. The keys with little arrows on keyboard are called cursor-movement keys.
1.5. A dumb terminal can be used only to input and receive data; it cannot do any independent processing.
1.6. A voice-input device converts a person's speech into digital code that your computer can understand.
1.7. A mouse is rolled about on the desktop to direct a pointer on the computer's display screen.
1.8. The term CPU is an abbreviation for central processing unit.
1.9. The small silicon chips that contain the CPU of a computer are called microprocessors.
1.10. Silicon is the semiconductor material commonly used to make integrated circuits (chips).
1.11. If data is stored in volatile memory, it will be lost if the power goes off.
1.12. Is the data on an internal hard disk volatile or non-volatile? non-volatile
1.13. RAM stands for random access memory.
1.14. Is RAM volatile or non-volatile? volatile
1.15. ROM stands for read-only memory.
1.16. Output is available in two principal forms: hardcopy and softcopy.
1.17. The more pixels that can be displayed on the screen, the better the resolution of the image.
1.18. Voice-output devices convert digital data into speech-like sounds.
1.19. Impact printers that form characters using pins in a rectangular patterns are called dot-matrix printer.

CamScanner

1.20. Two advantages of impact printers are (1)they are inexpensive and (2)they can produce carbon copies.
1.21. PostScript is a printer language; it describes the shape and position of letters and graphics to laser printers.
1.22. Plotters can produce high-quality graphics and are used most often for outputting maps and architectural drawings.
1.23. Personal computers have an internal hard disk that can hold large amounts of data. Is this primary storage or secondary storage? secondary storage
1.24. On a hard disk, data is recorded in rings called tracks.
1.25. Hard disk record addresses are based on three locations: the surface number; sector number and track number.
1.26. Backup is the name given to duplicate storage.
1.27. Even in systems that process data on disks, tapes are often used for backing up.
1.28. The time needed by a disk storage system to locate a disk record is called its access time.
1.29. An optical disk is a removable disk on which data is written and read through the use of laser beams.
1.30. An optical disk is a type of storage media that consists of a flat, round, portable disc made of metal, plastic, and lacquer that is written and read by a laser.
1.31. The sequence of instructions written in computer language is called program.
1.32. The person who writes a program is called programmer.
1.33. Translator converts a program written in high level language into machine language.
1.34. High level languages are more suitable for human use than are machine languages.
1.35. The output of the computer language translation is called executable file.
1.36. The executable file is in machine language.
1.37. The program prepared by programmer is called source program.
1.38. Compilers and interpreters are used to convert high level language into machine language.
1.39. A low-level language comprises of binary 0s and 1s.
1.40. Programming in high level language is an easy task.

CamScanner

CHAPTER 2

THE SYSTEM UNIT AND COMPUTER MEMORY

By: Sir Ahad

A *system unit* is the part of a computer that houses the primary devices that perform operations and produce results for complex calculations. It includes the motherboard, CPU, RAM and other components, as well as the case in which these devices are housed. This unit performs the majority of the functions that a computer is required to do. If you take off the cabinet, you will find that many parts can be easily removed for replacement. In addition, many computers are expandable. Thus, more main memory may be added as well as certain other devices.

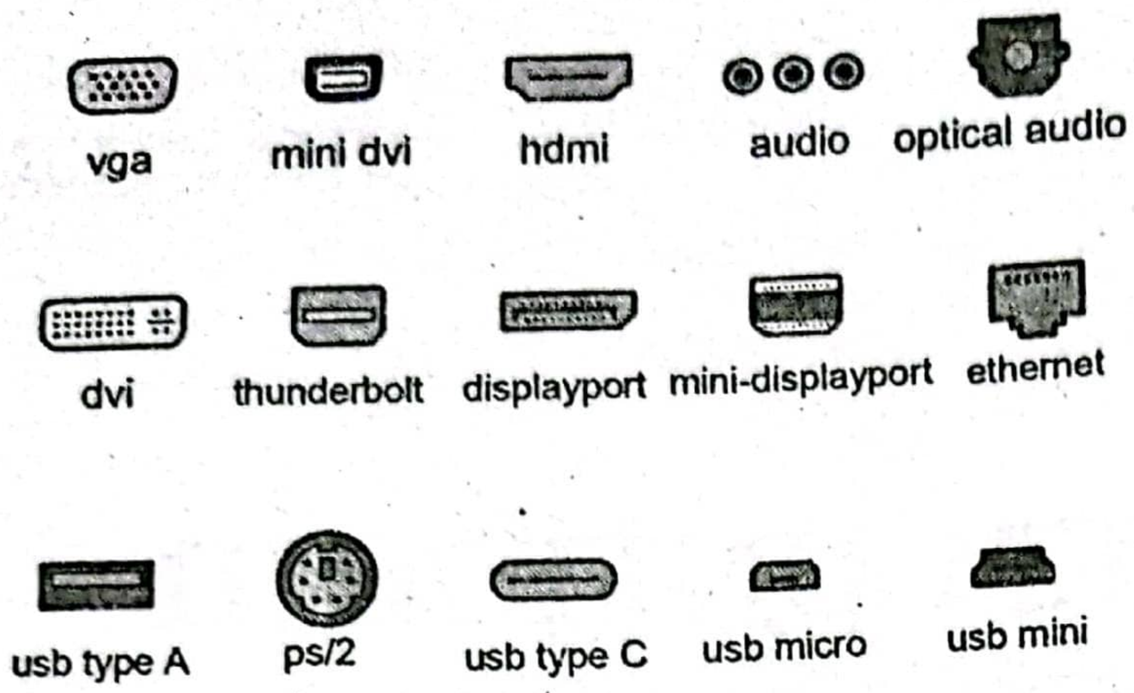
The components of the system unit that we shall describe are (see Fig. 2.1):

1. The power supply
2. The motherboard
3. Microprocessor—including registers, bus, clock, and coprocessors
4. Graphics processing unit (GPU)
5. Main memory or random access memory (RAM)
6. Read-only memory (ROM)
7. Expansion cards and slots
8. Ports
9. Cooling system

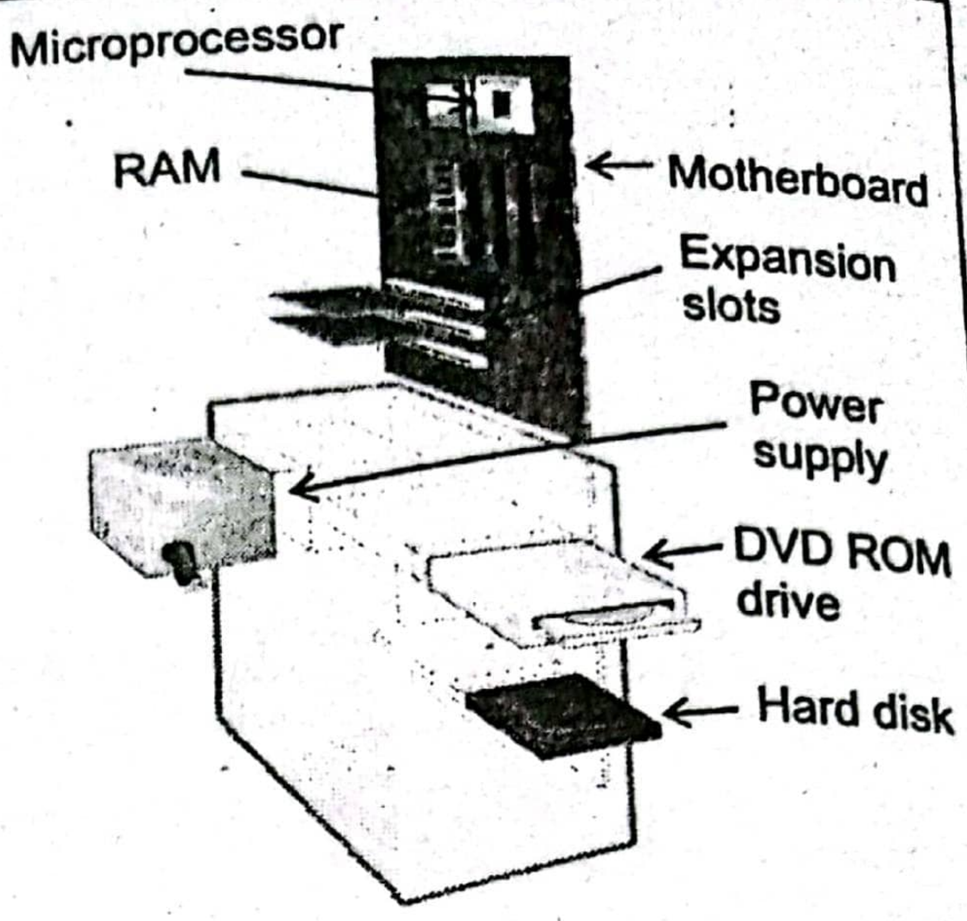
The Power Supply

Power supply is a box inside the system unit that provides electrical power to all components in system unit. It converts 220 volt main AC supply to the required 5 or 12 volt DC supply which computer components use. If the monitor is built into the system unit, as it is in the laptop or iMac, the power supply also provides electricity to it. Since electricity generates a lot of heat, the power supply unit is equipped with a fan that prevents the system and components from overheating.

CamScanner



(a) Different ports you may find on the rear panel of a PC system unit



(b) Inside view of a PC system unit

Fig. 2.1. Different ports and PC system unit

The Motherboard

The *motherboard*, also called the *system board* or *logic board*, is an electronic circuit board and foundation of a computer that is the biggest board in a computer chassis. It allocates power and allows communication to and between the CPU (central processing unit), RAM (random-access memory), hard drive, video card and all other computer hardware components. There are multiple types of motherboards, designed to fit different types and sizes of computers.

Each type of motherboard is designed to work with specific types of processors and memory, so they don't work with every processor and type of memory. However, hard drives are mostly universal and work with most motherboards, regardless of the type or brand. Some special features of the motherboard:

1. Motherboards significantly support various components of a computer.
2. It supports a single type of CPU along with different types of memory.
3. It is also responsible for compatible power supplies and allows everything works correctly together.
4. Other devices like a sound card, graphics card, network card, different adapters, and peripherals are also compatible with motherboard. If they are not, then they will not work properly.

Every computer has a BIOS (short for *Basic Input/Output System*) chip on motherboards that is used to store BIOS settings. BIOS is a ROM chip that lets you access and set up your computer system at the most basic level. BIOS is the first software that runs when you power on your system, performing an initial pack of diagnostic tests known as a POST (Power On Self-Test) to see if there are any issues with the hardware. POST is the first step in your hardware's boot sequence. If the

CamScanner

computer does not pass the POST, you hear a combination of beeps indicating what is malfunctioning in the computer.

There are two main types of BIOS:

1. Legacy BIOS: Legacy BIOS is used in older motherboards to turn on the computer, and it controls how the CPU and different computer components talk to each other. Unfortunately, the Legacy BIOS has limitations. For example, it can't handle or recognize data drives larger than 2.1 TB.
2. UEFI: The acronym stands for Unified Extensible Firmware Interface. Unlike the Legacy BIOS, the UEFI can accommodate 2.2 TB or larger drives. UEFI also has greater scalability, higher performance, better programmability, and higher security.

BIOS is traditionally called CMOS RAM because it uses a volatile, low-power complementary metal-oxide-semiconductor (CMOS) SRAM powered by a small *CMOS battery* when your computer is unplugged. The CMOS battery on computer's motherboard acts as an emergency power for the BIOS and date and time settings. The CMOS keeps those BIOS or UEFI settings stored in CMOS RAM when computer doesn't receive any external power, *e.g.*, when the power source is disconnected.

All batteries degrade over time, though, and CMOS battery may die with time. If that happens, the CMOS memory chip won't be able to hold BIOS settings between power cycles, resulting in a reset, which can be very frustrating.

In a computer, a *jumper* is a pair of prongs that are electrical contact points set into the computer motherboard or an adapter card. When you set a jumper, you place a plug on the prongs that completes a contact. In effect, the jumper acts as a switch by closing (or opening) an electrical circuit. Jumpers can be added or removed to change the function or performance of a PC component. A group of jumpers is sometimes called a *jumper block.*

Today, computers come with preset jumpers, although some manufacturers provide *jumper settings* in the instruction manual so that the owner can reset the jumpers if they want to customize performance.

A motherboard also includes expansion slots. *Expansion slots* are places for connecting additional circuit boards or electrical components.

The Microprocessor

As computers have evolved, the size of the microprocessor has become smaller while its speed and capacity have increased tremendously. Indeed, the *microprocessor*, also called the CPU, is a small chip, on which electronic circuitry has been etched. The chip is mounted on a piece of ceramic or plastic, called *carrier*, with metal leads attached to it (Fig. 2.2). The microprocessor manages the computer processing and the transfer of data to and from primary storage, commonly called random access memory (RAM).

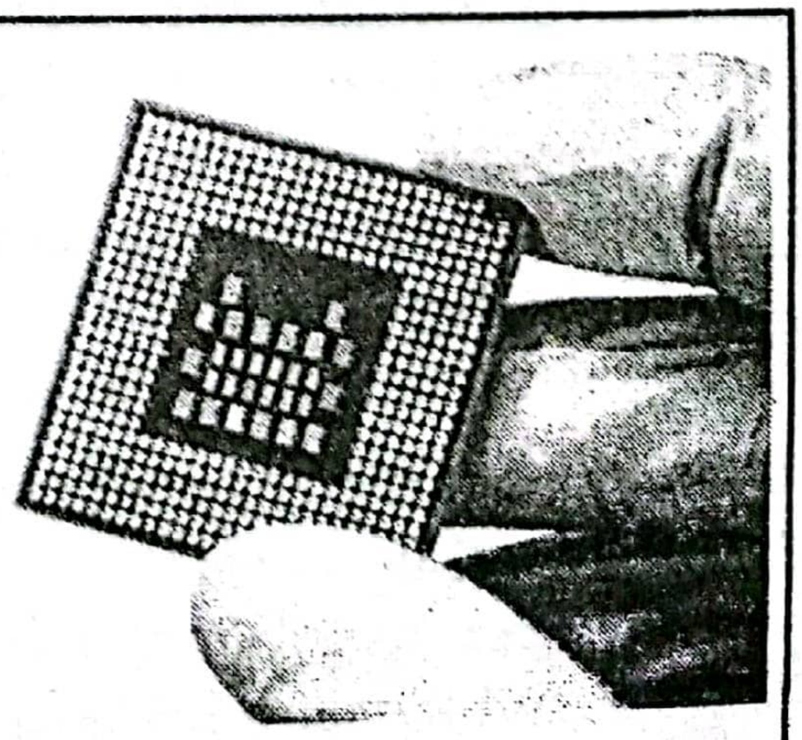


Fig. 2.2. A microprocessor chip

CamScanner

Main Components of a Microprocessor

The CPU has three main components (Fig. 2.3):

1. The control unit
2. The arithmetic/logic unit
3. Registers

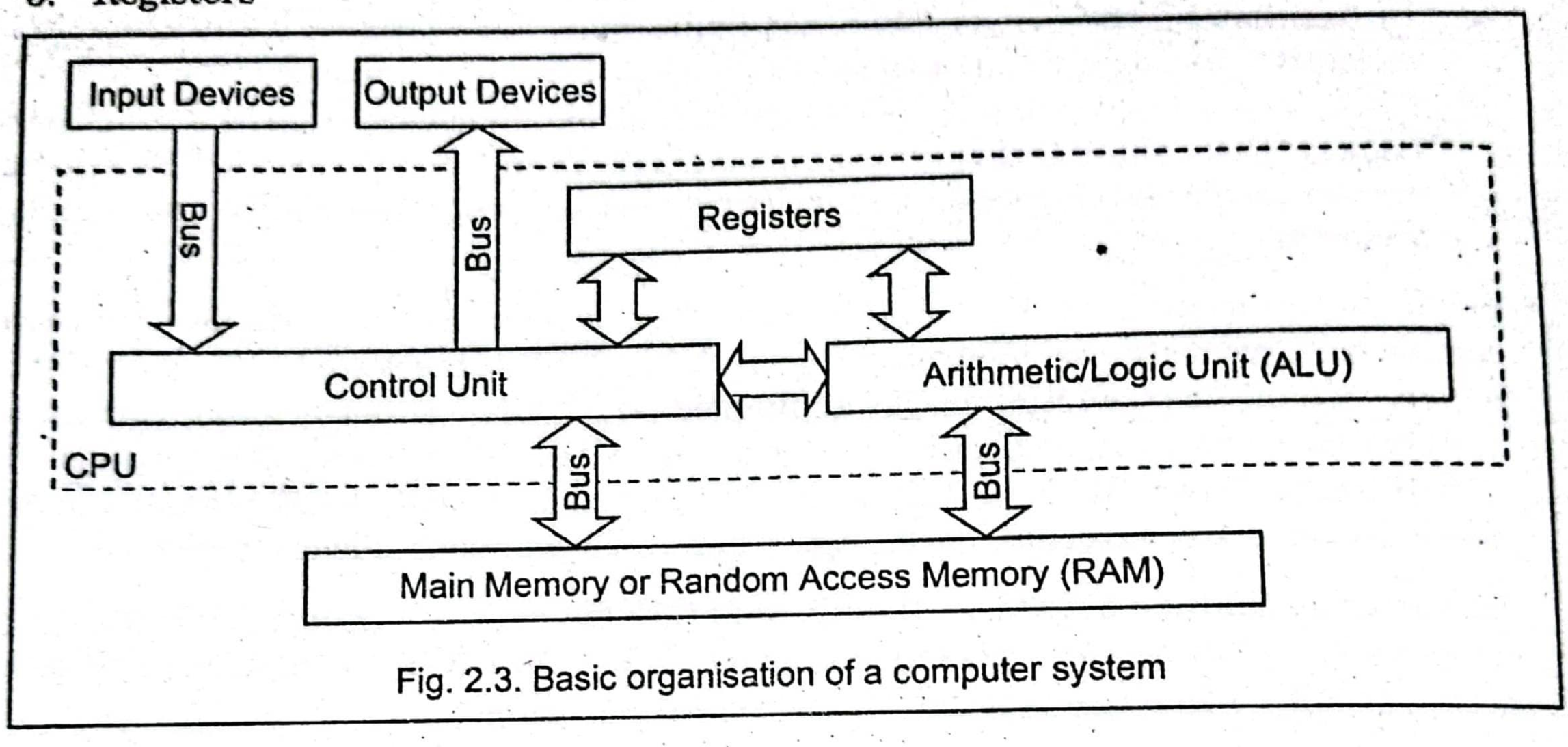


Fig. 2.3. Basic organisation of a computer system

The Control Unit: All the computer's resources are managed from the control unit, whose function is to coordinate all the computer's activities. You can think of the control unit as a traffic cop, directing the flow of data around the CPU and around the computer.

The control unit contains the CPU's instructions for carrying out commands. The *instruction set,* which is built into the circuitry of the control unit, is a list of all the operations that the CPU can perform. Each instruction in the instruction set is accompanied by *microcode*—very basic directions that tell the CPU how to execute the instruction. When the computer runs a program, it looks up the commands contained in the program in the CPU's instruction set and executes them in order.

The Arithmetic/Logic Unit: When the control unit encounters an instruction that involves arithmetic or logic operation, it passes control to the second component of the CPU, the arithmetic/logic unit (ALU). *Arithmetic operations* are, as you might expect, the fundamental math operations: addition, subtraction, multiplication, and division. *Logical operations* consist of comparisons. That is, two pieces of data are compared to see whether on is equal to (=), less than (<), or greater than (>) the other. There are also combinations, such as "greater than or equal to".

Registers: The CPU also uses specialized storage areas called *registers* in which data is held while operations are performed on it. Registers usually consist of a small amount of fast storage. Registers work under the direction of the control unit to accept, hold, and transfer instructions or data and perform arithmetic or logical comparisons at high speed. For example, if a logic function such as "Is number A is equal to number

CamScanner

B?" is called for, the two numbers are placed in registers by the control unit and compared by the ALU; the answer is then returned to the control unit.

Features of a Microprocessor

Some features of microprocessors are:

1. Registers and word size
2. Bus and bus width
3. Cache
4. Clock and clock speed
5. Coprocessors

Registers and Word Size: The size (capacity) and number of registers, which vary according to a CPU's design, dramatically affect a computer's processing speed. The size of a register determines how many bits can be processed at once. The size or capacity of a register is referred to as *word size*. Some personal computers have registers that hold only 32 bits. Newer personal computers have 64-bit registers, which can process data twice as fast as 32-bit ones. A computer with a smaller word size will eventually get the same amount of data processed as will a computer with a larger word size. However, it takes longer.

Bus and Bus Width: A computer sends several kinds of electronic signals back and forth among its various components. For efficiency, separate paths on the motherboard called *buses* each carry only one kind of signal; this boosts the computer's speed and simplifies its operation (see Fig. 2.4). The functions of the bus are:

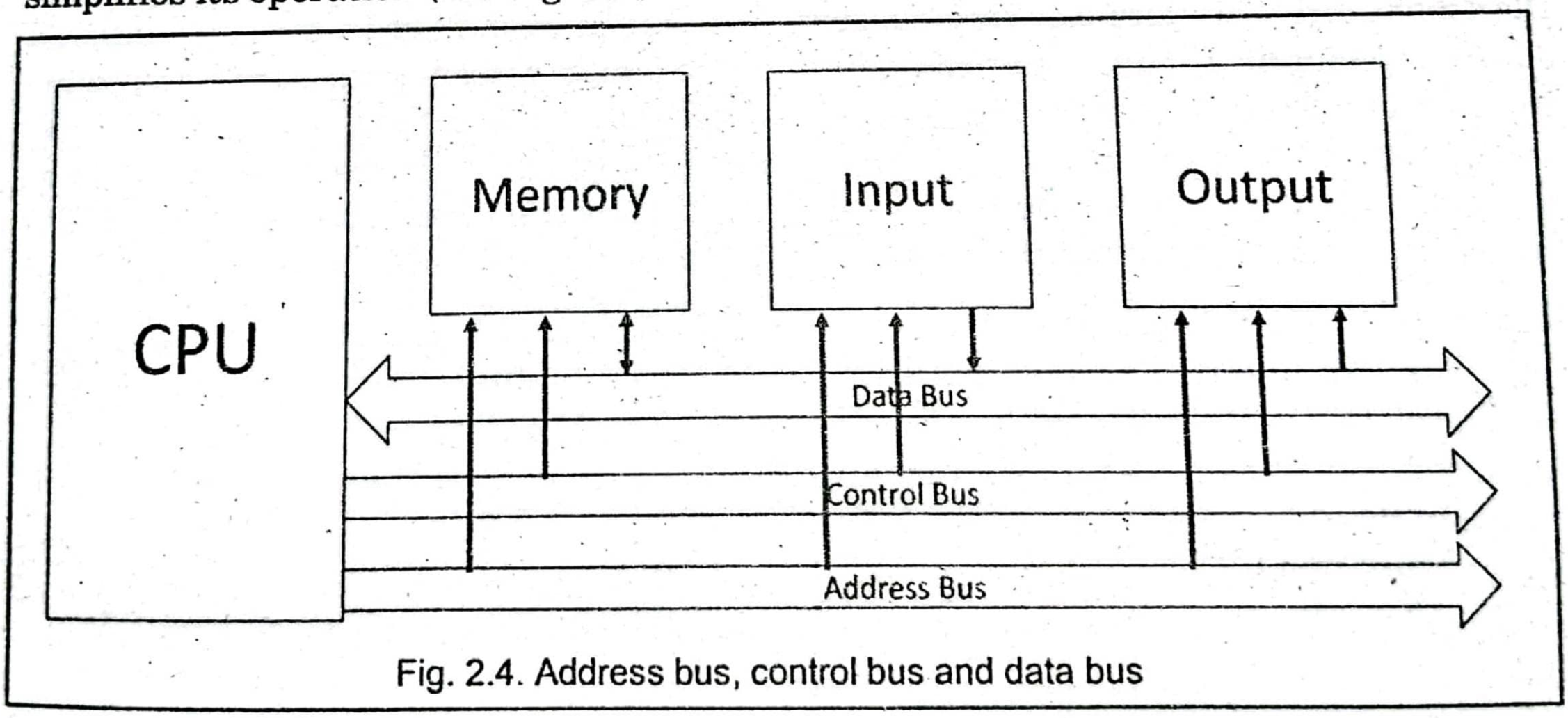


Fig. 2.4. Address bus, control bus and data bus

1. It carries information from one component to another.
2. It carries data, addresses, or control signals.
3. One component of the computer can interact with others by bus.

While PC motherboards have different buses for expansion cards and external devices, all computers have three fundamental buses: Address bus, Control bus, and Data bus.

Address bus—carries memory addresses from the processor to other components such

CamScanner

as primary storage and input/output devices. It allows the CPU to reference memory locations within the device. It connects the CPU and other peripherals and carries only memory addresses. In a computer system, each peripheral or memory location is identified by a numerical value, called an *address* and the address bus is used to carry this numerical value as well as it also contains a few control lines to carry control commands. The address bus is unidirectional, bits flow in one direction, from the processor to a peripheral or a memory location.

Control bus—carries control signals from the processor to other components. The control signal is used for controlling and coordinating the various activities across the computer. It is generated from the control unit within the CPU. The control unit generates a specific control signal for every operation, such as memory read or input/output operation. This signal is also used to identify a device type, with which the microprocessor intends to communicate. The control bus also carries the clock's pulses. The control bus is unidirectional.

Data bus—carries the data from the processor to memory and peripheral devices and vice versa. Thus, it is a bidirectional bus. It is one of most important parts of the connections to the processor because every program instruction and every byte of data must travel across the bus at some point.

The major difference between address bus, control bus, and data bus is that address bus identifies the source or destination of data, data bus used to carry data signals while the control bus controls the signals among devices.

The more data the data bus can carry at one time, the faster the processing speed of the computer. The data-carrying capacity of a bus is expressed in terms of its *bus width,* measured in bits. Older computers had microprocessors with data buses that were 32 bits wide; this meant they could carry 32 bits, or four byte, at a time. Newer microprocessors are 64 bits wide, which means they can move twice times more data through their data buses. The Intel Pentium microprocessor has a data bus that is 64 bits wide. Some supercomputers contain data buses that are 128 bits wide.

A computer *word* is defined as the number of bits or bytes that a microprocessor can manipulate as a unit. The length of a word varies by computer. Usually, the bus width is related to the number of bits in the word size. For instance, a 64-bit central processing unit has a 64-bit bus, meaning that data can be sent over the bus lines in groups of 64 bits—that is, a word at a time.

The number of bits a computer can store in a word and carry on its busses is very important. In general, the larger the word size or the more bits in a bus, the more powerful the computer. The larger the word or bus size:

1. the more data the computer can transfer at a time, making the computer faster.
2. the larger the address numbers the computer can reference, allowing more memory.
3. the greater the number and variety of instructions that the computer can support.

For example, the Intel Pentium (Fig. 2.5) central processing unit processes data by using a 32-bit word size whereas Intel Core i7 or i9 (Fig. 2.6) use 64-bit word size. Obviously, sending data 64 bits at a time is more efficient than sending it 32 bits at a

CamScanner



Fig. 2.5. Intel Pentium

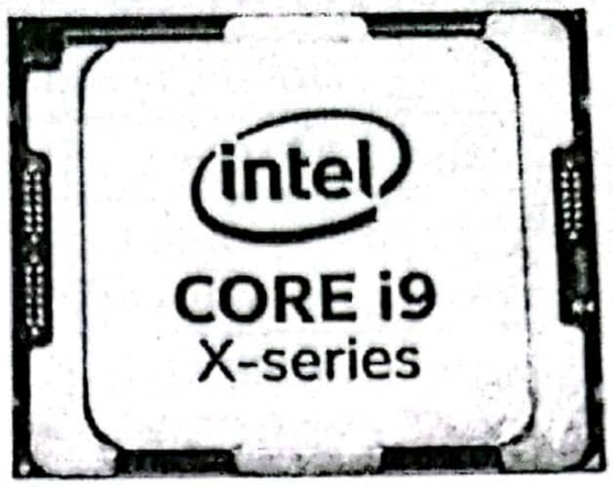


Fig. 2.6. Intel Core i9

time, so the computer with the 64-bit bus is more powerful (and more expensive!).

Cache: The term cache generally refers to hardware or software that temporarily stores frequently accessed data.

Moving data between RAM and the CPU's registers is one of the most time-consuming operations a CPU must perform, simply because RAM is much slower than the CPU. A partial solution to this problem is to include a cache memory on the microprocessor chip. Cache (pronounced "cash") is similar to RAM, except that it is extremely fast compared to normal memory, and is used in a different way.

The most frequently used instructions are kept in cache memory so the CPU can look there first. This allows the CPU to run faster because it doesn't have to take time to swap instructions in and out of main memory. Large, complex programs benefit the most from having a cache memory available. Cache chips are faster but much more expensive than main RAM chips.

L1, L2 and L3 are different levels of cache memory. All types of cache memory are used to reduce access times and latency, while improving I/O. Because almost all application workloads depend on I/O operations, caching improves application performance. Faster data access and I/O also improves computer performance.

Because it's built directly into the CPU, L1 cache memory provides the fastest possible access to memory locations, which supports faster CPU performance. L2 cache can be integrated in the processor, but more frequently is placed on a chip adjacent to the CPU, as is L3 cache. As a result, the adjacent chips that hold L2 and L3 memory cache can be somewhat slower and usually have a direct pathway to the CPU to optimize performance.

Clock and Clock Speed: The *clock* controls how fast all the operations within a CPU take place. If you are buying a personal computer system, clock speed is an important measurement to know about. *Clock speed,* which is expressed in gigahertz, indicates how fast a computer processes information. Clock speed is a function of the ease with which electricity passes through the CPU. Hertz (Hz) is equivalent to cycle per second (c/s); 1 GHz is equal to 1 billion cycles per second.

Number of cores, production year, bus speed, and clock speed of common Intel microprocessors are shown in the table 2.1. The higher the number, the faster the microprocessor.

CamScanner

Table 2.1. Comparison Chart of some Popular Intel Processors for PCs

	Intel Core i3	Intel Core i5	Intel Core i7	Intel Core i9
Production Year	2010-present	2009-present	2008-present	2017-present
Number of Cores	2 /w hyperthreading	2 , 4 /w hyperthreading	4, 4 /w hyperthreading	10-18 (with hyperthreading)
Bus Speed	1066 MHz-5 GT/s	2.5 - 5 GT/s	4.8 GT/s, 6.4 GT/s	8 GT/s
Clock Speed	1.2 GHz - 3.7 GHz	1.06 GHz - 4.2 GHz	1.6 GHz - 4.4 GHz	2.90 GHz - 4.30 GHz
Cache Size	3MB	3-6MB	4-8MB	16MB

Coprocessors: A coprocessor is an additional processor unit or an entirely different circuitry that is designed to complement the central processing unit (CPU) of a computer. Its basic functionality is to offload other processor-intensive tasks from the CPU in order to achieve accelerated system performance, by allowing the CPU to focus on tasks essential to the system. There are various types of coprocessors available to perform unique tasks – from I/O interfacing or encryption, string processing, floating-point arithmetic and signal processing.

Currently CPUs have been designed to absorb functionalities of popular coprocessors. But there are still coprocessors that are separately available, allowing customization for personal or business use. The graphics processing unit (GPU) is the most common example of this; it is designed solely for graphics processing so that the CPU does not have to do any of it. Other examples are broadband signal processing units and encryption/decryption units.

External Architecture of a Microprocessor

Modern microprocessors are incredibly complex devices capable of doing many things and thus requiring many pins. The pins of the microprocessor, or any chip for that matter, carry electrical signals in and out of the chip to allow it to function in a system. It is important to remember that processors are circuits: more complex circuits typically require access to outside hardware and thus more pins.

As processors grew ever more complex over the years, the required number of pins grew. In modern microprocessors, we can see pins for the address bus, the data bus, power, a clock as well as a few control, interrupt and status pins. Wider data buses (current mainstream CPUs have 64 bit data buses) and address buses required more pins, the increasing power requirements also mandated more pins be used for power transmission, etc. Not only that, the increasing diversity and complexity of hardware integrated on both the microprocessor and motherboard itself also required more pins to be allocated to those purposes. Given that a modern microprocessor has in excess of 1000 pins, each pin has a specific function and the data sheet for the chip describes each.

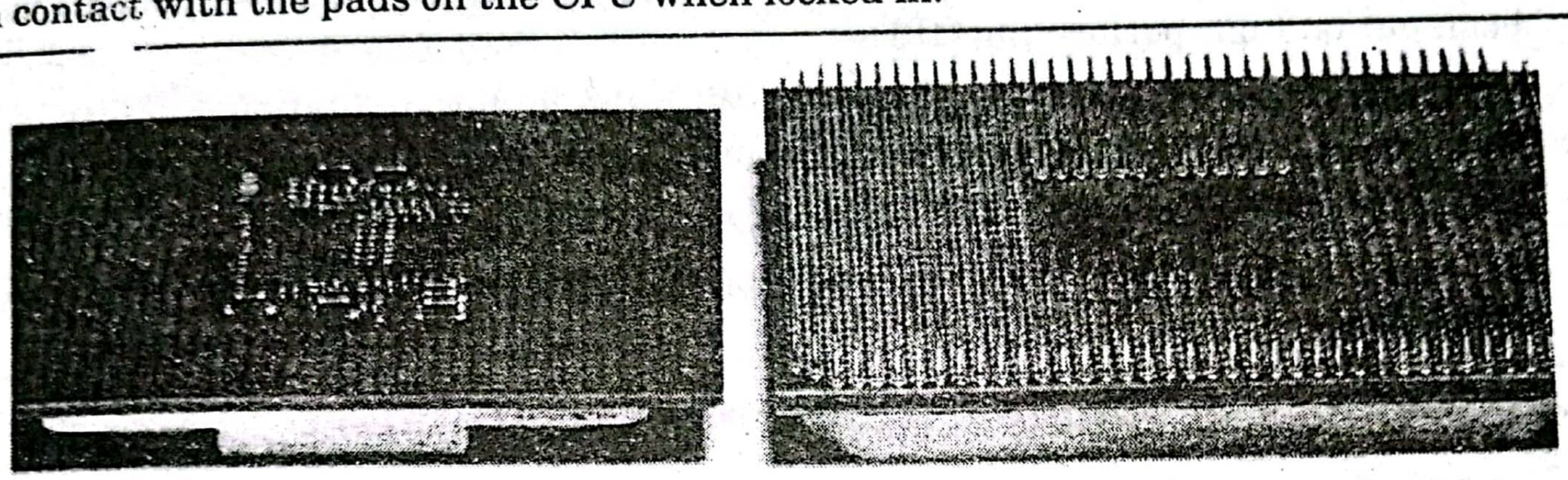
Here are some examples of processors and their pin counts:

1. Intel 8086 had 40 pins.
2. Original Pentium had 273 pins.

CamScanner

3. Intel Core i7 chips have 2011 pads.
4. AMD AM4 CPUs have 1331 pins.

It is important to note that there are three different socket types. Not all processors have pins, some have pads (see Fig. 2.7) and other have solder balls. Modern Intel processors do not have pins sticking out on their bottom. Instead, they have small *pads* which are also known as *lands*. As such with Intel, instead of having the CPUs with pins sticking out, the CPU Socket on the motherboard has pins sticking out that come in contact with the pads on the CPU when locked in.



(no pin)
Intel Processor

(pin)
AMD Processor

Fig. 2.7. Land Grid Array (with flat contacts) Left; and Pin Grid Array processors (with needles sticking out) Right.

Graphics processing unit (GPU)

A graphics processing unit (GPU) is a specialized processor that renders graphics and images by performing rapid mathematical calculations. GPUs are used for both professional and personal computing. Traditionally, GPUs are responsible for the rendering of 2D and 3D images, animations and video—even though, now, they have a wider use range.

In the early days of computing, the central processing unit (CPU) performed these calculations. As more graphics-intensive applications were developed, however, their demands put a strain on the CPU and decreased performance. GPUs were developed as a way to offload those tasks from CPUs and to improve the rendering of 3D graphics. GPUs work by using a method called parallel processing, where multiple processors handle separate parts of the same task.

GPUs are well known in PC gaming, allowing for smooth, high-quality graphics rendering. Developers also began using GPUs as a way to accelerate workloads in areas such as artificial intelligence (AI).

GPUs are found in personal computers, workstations, cell phones, and game consoles. The first company to develop the GPU was Nvidia. Some GPUs are integrated into the motherboard; high-powered GPUs are on the video cards and have their own RAM.

Uses of GPUs: Today, graphics chips are being adapted to a wider variety of tasks than originally designed for, partially because modern GPUs are more programmable than they were in the past.

CamScanner

Some examples of GPU use cases include:

1. GPUs can accelerate the rendering of real-time 2D and 3D graphics applications.
2. Video editing and creation of video content has improved with GPUs. Video editors and graphic designers, for example, can use the parallel processing of a GPU to make the rendering of high-definition video and graphics faster.
3. Video game graphics have become more intensive computationally, so in order to keep up with display technologies—like 4K and high refresh rates—emphasis has been put on high-performing GPUs.
4. GPUs can accelerate machine learning. With the high-computational ability of a GPU, workloads such as image recognition can be improved.
5. GPUs can share the work of CPUs for AI applications.
6. GPUs have also been used to mine bitcoin and other cryptocurrencies like Ethereum.

Main Memory or Random Access Memory (RAM)

RAM, or *random access memory* described earlier as main memory or primary storage, is used to store programs and data that the CPU is presently processing. The reason it is called "random access" memory is that data can be accessed by CPU at random from anywhere in a RAM chip in equal amounts of time. RAM is *volatile memory* which means that any data stored in RAM is lost when the computer is switched off; that is, it is temporary storage. (Secondary storage, which is described in the chapter 1, is permanent storage, such as data stored on hard disk. Data from this kind of storage must be loaded into RAM before it can be used.)

RAM capacity is expressed in gigabytes. This capacity has grown over the years. In 1979, a personal computer with 64 KB of RAM (about 64 thousand characters of data) was considered satisfactory. Today many users have 16 GB or more of RAM. Physically, it consists of some chips on a small circuit board.

In general, the greater your machine's memory capacity, the better because:

1. It can receive and use much larger programs.
2. It can hold copies of more than one program in main memory to support the sharing of computer by more than one user at a time.
3. It can operate faster and more efficiently.
4. It will be able to use new, sophisticated software.
5. It can hold images for creating graphics and animation.
6. It can work with and manipulate more data at one time.

Like the microprocessor, RAM consists of circuit-inscribed silicon chips attached to the motherboard. RAM chips are often mounted on a small circuit board, such as a SIMM or DIMM, which is plugged into the motherboard. A SIMM (*single inline memory module*) has RAM chips on only one side. A DIMM (*dual inline memory module*) has RAM chips on both sides. Although Personal Mobile Devices like the iPad use individual SIMMs, memory sold for personal computers is DIMMs.

RAM chips are available in two types:

CamScanner

DRAM (Dynamic RAM), is widely used as a computer's main memory. Each DRAM memory cell is made up of a transistor and a capacitor within an integrated circuit, and a data bit is stored in the capacitor. Since transistors always leak a small amount, the capacitors will slowly discharge, causing information stored in it to drain; hence, DRAM has to be refreshed (given a new electronic charge) every few milliseconds to retain data.

SRAM (Static RAM) is made up of four to six transistors. It keeps data in the memory as long as power is supplied to the system unlike DRAM, which has to be refreshed periodically. As such, SRAM is faster but also more expensive, making DRAM the more prevalent memory in computer systems.

Below table 2.2 lists some of the differences between SRAM and DRAM.

Table 2.2. Difference between SRAM and DRAM

SRAM	DRAM
SRAM has lower access time, so it is faster compared to DRAM.	DRAM has higher access time, so it is slower than SRAM.
SRAM is costlier than DRAM.	DRAM costs less compared to SRAM.
SRAM requires constant power supply, which means this type of memory consumes more power.	DRAM offers reduced power consumption, due to the fact that the information is stored in the capacitor.
Due to complex internal circuitry, less storage capacity is available compared to the same physical size of DRAM memory chip.	Due to the small internal circuitry in the one-bit memory cell of DRAM, the large storage capacity is available.
SRAM has low packaging density.	DRAM has high packaging density.

There are many types of DRAM that can be used in a device. Some examples include the following:

SDRAM (synchronous dynamic RAM) "synchronizes" the memory speed with CPU clock speed so that the memory controller knows the exact clock cycle when the requested data will be ready. This allows the CPU to perform more instructions at a given time. Typical SDRAM transfers data at speeds up to 133 MHz.

RDRAM (Rambus dynamic RAM) takes its name after the company that made it, Rambus. It was popular in the early 2000s and was mainly used for video game devices and graphics cards, with transfer speeds up to 1 GHz.

DDR SDRAM (double data rate synchronous dynamic RAM) is the faster version of SDRAM that performs its operations on both the rising and falling edges of the clock signal without any increase in clock frequency. This method is called "double pumping" (a standard SDRAM performs its operations on the rising edge of the clock signal only). Since they transfer data on both edges of the clock, the data transfer rate is doubled. To access the data at high rate, the memory cells are organized into two groups. Each group is accessed separately. DDR SDRAM is the current standard of RAM chip in PCs used at home. DDR1 SDRAM has been succeeded by DDR2, DDR3, DDR4 and most recently, DDR5 SDRAM. Although operating on the same principles,

CamScanner

the modules are not backward-compatible. Each generation delivers higher transfer rates and faster performance. The latest DDR5 modules, for example, feature fast transfer rates at 4.8 to 6.4 Gbps.

Personal computers come with different amounts of RAM. In many cases, additional RAM chips can be added by plugging a memory-expansion card into the motherboard. The more RAM you have, the faster the computer operates, and the better your software performs.

Differences between RAM and Cache

There are several key differences between RAM and cache memory.

Table 2.3. RAM and Cache Memory Compared

	RAM	Cache
What they are used for	Operating system, applications, data in use	Frequently used instructions and data
Speed	Fast	10 to 100 times faster
Capacity	More	Less
Cost	High	Higher

Usage: RAM provides short-term access to applications and files running in the background. Insufficient RAM will cause a system to slowdown data processing. But RAM is expensive and should be added only if the performance improvement justifies the cost. Cache serves recurring data to the CPU whenever it's requested. This includes data with common access patterns or calculations that would take too long to compute on a repetitive basis.

Capacity: Most modern computers are built with 64-bit computer architecture and 32 GB of RAM, with additional RAM available via expansion cards. Additional cache memory is available in capacities up to 512 KB.

CPU proximity: Comparing cache vs. RAM, both are situated near the computer processor. Both deliver high performance. Within the memory hierarchy, cache is closer and thus faster than RAM.

Cost: Cache is made of static RAM (SRAM) cells engineered with four or six transistors. SRAM is more expensive to manufacture than other types of computer memory and storage, including HDDs and SSDs.

Operations: Cache provides a direct memory interface to the CPU. RAM cells can be accessed directly in any order.

Performance: CPU cache is built into the CPU or on an adjacent chip. Its memory access time reportedly is 10 to 100 times faster than RAM, requiring only a few nanoseconds to respond to the CPU request. RAM data access is faster than secondary storage devices such as hard disks and SSDs.

How Data and Programs are Represented in the Computer

To understand how processing hardware works, you need to understand how data is represented electronically.

CamScanner

When you write a document, you use the letters A–Z (both upper- and lower-case); the numbers 0–9; and numerous punctuation marks and other special symbols, such as ?, #, and %. People understand what these characters mean; computers cannot. Computers deal with data converted into the simplest form that can be processed magnetically or electronically—binary form. The binary system consists of only two numbers—0 and 1.

People are used to counting in the decimal system (0, 1, 2, 3, 4, 5, 6, 7, 8, 9). The computer uses the binary system (0, 1) because it uses electricity. Thus, the 0 can be represented by the electricity being *off* and the 1 by the electricity being on. Or the 0 can be represented by the absence of a magnetic spot on a magnetic disk, and the 1 by the *presence* of a magnetic spot. (The same can be done with electricity being high voltage or low voltage, or with light being on or off, or with a hole being punched or not being punched in a card.) This on-off, present-absent, yes-no state of affairs is used to represent data throughout computers. For example, the letter E could be represented by electrical switches as 01000101 (or *off-on-off-off-off-on-off-on*).

Each 0 and 1 in the binary system is called a *bit* (for *bi*nary digi*t*). The bit is the basic unit for storing data in computer memory—0 means off, 1 means on. Notice that since a bit is always either on or off, a bit in computer memory is always storing some kind of data.

Since single bits by themselves cannot store all the numbers, letters, and special characters (such as # and ?) that must be processed by a computer, the bits are put together in a group called a byte (pronounced 'bite'). There are usually 8 bits in a byte (Fig. 2.8). Each byte usually represents one character of data—a letter, digit, or special character.

Computer manufacturers express the capacity of a computer to hold or store data in terms of the number of bytes or multiples of bytes. To summarize:

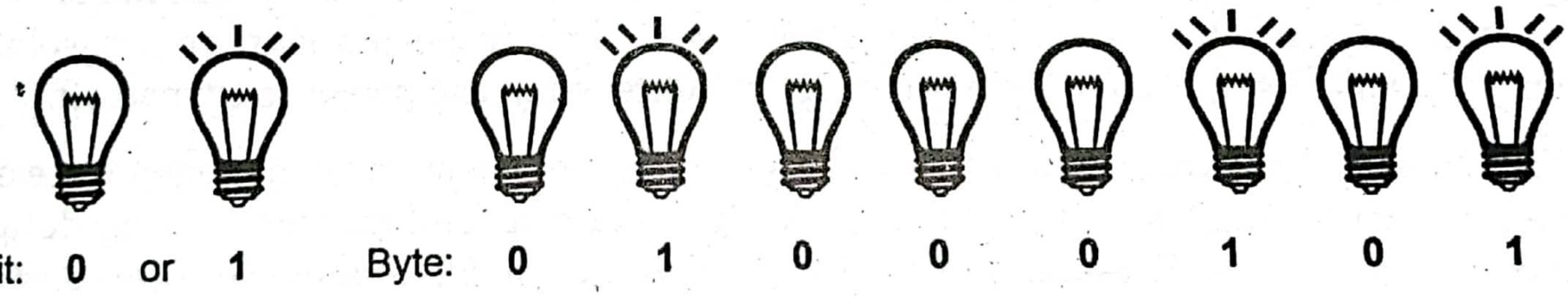


Fig. 2.8. Bit as light bulb. In this illustration a light bulb operates as a binary digit (bit), with *off* representing 0 and *on* representing 1. Light bulbs, of course, are not used in computers, but transistors, silicon chips, or anything else that can conduct an electrical signal can be used. The group of eight bulbs, each of which can be on or off, represents 1 byte.

Bit: Binary digit—0 or 1.

Nibble: Nibble is 4 binary digits or half of an 8-bit byte. A nibble can be conveniently represented by one hexadecimal digit.

Byte: 8 bits.

Kilobyte: *Kilo* represents 2 to the tenth power (2^{10}), or 1024. (This is commonly rounded to 1000 bytes.) Kilobyte is abbreviated KB or, simply, K. Thus, the memory of

CamScanner

a 640K in a computer can store 640×1024, or 655,360 bytes.

Megabyte: A *megabyte* is equal to 1024 KB or 1,048,576 bytes. (This is commonly rounded to 1 million bytes.) The megabyte was a common unit to measure memory or secondary-storage capacity on older computers.

Gigabyte: A *gigabyte* is equal to 1024 MB or 1,073,741,824 bytes. (This is commonly rounded to 1 billion bytes.) Many measures of personal computer capacity today are expressed in gigabytes.

Terabyte: A terabyte is equal to 1024 GB or 1,009,511,627,776 bytes. (This is commonly rounded to 1 trillion bytes.) The terabyte is a measure used with mainframe computers, supercomputers and now also with personal computers.

Petabyte: A *petabyte* is equal to about 1 quadrillion bytes (1,048,576 gigabytes). The huge storage capacities of modern databases are now expressed in petabytes.

Binary Coding Schemes

For information to be transferred into or out of a computer, numbers, letters, and several other symbols must be transformed into binary code. The commonly used coding schemes is ASCII.

ASCII (pronounced "ass-key") which stands for *American Standard Code for Information Interchange,* is used in virtually all personal computers and some larger computers as well.

The ASCII code was originally developed for electric teletypewriters but eventually found wide application in personal computers. The standard ASCII code uses seven-digit binary numbers; *i.e.,* numbers consisting of various sequences of 0s and 1s. The code can represent 128 different characters, since there are 128 different possible combinations of seven 0s and 1s.

Read-Only Memory (ROM)

Read-only Memory (ROM) is memory from which data may be retrieved, but the contents cannot be changed by the user. They store instructions in permanent, or non-volatile, form. Non-volatile storage does not lose its contents when the power is turned off.

ROM contains the programming that allows a computer to start up or regenerate each time it is turned on. Also, ROM chips typically contain special instructions for detailed computer operations. For example, UEFI (Unified Extensible Firmware Interface) chip fitted in your PC contains a set of programs that perform the most basic control and supervision operation for your PC and defines a software interface between an operating system and platform firmware. Other ROM instructions help the CPU transfer information from the keyboard to the screen or printer. ROMs are also called *firmware* because the instructions are "firm" and cannot be altered.

However, ROM is also utilized in video game consoles, allowing one system to run various games. Additionally, ROM is used in optical storage, including different kinds of compact discs (CD)—such as CD-ROM and CD-RW. ROM is also used frequently in calculators and peripheral devices like laser printers, whose fonts are commonly stored in ROM.

CamScanner

Types of ROM

Following are the types of read only memory:

1. **PROM (programmable ROM):** A variation of ROM chip is Programmable Read Only Memory (PROM). ROM chips are supplied by the computer system manufacturer, and it is not possible for a user to modify to programs stored inside the ROM chip. However, it is possible for a user to "customize" a system by writing his own programs and storing them in a PROM chip. However, once the chip has been programmed, the recorded information cannot be changed, *i.e.*, the PROM becomes a ROM and it is only possible to read the stored information. PROM is also non-volatile storage, *i.e.*, the stored information remains intact even if power is switched off.
2. **EPROM (erasable programmable ROM):** Once information is stored in a ROM chip or a PROM chip it cannot be changed or altered. However, there is another type of memory chip called Erasable Programmable Read Only Memory (EPROM) that overcomes this problem. As the name implies, it is possible to erase information stored in an EPROM chip and the chip can be reprogrammed to store new information. When an EPROM is in use, information can only be 'read' and the information remains on the chip until it is erased. EPROMs are mainly used by Research and Development (R & D) personnel because they frequently change the program written in EPROM to test the efficiency of the computer system in different atmospheres.
3. **EEPROM (electrically erasable programmable ROM):** Often used in older computer chips and to control BIOS, EEPROM can be erased by applying an electric field and reprogrammed several times. We can erase only portions of the chip too. Flash memory is an updated version of EEPROM that allows numerous memory locations to be changed at the same time.

Table 2.4. Difference between RAM and ROM

RAM	ROM
RAM is the volatile memory type capable of holding the data till there is a power supply.	ROM is a non-volatile memory type capable of retaining the data even when power is not supplied.
RAM's data can be altered or retrieved.	ROM's data can only be read.
This memory type stores the data that CPU process.	This memory type stores the instructions that are necessary during the computer's start up and basic control and supervision operation.
It is a high-speed memory unit.	ROM is much slower in operation than RAM.
CPU can easily access the data that RAM stores.	The CPU cannot easily access the data that ROM stores unless it gets transferred to RAM.
RAM has a larger size and more capacity.	ROM has a smaller size and lesser capacity.
RAM is used as a primary memory and CPU cache.	ROM is used as Microcontrollers and Firmware.
RAM is costly.	ROM is cheaper.

CamScanner

Advantages of ROM

ROM provides the necessary instructions for communication between various hardware components. As mentioned before, it is essential for the storage and operation of the UEFI, but it can also be used for basic data management, to hold software for basic processes of utilities and to read and write to peripheral devices.

Other advantages of ROM include:

1. Its static nature means it does not require refreshing.
2. It is easy to test.
3. ROM is more reliable than RAM since it is non-volatile in nature and cannot be altered or accidentally changed.
4. The contents of the ROM can always be known and verified.
5. Less expensive than RAM.

Expansion Cards and Slots

Computers are known for having different kinds of "architectures". Machines that have closed *architectures* are those manufactured in such a way that users cannot get inside to add any new devices. Most computers have *open architectures*. They allow users to expand their systems. This permits users to insert optional devices known as *expansion cards* (also known as *expansion boards* or *plug-in boards*) which give the computer additional options. Most motherboards in computers have expansion slots. *Expansion slots* are sockets on the motherboard into which expansion cards may be plugged. The slot connects the expansion card with the power supply and the buses for moving data and instructions. The following are some of the common expansion cards used in modern computers:

1. *Graphic Card:* When it comes to applications such as gaming and video editing, having a powerful dedicated graphics card is a must. This is why people still prefer to buy graphics cards.
2. *Network Interface Card (NIC):* The network interface card (NIC) comes in 2 flavors – Wired and wireless. Some modern motherboards already come with a built-in wired network port and wireless.
3. *Sound:* Users can add a sound card that can be used to improve sound input and output.
4. *Memory:* Memory expansion cards containing additional RAM chips will give the user additional main memory.
5. *Modem:* A *modem* changes the electronic signals of the computer into a form that can be sent over telephone lines. An *internal modem* is installed as an expansion card.

In order to make these expansion cards work, it is usually necessary to follow special *software* instructions that come with them.

There have been several types of expansion slots over the years, including PCI, EISA and VESA, but the most popular one used today is PCIe. While some newer computers still have PCI slot, PCIe has basically replaced all the older technologies.

CamScanner

Let's get to know the different types of slots on the motherboard.

PCI Slots: A PCI slot is an interface on the motherboard that allows you to put in a device that requires power, famous for coming with the first Intel Pentiums in 1993.

The PCI stands for *Peripheral Components Interconnect*, a standard computer bus to connect peripheral devices directly to your motherboard. Its name comes from its physical location between two other elements and from the electrical signals it sends and receives.

The devices installed on such PCI slots are very diverse: various controllers, network cards, modems, sound cards, Wi-Fi adapters, TV tuners, satellite receivers, etc. The number of available slots on motherboards ranges from two to six.

PCIe Slots: PCIe or PCI Express slots are the most common slots on the motherboard that optimize the communication with the microprocessor. The PCIe communicates with the motherboard, and therefore with the microprocessor, very quickly and efficiently.

It was a milestone in the history of computing, becoming a standard that has not yet been replaced but has been improved. It began to be used in 2005 developed by Intel, gradually replacing the rest of the expansion slot models.

When you need to mount a graphic card, or storage drive etc. based on a PCIe protocol, PCI Express Slots will be used for that purpose. There might be several numbers of PCIe slots on the motherboard.

EISA Slots: The EISA 'Extended Industry Standard Architecture' slots were introduced in 1988 by a group of PC manufacturers to compete with IBM.

The EISA bus provided 32-bit slots for use with 386DX or higher processors. EISA can also accommodate a 16-bit ISA card in the first row. EISA expansion slots are used to connect network cards, SCSI, and video cards.

VESA Slots: VESA stands for *Video-Electronics Standards Association*, it is a group of monitor and graphics card manufacturers that establish video bus and display standards. The aim of VESA slots is to optimize the video operations.

Although, VESA slots were only used in the PCs for a little amount of time. VESA company introduced these slots but after some time, ISA Bus replaced VESA.

Ports

Ports are connecting sockets on the outside of the system unit. This allows you to plug in other devices, such as keyboard, mouse, printer, display unit, monitor, flash drive and speaker.

In general, the computer ports classification can be done based on the type of communication like serial ports or parallel ports (see Table 2.5). Both serial ports and parallel ports are examples of computer technology that were once cutting edge; for most of the history of personal computers, both the serial and parallel port were the most common means of data transfer and communication. With technological advances, both the serial and parallel port have largely been replaced by the use of

CamScanner

USB ports, and fewer new devices are designed to include either a serial or parallel port.

Table 2.5. Differences between Serial Ports and Parallel Ports

	Serial Ports	Parallel Ports
Purpose	Serial Port is used for serial data transmission.	Parallel Port is used for parallel data transmission.
Transmission speed	Transmission speed of a serial port is slow as compared to a parallel port.	Transmission speed of a parallel port is very high as compared to a serial port.
No. of wires	Wire connections to serial port are quiet less as compared to parallel port.	No. of wires that are connected to parallel port are very high as compared to serial port.
Capability	A serial port can transmit a single stream of data at a time.	A parallel port can transmit multiple data streams at a time.
Data sending mechanism	A serial port sends data bit by bit after sending a bit at a time.	A parallel port sends data by sending multiple bits in parallel fashion.
Port type	A serial port uses Male ports.	A parallel port uses Female ports.
Applications	Modems, security cameras, device controllers used to have serial ports.	Printers, hard drives, CD drives used to have parallel ports.

Some common ports are Universal Serial Bus ports, USB-C ports, Ethernet ports or Display Ports. Some older ports no longer in common use are serial ports, parallel ports and mouse/keyboard PS/2 ports.

1. *USB / USB Type-A Ports:* A USB (Universal Serial Bus) port can connect up to 127 different peripherals together with a single connector. Devices that connect to a USB port include the following: printer, mouse, digital camera, speakers, scanner, portable media player, optical disc drive, smart phone, PDA, game console, and removable hard disk. The typical USB port is known as USB Type-A and has a simple, rectangular shape. Depending on the hardware, it can be capable of either USB-2.0 or USB-3.0 speeds. USB 3.0 has a maximum transfer rate of 5 Gbps, more than 10 times that of its predecessor, USB 2.0.

   The USB port allows peripheral devices to be connected or disconnected even while the PC is running. In addition, USB permits *plug and play,* which allows peripheral devices and expansion cards to be automatically configured while they are being installed. Today, most new computers feature at least four or six USB ports.

   Can you really connect up to 127 devices to a single port? An Intel engineer did set a world record at an industry trade show before a live audience by connecting 111 peripheral devices to a single USB port on a PC.

2. *USB Type-C:* This slim USB port is the connector of the future, already available on a number of devices and likely to replace USB Type-A, USB Type-B and microUSB on all new systems in the near future. Because it's much thinner than its predecessors, Type-C can fit on extremely slim laptops. It's also reversible, so

CamScanner

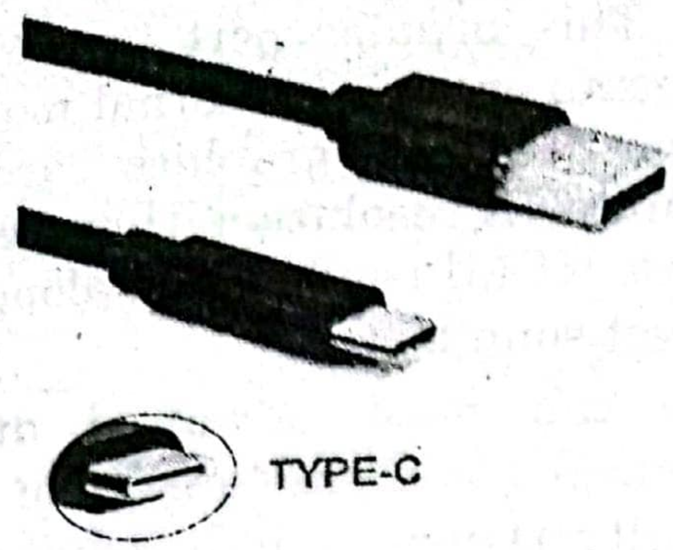


Fig. 2.9a. USB-C to USB-A converter

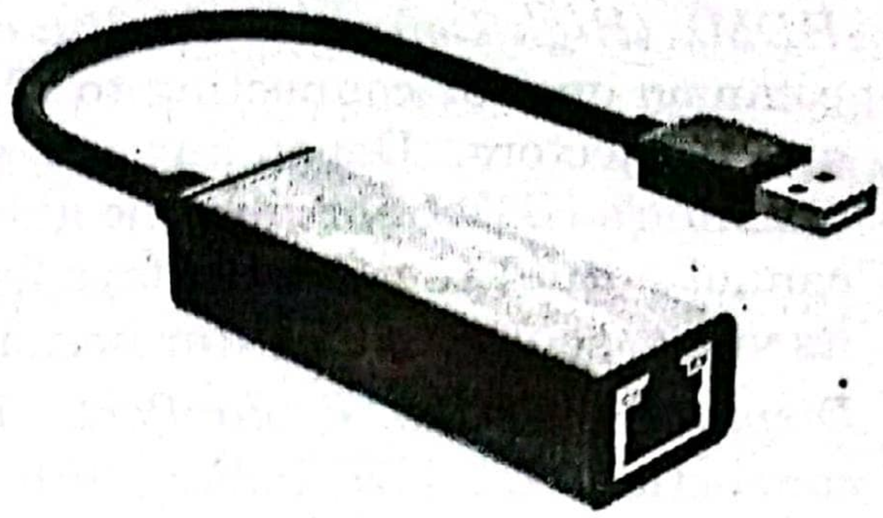


Fig. 2.9b. USB-to-Ethernet converter

you never have to worry about putting your plug in upside down. If you have a USB Type-C port and need to go to a USB Type-A dock or device, you can get a C to A wire (Fig. 2.9a).

USB Type-C ports can support a number of different standards, but not all of them offer the same functionality. Type-C can transfer files at either USB 3.1 gen 1 (5 Gbps) or USB 3.1 gen 2 (10 Gbps) speeds. It can accept USB Power Delivery (USB-PD) so you can charge your laptop with it. It also can even operate as a Thunderbolt port. If you have a rectangular USB Type-A port and need to go to a USB Type-C dock or device, you can get a A to C wire.

3. *Thunderbolt 3:* The fastest common connection on the market today, Thunderbolt 3 can transfer data at up to 40 Gbps, four times faster than the fastest USB connection (USB 3.1 gen 2). This high-speed standard can also output to up to two 4K monitors at once, because a single port carries dual DisplayPort signals. On several new devices you can use Thunderbolt 3 to connect to an external graphics card, which allows you to play high-end games on an otherwise slim laptop.

   All Thunderbolt 3 ports use USB Type-C connections and double as USB Type-C ports, allowing them to connect to an entire universe of USB peripherals and, in most cases, to charge a laptop or tablet.

4. *3.5mm Audio Jack:* Also known as headphone jack, it is the most common audio jack in the world. The 3.5mm audio jack appears on most computers, tablets and phones, and connects to the majority of the world's wired headphones and speakers. While some older devices have two audio jacks for mic and headphone separately, most current models incorporate both into the same port.

5. *Ethernet:* Also known as RJ-45, Gigabit Ethernet, or 10/100 Ethernet, found primarily on business laptops and on desktops, this port allows you to connect directly to wired networks. While WiFi continues to improve, having the option to plug in via Ethernet is really valuable in enterprise environment. If you don't have an Ethernet port built-in, you can purchase a USB-to-Ethernet converter (Fig. 2.9b).

6. *VGA (Video Graphics Array):* VGA dates all the way back to 1987 but is still commonly used on many monitors and projectors today. However, because the 15-pin connector is rather big in size, you won't find too many current-gen laptops or desktops that have VGA-out. It is an analogue connection, which could lead to signal degradation over longer cables, and it outputs at only up to 1920x1200.

CamScanner

7. *HDMI (High-Definition Multimedia Interface):* This popular port is the most common one for connecting to TVs and also appears on many external monitors and projectors. Depending on your laptop's or desktop's graphics card, the machine's HDMI port may be able to output at up to 4K resolution. However, you cannot output to dual displays from a single port. HDMI sends audio along with its video, so if your monitor has speakers, you'll get sound.
8. *DisplayPort / mini-DisplayPort:* DisplayPort is the most advanced display-connection standard today, with the ability to output to a single monitor at 4K resolution and 60 Hz, or up to three monitors at full HD (using a hub or dock). Most laptops that have DisplayPort either use the smaller, mini-DisplayPort connector or send their DisplayPort signals out of a USB Type-C port.

Cooling System

Computer cooling systems are passive or active systems that are designed to regulate and dissipate the heat generated by a computer so as to maintain optimal performance and protect the computer from damage that will occur from overheating.

If the CPU temperature is too high, for example, a mechanism will trigger that reduces performance in order to avoid damaging the processor. CPUs have very robust thermal management features that quickly adjust operating frequencies to reduce power when the system cooling solution is insufficient.

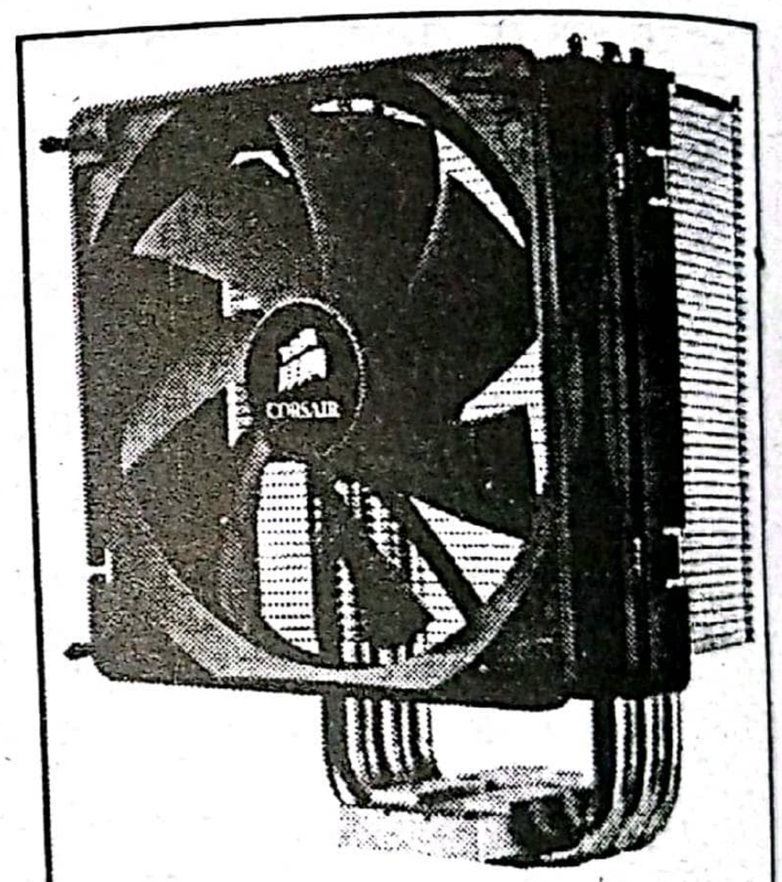


Fig. 2.10. Cooling system in a computer

This safety mechanism, sometimes referred to as *dynamic frequency scaling*, is useful in that it protects your processor from potential damage. However, this protection results in a performance cost when active. A better option is keeping the CPU cool enough that the mechanism doesn't trigger in the first place.

It is not just the CPU that requires temperature management. The GPU (graphics processing unit) is another critical component in a gaming PC and also needs to be adequately cooled. GPU cooling solutions are pre-installed and usually consist of fans installed on the graphics processor.

There are also many aftermarket solutions to keep a CPU or GPU cool, but by far the most popular solutions are *air* or *liquid* CPU coolers.

Air Cooling Systems: With *passive air cooling systems*, heat is distributed out to the surrounding air via cooling elements. To achieve higher performance, some cooling systems use heat pipes to send excess heat out of the PC (see Fig. 2.10). With *active air cooling systems*, the exhaust heat of the components is sent outside via an additional cooling element attached to the fan. A constant stream of air is produced in the computer case.

Laptops usually use sophisticated air cooling systems specifically designed for the smaller chassis and aren't normally designed to be upgraded or replaced.

CamScanner

Liquid Cooling Systems: The main advantage of liquid cooling systems is that exhaust heat is transported silently and efficiently outside the PC. The cooling performance is higher than that of air cooling systems, so this option is particularly useful for users wanting to overclock their computers. Overclocking is when a computer is operated with modified specifications, which are above the officially approved ones.

The most common liquid used during this cooling process is distilled water since water has a high specific heat capacity and also possesses thermal conductivity.

In most cases, the cooling element is made from aluminum or copper. In it, a pump moves the water in a circuit. The exhaust heat of the processor is integrated into this circuit and transported to the radiator. Here, the heat that has previously been cooled down through the water is distributed to the surrounding air.

As with air cooling, an active and a passive option exist with water cooling systems. With passive systems, the cooling process at the radiator happens through standard air movement. With the active option, a fan is responsible for producing the air stream.

CamScanner

Questions

2.1. What is the function of BIOS in a computer? List two main types of BIOS?
2.2. List and briefly describe the three components of the CPU. What is each part responsible for?
2.3. What is the function of the ALU in a personal computer system?
2.4. What are registers? Name some of the commonly used registers.
2.5. What is the main difference between a 66 GHz computer and a 133 GHz computer?
2.6. What is the function of primary storage, and how does secondary storage differ from it.
2.7. Describe why having more main memory, or RAM, in your computer (as opposed to less) is useful.
2.8. What is the difference between volatile and non-volatile storage? Give an example of each of these types of storage.
2.9. What is a cache memory? How is it different from a primary memory?
2.10. Differentiate between a bit, a byte, and a word.
2.11. How many bytes can be stored in the primary storage unit of a 8 megabyte computer?
2.12. What kinds of computers use ASCII coding?
2.13. Differentiate between RAM, ROM, PROM, EPROM and EEPROM.
2.14. Why do users use graphic card? Describe some commonly used cards.

Objectives and MCQs

Each of the following questions has four alternative answers, of which one or more may be correct. Pick up the correct alternative / alternatives.

2.1. The central processing unit is an example of
(a) software
(b) a program
(c) hardware
(d) an output unit

2.2. Which part of CPU performs all arithmetic and logic calculations on data it receives?
(a) Control unit
(b) Main Memory Unit
(c) ALU
(d) Read Only Memory

CamScanner

2.3. Which of the following can be output by a computer?
(a) graphics and text
(b) voice
(c) computer usable information
(d) All of the above

2.4. An example of peripheral equipment is the
(a) CPU
(b) spreadsheet
(c) printer
(d) personal computer

2.5. Which element is commonly used in the making of solid-state integrated circuits?
(a) Pentium
(b) lithium
(c) copper
(d) iron
(e) silicon

2.6. ASCII is the abbreviation of
(a) Application Software Code for Information Interchange
(b) American Standard Code for Information Interchange
(c) American System Code of Information Interchange
(d) American Symbolic Code of Information Interchange

2.7. ROM is the abbreviation of
(a) read-only memory
(b) random only memory
(c) read on memory
(d) all of the above

2.8. The memory that remains unaffected when the power is switched off is
(a) ROM
(b) RAM
(c) cache memory
(d) video memory

2.9. A group of 4 bits is called a
(a) byte
(b) word
(c) nibble
(d) all of the above

Objectives

2.1. Data means names, numbers, facts, or anything needed to work out a problem.
2.2. Input and output devices—as well as storage devices—are categorized as peripheral.

CamScanner

2.3. The term CPU is an abbreviation for central processing unit.
2.4. The small silicon chips that contain the CPU of a personal computer are called microprocessors.
2.5. Silicon is the semiconductor material commonly used to make integrated circuits (chips).
2.6. The bus links the CPU to every hardware device in the computer system.
2.7. The CPU's three components are the control unit, arithmetic/logic unit and registers.
2.8. The control unit is the CPU component that manages the processing of data.
2.9. The part of the CPU that does math and logic function is the arithmetic/logic unit.
2.10. During processing, data or instructions are held in temporary storage places called registers.
2.11. The power supply is located inside the system unit and converts AC current to DC current to run the computer.
2.12. The main circuit board in a system unit is called the motherboard.
2.13. The system clock is located inside the system unit and controls how fast all the operations within the computer take place.
2.14. If data is stored in volatile memory, it will be lost if the power goes off.
2.15. Is the data on an internal hard disk volatile or non-volatile? non-volatile
2.16. RAM stands for random access memory.
2,17. Is RAM volatile or non-volatile? volatile
2.18. ROM stands for read-only memory.
2.19. A kilobyte is 1024 bytes.

CamScanner

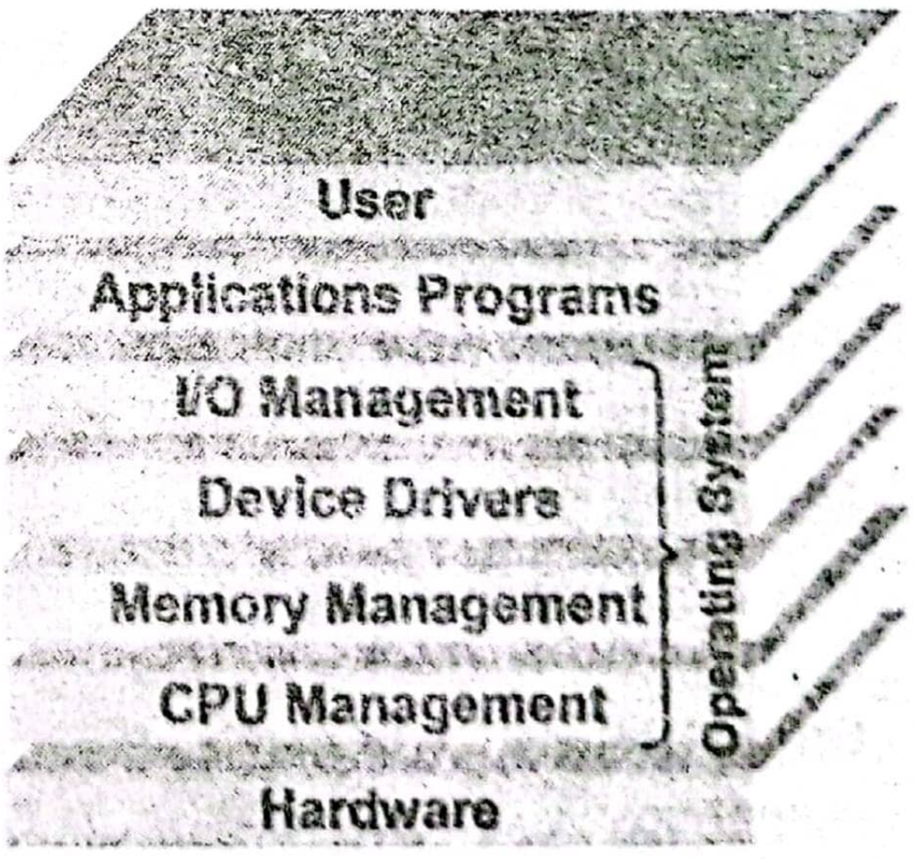
CHAPTER 3

OPERATING SYSTEM

By: Sir Ahad

It's important to realize that not all computers have operating systems. The computer that controls the microwave oven in your kitchen, for example, doesn't need an operating system. It has one set of relatively simple tasks to perform, very simple input and output methods (a keypad and an LCD screen), and simple, never-changing hardware to control. For a computer like this, an operating system would be unnecessary baggage, adding complexity where none is required. Instead, the computer in a microwave oven simply runs a single program all the time.

For computer systems that go beyond the complexity of the microwave oven, however, an operating system can be the key to greater operating efficiency and easier application development. All desktop computers have operating systems. The most common are the Windows family of operating systems, the Linux family of operating systems and the Macintosh operating systems. There are hundreds of other operating systems available for special-purpose applications, including specializations for mainframes, robotics, manufacturing, real-time control systems and so on.



At the simplest level, an operating system does two things:

1. It manages the hardware and software resources of the computer system. These resources include such things as the processor, memory, disk space, etc.
2. It provides a stable, consistent way for applications to deal with the hardware without having to know all the details of the hardware.

CamScanner

The first task, *managing the hardware and software resources*, is very important, as various programs and input methods compete for the attention of the central processing unit (CPU) and demand memory, storage and input/output (I/O) bandwidth for their own purposes. In this capacity, the operating system plays a supervisory role, making sure that each application gets the necessary resources while playing nicely with all the other applications, as well as managing the limited capacity of the system in a best way for all the users and applications.

The second task, *providing a consistent application interface*, is especially important if there is to be more than one of a particular type of computer using the operating system, or if the hardware making up the computer is ever open to change. A consistent *application program interface (API)* allows a software developer to write an application on one computer and have a high level of confidence that it will run on another computer of the same type, even if the amount of memory or the quantity of storage is different on the two machines. Even if a particular computer is unique, an operating system can ensure that applications continue to run when hardware upgrades and updates occur, because the operating system and not the application is charged with managing the hardware and the distribution of its resources. Microsoft Windows 11 is a great example of the flexibility an operating system provides. Windows 11 runs on hardware from thousands of vendors. It can accommodate thousands of different printers, disk drives and special peripherals in any possible combination.

What Is a User Interface?

A user interface is the combination of hardware and software that you use to communicate with and control the computer. Through the user interface, you are able to make selections on the computer, request information from the computer, and respond to messages displayed by the computer. Thus, a user interface provides the means for dialogue between you and the computer.

Hardware and software together form the user interface. Among the hardware devices associated with a user interface are the monitor, keyboard, mouse, and touchscreen. The *monitor* displays messages and provides information. You respond by entering data in the form of a command or other response using the *keyboard* or *mouse*. Touchscreens are displays that accept input by touch of fingers or a stylus. Among the responses available to you are responses that specify which application program to run, what document to open, when to print, and where to store data for future use.

The computer software associated with the user interface consists of the programs that engage you in dialogue. The computer software determines the messages you receive, the manner in which you should respond, and the actions that occur based on your responses.

The two broad categories of user interfaces supported by various operating systems are:

1. Command-line Interface
2. Graphical User Interface (GUI)

CamScanner

Command-line Interface

In a command-line interface, the user give instructions to the computer by typing commands. That is, to enter a command, the user uses the keyboard to type words and symbols. For example, in MS-DOS, typing

```
C:\>copy project.* d:
```

will copy all files named `project` (with any extension) in the current folder to the current folder on the drive D. The command prompt is the > character and the `C:\` is the pathname for the current folder. The user interface module contains a *command-line interpreter (CLI)* which performs the actual task of identifying and executing the command.

MS-DOS and Unix use an interface of this type.

Graphical User Interface

Graphical user interface or GUI (pronounced "gooey") allows the user to interact with the computer using graphics such as icons, menus, and dialogue boxes. Instead of typing a command, selection is achieved by moving a pointer with a mouse and clicking a mouse button.

GUIs are *user friendly*, which means the software can be used easily by individuals with limited training. Research studies have indicated that the use of graphics can play an important role in aiding users to interact effectively with a computer. The screen is arranged as a desktop with graphical symbols to represent familiar objects. Only valid options are available and there is a consistency of layout and command representation in applications which can be launched into execution through operation of the GUI. Comprehensive on-line help is available.

Examples of operating system that use a GUI include Windows 11, macOS, Chrome OS and Linux.

Disadvantages of a GUI over a command-line interface are:

1. They use more RAM and secondary storage capacity.
2. They require a more powerful processor and a better graphics display.
3. They are slower when executing a command because much more interpretation takes place.
4. They can be irritating to use for simple tasks because a greater number of operations is required.

Examples of Desktop Operating System

Common examples of desktop operating systems include the following:

MS-DOS and PC-DOS

During the 1980s and early 1990s, DOS (for Disk Operating System) was the dominant operating system for personal computers. Millions of copies of DOS were sold, making it one of the most widely used OS.

DOS was available in two forms: PC-DOS (Personal Computer Disk Operating System)

CamScanner

and MS-DOS (Microsoft Disk Operating System). Both were originally developed by Microsoft Corporation. Only minor differences separated these virtually identical operating systems. PC-DOS was created originally for IBM personal computers, whereas MS-DOS was designed for a whole group or class of computers, namely those said to be "IBM PC compatible". The last retail version was PC-DOS 2000, released in 1998, worked on both IBM personal computers as well as PCs made by other manufacturers. Microsoft still owns MS-DOS, but the firm no longer updates the program; Version 7 was the last Microsoft update.

DOS has a command-line interface. In a *command-line interface*, a computer user types commands or presses special keys on the keyboard to enter instructions and data. This kind of interface requires more accuracy and is more difficult to use than a graphical user interface.

DOS is a single-user, single-tasking operating system which is disk based. This operating system has a number of commands to make it user-friendly. The commands of DOS are divided into two groups namely, *internal commands* and *external commands*. Internal commands reside in memory whereas the external commands reside on disk and can be loaded when required. Examples of a few DOS commands are given here:

CLS	clears the screen.
COPY	copies a file.
DATE	reports the current date and allows user to change it.
DEL	deletes a file or multiple files.
DIR	lists all files on a disk.
DISKCOPY	copies all files from one disk to another.
EXIT	goes back in previous software.
FORMAT	prepares a disk for use.
REN	Renames individual files
TYPE	displays the contents of a file onto the screen.

Both the MS-DOS and PC-DOS operating systems were passed through many revisions since they were first developed. All versions starting with the number 1 (such as 1.0 and 1.1) were designed for the original IBM PC and other early PCs. Computers in 1990s used to run DOS Version 7 and later, with capabilities for managing hard disks and working on networks. Yet its original design based on 16-bit CPU chips—which supported only 1 megabyte of RAM—prevented DOS from taking full advantage of the power of newer Intel chips.

Unix

The UNIX (trademark) operating system was first developed by Bell laboratories, USA, in 1969. It is a family of multitasking, multiuser computer operating systems. Unix has a number of impressive features which have given rise to its widespread use. It is a full multitasking, multi-user time-sharing operating system written mostly in C, thereby making it relatively easy to port from machine to machine. It has a sophisticated and powerful command interpreter, called the *shell*, which is designed for used by computer professionals.

CamScanner

Unlike other operating systems—such as Windows 11 which are based on Intel, AMD or ARM chips—Unix is not built around a specific family of processors. Computer systems from micros to mainframes can run Unix and it can easily integrate a variety of devices from different manufacturers through network connections. This flexibility gives Unix a big advantage over competing operating systems for many types of applications. Unix operating systems are widely used in modern servers, workstations, and mobile devices.

But there are certain disadvantages of using Unix. Many people complain that is not very easy to use. What is more, the same features that give Unix its flexibility make it run more slowly than operating systems tailored around a particular family of microprocessors. This characteristic limit the benefits of applying Unix in an environment dominated by, say, Intel, AMD, or ARM CPUs. Perhaps the greatest disadvantage of Unix is that several "brands" of it are now available, many of them incompatible with each other.

Linux

A version of Unix that has achieved a loyal band of followers over the past few years is *Linux*. Originally created in 1991, Linux is a free, open-source operating system available over the Internet and is largely being improved by volunteers submitting new programming code over that same medium. Because of the dominance of the Linux kernel-based Android OS on smartphones, Linux has the largest installed base of all general-purpose operating systems. Linux is also the leading operating system on servers and other big iron systems such as mainframe computers. It is used by around 2.8% of desktop computers.

The development of Linux is one of the most prominent examples of free and open-source software collaboration. The underlying source code may be used, modified and distributed—commercially or non-commercially—by anyone under the terms of its respective licenses.

Some of the most popular and mainstream Linux distributions are CentOS, Debian, Fedora, Linux Mint, and Ubuntu, together with commercial distributions such as Red Hat Enterprise Linux. Distributions include the Linux kernel, supporting utilities and libraries.

macOS

macOS is a graphical operating system developed by Apple since 2001. It is the primary operating system for Apple's Mac family of computers. Within the market of desktop and laptop, it is the second most widely used desktop OS, after Microsoft Windows.

macOS is the second major series of Macintosh operating systems. The first is colloquially called the "classic" Mac OS, which was introduced in 1984, and the final release of which was Mac OS 9 in 1999. The first desktop version, Mac OS X 10.0, was released in March 2001, with its first update, 10.1. After this, Apple began naming its releases after big cats, which lasted until OS X 10.8 Mountain Lion. Since OS X 10.9 Mavericks, releases have been named after landmarks in California. Apple shortened the name to "OS X" in 2012 and then changed it to "macOS" in 2016, adopting the nomenclature that they were using for their other operating systems, iOS, watchOS,

CamScanner

and tvOS. The latest version is macOS Ventura, which was publicly released as version 13 in 2022.

macOS is based on technologies developed between 1985 and 1997 at NeXT, a company that Apple co-founder Steve Jobs created after leaving Apple. The "X" in Mac OS X and OS X is the Roman numeral for the number 10 and is pronounced as such. The X was a prominent part of the operating system's brand identity and marketing in its early years.

macOS has supported three major processor architectures, beginning with PowerPC-based Macs in 1999. In 2006, Apple transitioned to the Intel architecture with a line of Macs using Intel Core processors. In 2020, Apple began the Apple silicon transition, using self-designed, 64-bit ARM-based Apple M series processors on the latest Macintosh computers.

Microsoft Windows

The Windows operating system was developed by the Microsoft corporation in 1985. It was a graphical user interface (GUI) that gave MS-DOS-based computers many of the same capabilities of the Macintosh. Microsoft expanded Windows to Windows NT (New Technology) in 1993 so that it can handle UNIX in addition to MS-DOS. Windows NT was a powerful multi-user, multitasking system.

In 1990 Microsoft corporation introduced Windows 3.0, a revised and improved version of a program that had been introduced in 1985. Windows 3.0 permitted PC machines to be operated by means of a "mouse" and on-screen symbols instead of by typed commands. By 1995, with the introduction of new version of Windows called Windows 95, and its subsequent versions (*e.g.*, Windows 98, Me, XP, Vista, 7, 8, 8.1, 10, and 11), the Windows became the dominant operating environment for personal computers. Nowadays this operating system is running in nearly 75 percent of the world's personal computers.

Windows 11 is a personal computer operating system developed and released by Microsoft, as part of the Windows NT family of operating systems. It was released in 2021. Windows 11 is often described by Microsoft as a being as a "service", as it receives regular feature updates that contain new features and other changes.

Windows 11 features major changes to the Windows shell, including a redesigned Start menu, a separate "Widgets" panel on the taskbar—the ability to create tiled sets of windows that can be minimized and restored from the taskbar as a group, and new gaming technologies inherited from Xbox consoles such as Auto HDR and DirectStorage on compatible hardware. Internet Explorer (IE) has been replaced by the Chromium-based Microsoft Edge as the default web browser like its predecessor, Windows 10. Microsoft Teams is now integrated into the Windows shell and accessible from Taskbar, making it easier to access. You can access Teams from Windows, Mac, Android or iOS. Microsoft also announced plans to allow more flexibility in software that can be distributed via Microsoft Store, and to support Android apps on Windows 11 (including a partnership with Amazon to make its app store available for the function).

For tablets, Windows 11 improved the experience for touch, with more space between

CamScanner

icons on the Taskbar, and support for gestures. Windows 11 also added haptics to the digital pen, so user can hear and feel vibrations as you use it for taking notes or drawing. Finally, the OS introduces voice typing and commands across the system.

Citing security considerations, the system requirements for Windows 11 were increased over Windows 10. Microsoft only officially supports the operating system on devices using an eighth-generation Intel Core CPU or newer (with some minor exceptions), AMD Ryzen CPU based on Zen+ microarchitecture or newer, or a Qualcomm Snapdragon 850 ARM system-on-chip or newer, with UEFI secure boot and Trusted Platform Module (TPM) 2.0 supported and enabled. While the OS can be installed on unsupported processors, Microsoft does not guarantee the availability of updates. Windows 11 removed support for 32-bit x86 CPUs and devices which use BIOS firmware.

Features of an Operating System

Following are the most important features of an Operating System:

1. Batch Processing
2. Multitasking
3. Time-Sharing
4. Multiprocessing
5. Parallel Processing
6. Distributed System
7. Embedded Operating System

Batch Processing

Batch processing is a technique in which an operating system collects the programs and data together in a batch before processing starts (see Fig. 3.1). Main motive of using batch processing is to decrease the set-up time while submitting the similar jobs to CPU.

Batch processing put less strain on the CPU and require less user interaction. Another advantage of batch processing is that large repetitive tasks can be completed without interacting with the computer to notify the system that you need to perform another task after you finish the current one.

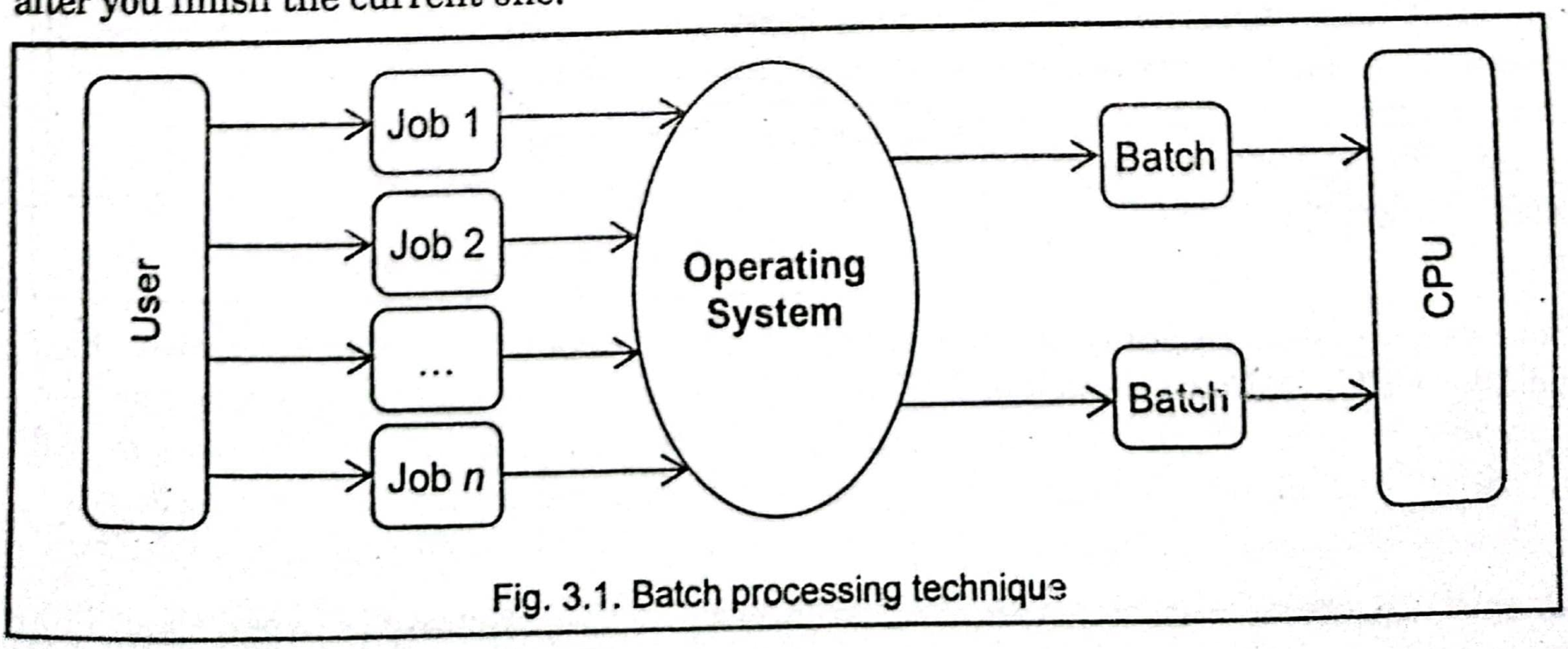


Fig. 3.1. Batch processing technique

CamScanner

Characteristics of Batch Processing: The characteristics of the batch processing are as follows:

1. In batch processing, jobs that have similar resource requirements are grouped together, known as a *batch*.
2. These jobs are executed based on a first-come, first-serve basis. It totally depends on which job has been sent first by the user to the CPU.
3. When a task is successfully executed, the OS releases the memory space held by that job.

Examples of Batch Processing

1. *Payroll System:* Batch processing is more suitable to payroll, for calculating the salaries of all employees in the end of month.
2. *Bank Invoice System:* Every end of the month, all banks make the monthly statement for all account holders. So, by using batch processing, banks produce monthly statements of all the clients.
3. *Daily Report:* In the manufacturer industries, every day need operational statement for production line. The batch processing is used for generating the daily report for maintaining the records.
4. *Billing System:* Every telecom organization uses the batch processing for processing millions of calls and their detail records including calculate rates.

Multitasking

Multitasking is when multiple jobs are executed by the CPU simultaneously by switching between them. Switches occur so frequently that the users may interact with each program while it is running (see Fig. 3.2). The operating system is able to keep track of what the user is doing and is able to navigate from one application to another without losing information.

Almost all operating systems nowadays perform multitasking. Users of these operating systems are always in need of the system to carry out multitask operations—for example, a simple operation that requires multitasking is using a 'Word document' and a 'Web browser' at the same time.

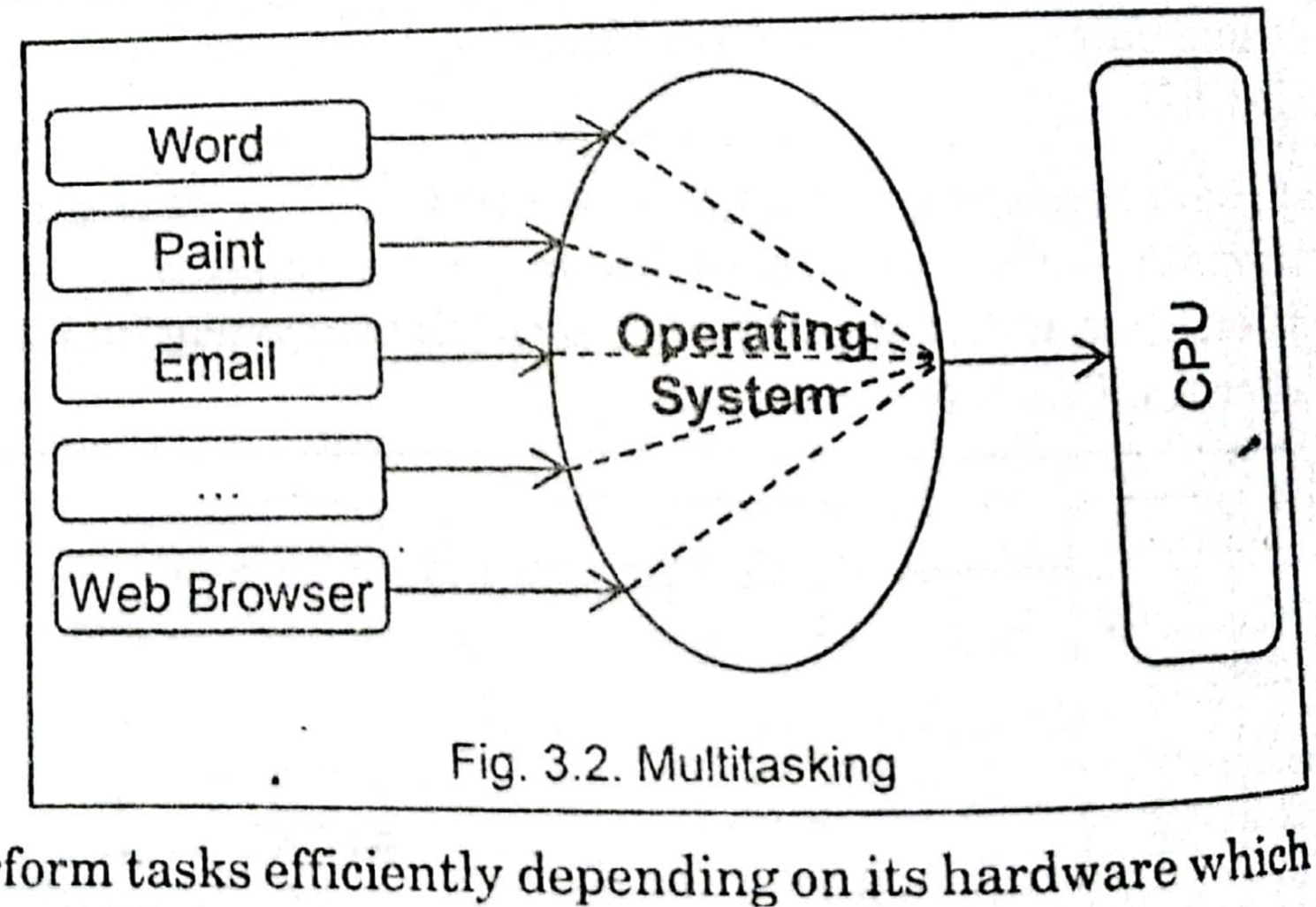


Fig. 3.2. Multitasking

However, a system can only perform tasks efficiently depending on its hardware which includes CPU, RAM, hard disk or SSD etc. This is because each task consumes system storage and other resources, therefore as more tasks are initiated, the system will prioritize some of that storage to that application, thus decreasing the overall efficiency of all the other applications, as it portioned out some run time for the new application.

An operating system works on more than one task, by switching between them in a

CamScanner

very quick manner. Tasks may concern a single user or multiple users, the operating system will ultimately save the current state of each user and task so that it does not lose its place when it comes back to a task to resume the execution. This allows for efficiency and allows the system to switch smoothly between tasks.

An OS does the following activities related to multitasking:

1. The user gives instructions to the operating system or to a program directly and receives an immediate response.
2. The OS handles multitasking in the way that it can handle multiple operations/executes multiple programs at a time.
3. All tasks are allotted a specified amount of time so that they do not have to wait for the CPU.
4. Each user has at least one separate program in memory.
5. A program that is loaded into memory and is executing is commonly referred to as a process.
6. When a process executes, it typically executes for only a very short time before it either finishes or needs to perform I/O.
7. As the system switches CPU rapidly from one user/program to the next, each user is given the impression that he/she has his/her own CPU, whereas actually one CPU is being shared among many users.

Types of Multitasking

There are mainly two types of multitasking. These are as follows:

1. Preemptive Multitasking
2. Cooperative Multitasking

Preemptive Multitasking: *Preemptive multitasking* is a special task assigned to a computer operating system. It decides how much time one task spends before assigning another task to use the operating system. Because the operating system controls the entire process, it is referred to as *preemptive*.

Unix was the first operating system to use this method of multitasking. Windows NT and Windows 95 were the first versions of Windows that use preemptive multitasking.

Cooperative Multitasking: The main purpose of cooperative multitasking is to run the present task while releasing the CPU to allow another process to run.

Windows 11 and MacOS used cooperative multitasking. A Windows program will respond to a message by performing some short unit of work before handing the CPU over to the operating system until the program receives another message.

Time-Sharing

Time-sharing is a technique which enables many people, located at various terminals, to access the same computer system simultaneously. In the time-sharing technique, many processes are assigned to computer resources in respective time slots. In this scenario, the processor's time is shared with multiple users. That is why it is called a *time-sharing*. As the system rapidly switches from one user to another, each user is

CamScanner

given the impression that the entire computer system is dedicated to its use, although it is being shared among multiple users.

The CPU executes multiple jobs by switching between them, but the switches occur so frequently. Thus, the user can receive an immediate response. The operating system uses CPU scheduling and multitasking to provide each user with a small amount of time.

A *time-shared* technique uses CPU scheduling to provide each user with a small portion of a shared computer at once. Each user has at least one separate program in memory. A program loaded into memory and executes, it performs a short period of time either before completion or to complete I/O. This short period of time during which user gets attention of CPU is known as *time slice, time slot* or *quantum*. It is typically of the order of 10 to 100 milliseconds. In time sharing process, multiple jobs must be kept in memory simultaneously, so the system must have memory management and security. In Fig. 3.3, the user 5 is *active state* but user 1 to user 4 are in *waiting state* whereas user 6 is in *ready state*.

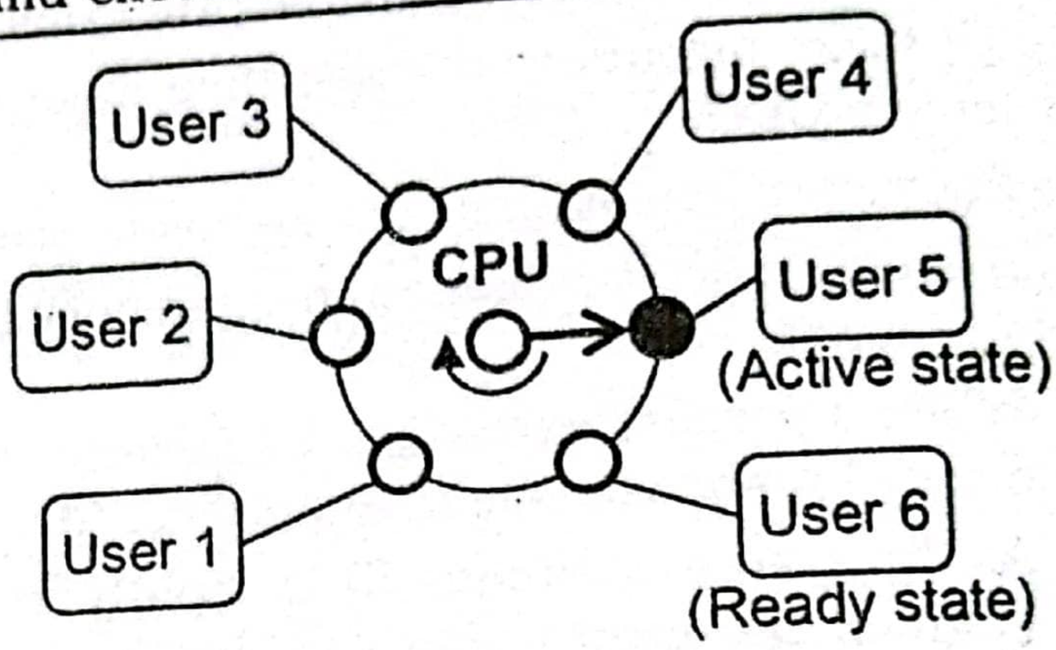


Fig. 3.3. Time Sharing Operating System

1. *Active State:* The user's program is under the control of CPU. Only one program is available in this state.
2. *Ready State:* The user program is ready to execute but it is waiting for its turn to get the CPU. More than one user can be in ready state at a time.
3. *Waiting State:* The user's program is waiting for some input/output operation. More than one user can be in a waiting state at a time.

Features of Time-Sharing Technique

Time-sharing technique provides the following features for users:

1. Each user grabs dedicated time for all operations.
2. Multiple online users can use the same computer at the same time.
3. Better interaction between users and computers.
4. User requests can make in small-time responses.
5. It does not need longer have to wait for the last task to end to get processor.
6. It can make quick processing with a lot of tasks.

Multiprocessing

Multiprocessing refers to the ability of an operating system to support two or more CPUs or cores (*e.g.*, dual-core and quad-core processor) at the same time within a single computer system. Multiple CPUs or cores are interconnected so that a job can be divided among them for faster execution. When a job finishes, results from all processing units are collected and compiled to give the final output. CPUs or cores in a multiprocessing mode share main memory and peripherals, in order to

CamScanner

simultaneously process programs as shown in Fig. 3.4.

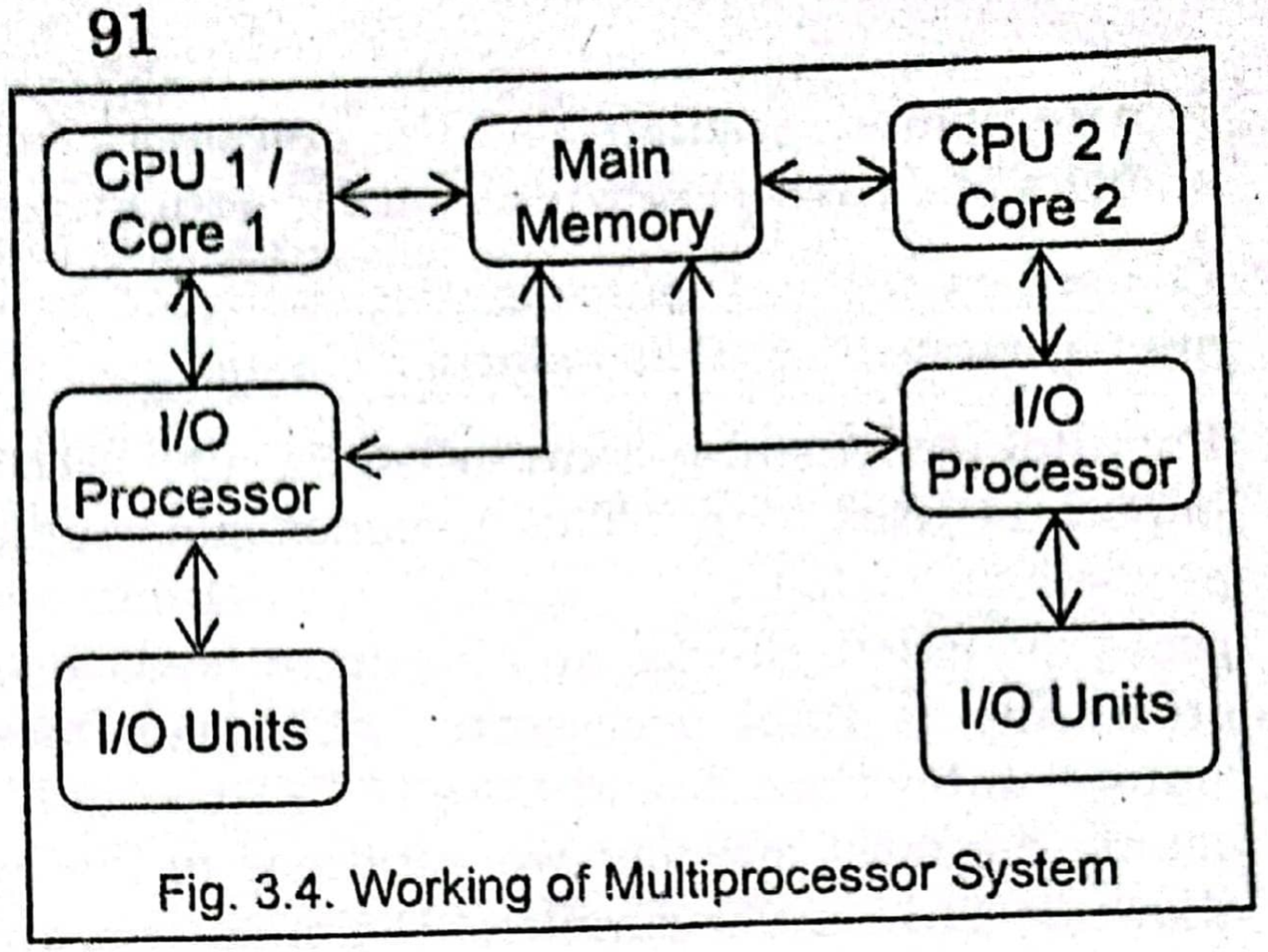


Fig. 3.4. Working of Multiprocessor System

Multi-core processors are chips that contain two or more processors for better performance, reduced power consumption and more efficient processing of multiple tasks. These multi-core set-ups are similar to having multiple, separate processors installed in the same computer. Most computers may have anywhere from two to four cores; increasing up to 12 cores.

At the operating system level, multiprocessing is sometimes used to refer to the execution of multiple concurrent processes in a system, with each process running on a separate CPU or core, as opposed to a single process at any one instant. Multiprocessing doesn't necessarily mean that a single process or task uses more than one processor simultaneously; the term *parallel processing* is generally used to denote that scenario.

Multiprocessing systems are used when very high speed is required to process a large volume of data. To employ a multiprocessing feature effectively, the computer system must have the following things:

1. A motherboard which is capable of handling multiple processors and/or single processor with multiple cores.
2. Processors used should also be capable of handling multiprocessing.
3. An environment using multiprocessing supportive operating system (all modern operating systems support multiprocessing).

Parallel Processing

Parallel processing is a computing technique to speed up the execution of programs or data processing tasks by dividing the task into multiple streams of tasks and processing these streams of tasks simultaneously through multiple central processing units (CPUs) or cores. Breaking up different parts of a task among multiple processors will help reduce the amount of time to run a program. Any system that has more than one CPU can perform parallel processing, as well as multi-core processors which are commonly found on computers today.

Typically, a computer scientist will divide a complex task into multiple parts with a software tool and assign each part to a processor, then each processor will solve its part, and the data is reassembled by a software tool to read the solution or execute the task.

Typically, each processor will operate normally and will perform operations in parallel as instructed, pulling data from the computer's memory. Processors will also rely on software to communicate with each other so they can stay in sync concerning changes

CamScanner

in data values. Assuming all the processors remain in sync with one another, at the end of a task, software will fit all the data pieces together.

Computers without multiple processors can still be used in parallel processing if they are networked together to form a cluster.

Parallel Processing Examples: Parallel processing or parallel computing has many important uses today. This includes supercomputers for use in astronomy, making predictions in agriculture, risk calculations and cryptocurrencies in banking, video post-production effects, and accurate medical imaging. Another example of parallel processing is Intel processors, which run most high-power modern desktops and laptops. Intel Core i5 and Core i7 CPUs each have four processing cores. The HP Z8, among the most efficient workstations in the industry, contains 56 computing cores, allowing it to handle complex 3D simulations and real-time 8K video editing.

Distributed System

Distributed system comprises several software components that reside on multiple independent computer systems but operate as a single system. A distributed system's computers can be physically close together and linked by a local network or geographically distant and linked by a *wide area network (WAN)*. A distributed system can be made up of any number of different configurations, such as mainframes, workstations, and PCs. The main aim of distributed computing is to make a network work as a single computer.

An operating system does the following activities related to distributed environment:

1. The OS distributes computation logics among several independent computer systems.
2. The processors do not share memory or a clock. Instead, each processor has its own local memory.
3. The OS manages the communications between the computer systems. They communicate with each other through various communication lines.

Some features of a distributed system are:

1. The system has independent failure modes. It means that the system components may fail without affecting the rest of the system.
2. All the components of the distributed system work at the same time.
3. Each system component has its local clock. All the clocks of the different system might not record the same time. Also, the clock in each hardware may run at different rates. Such a feature is called *Clock Drift* and only occurs in the case of distributed systems.
4. The most powerful feature of a distributed system is scalability. The operating system supports growth and can be used for industrial purposes.

There are various benefits of using distributed computing. It enables scalability and makes it simpler to share resources. It also aids in the efficiency of computation processes.

CamScanner

Table 3.1. Comparison between the Parallel Computing and Distributed Computing

	Parallel Computing	Distributed Computing
Definition	It is a type of computation in which various processes runs simultaneously.	It is that type of computing in which the components are located on various networked systems that interact and coordinate their actions by passing messages to one another.
Communication	The processors communicate with one another via a bus.	The computer systems connect with one another via a network.
Functionality	Several processors execute various tasks simultaneously in parallel computing.	Several computers execute tasks simultaneously.
Number of Computers	It occurs in a single computer system.	It involves various computers.
Memory	The system may have distributed or shared memory.	Each computer system in distributed computing has its own memory.
Usage	It helps to improve the system performance	It allows for scalability, resource sharing, and the efficient completion of computation tasks.

Embedded Operating System

An embedded operating system is a specialized operating system (OS) designed to perform a specific task for a device that is not a computer. The main job of an embedded OS is to run the code that allows the device to do its job. The embedded OS also makes the device's hardware accessible to software that is running on top of the OS.

Examples of an embedded OS include computer systems in cars, traffic lights, digital televisions, ATMs, airplane controls, point of sale (POS) terminals, digital cameras, GPS navigation systems, elevators and Smart meters.

Networks of devices containing embedded systems make up the internet of things (IoT). The embedded systems perform basic operations inside IoT devices, such as transferring data over a network without human interaction.

In contrast to an OS for a general-purpose computer, an embedded OS has limited functionality. Depending on the device in question, the system may only run a single embedded application. However, that application is likely crucial to the device's operation. Given that, an embedded OS must be reliable and able to run with constraints on memory and processing power.

In the case of a Raspberry PI system on a chip, an SD card acts as the device's hard drive and contains the code that runs on the device. The SD card is removable, so its contents can be modified on demand. The embedded OS makes the device's hardware—such as USB and HDMI ports—accessible to the application running on top of the OS.

CamScanner

Embedded versus non-embedded OSes: What is the Difference?

An embedded OS may reside on a chip within an electronic device. They are often limited in the scope of what they can do. In contrast, a non-embedded OS runs from a hard disk or a solid-state drive. Non-embedded OSes, such as Windows 11 or MacOS, are configurable and upgradable. They are designed for general-purpose use.

Another difference between embedded and non-embedded OSes is in how the operating system is coded. Embedded OSes are usually contained in a single executable image and execute one task. Desktop operating systems and network operating systems contain many applications.

Embedded OSes also have a minimal or no user interface. Non-embedded OSes, on the other hand, have a more dynamic user interface.

Functions of the Operating System

An operating system is a piece of software that manages the allocation of computer hardware. The coordination of the hardware must be appropriate to ensure the correct working of the computer system and to prevent user programs from interfering with the proper working of the system.

The main task an operating system carries out is the allocation of resources and services, such as the allocation of memory, devices, processors, and information. The operating system also includes programs to manage these resources, such as a traffic controller, a scheduler, memory management module, I/O programs, and a file system.

Main functions of an operating system are:

1. **Booting:** Booting is the process which is initiated once the computer system starts and executes small set of instructions present in the ROM which will setup system hardware by testing them and load the operating system so that computer system can carry out further tasks.

   There are two types of Booting available: *Cold booting* is the process when our computer system moves from shut down state to the start by pressing the power button. The system reads the BIOS from ROM and will eventually load the Operating System. *Warm booting* is the process in which the computer gets restart due to reasons like setting the configuration for newly installed software or hardware. Warm booting is called as *rebooting*.

2. **Processor Management:** Every software that runs on a computer, whether in the background or in the frontend, is a process.

   The OS decides the order in which processes have access to the processor, and how much processing time each process has. This function of OS is called *process scheduling*. An Operating System performs the following activities for processor management.

   (a) Keeps track of the status of processes. The program which performs this task is known as a *traffic controller*.
   (b) Allocates the processor to a process.
   (c) De-allocates the processor after the process is completed.

CamScanner

3. **Memory Management:** The operating system manages the Main Memory or RAM. Main memory is made up of a large array of bytes or words where each byte or word is assigned a certain address. Main memory is fast storage and it can be accessed directly by the CPU. For a program to be executed, it should be first loaded in the main memory. An Operating System performs the following activities for memory management:
   (a) It keeps track of primary memory, *i.e.*, which bytes of memory are used by which user program. The memory addresses that have already been allocated and the memory addresses of the memory that has not yet been used.
   (b) It Allocates the memory to a process when the process requests it and deallocates the memory when the process has terminated or is performing an I/O operation.
   (c) In multitasking, the OS decides the order in which processes are granted access to memory, and for how long.
4. **File Management:** The Operating System is responsible for making decisions regarding the storage of all types of files on various storage devices and the transfer of these files from one storage device to another. The Operating System decides how the data should be manipulated and stored.

   A file system is organized into directories for efficient or easy navigation and usage. These directories may contain other directories and other files. An Operating System carries out the following file management activities: It keeps track of where information is stored, user access settings and status of every file, and more. These facilities are collectively known as the *file system*.
5. **Input/Output Management:** The Operating System is responsible for handling all sorts of input devices, *e.g.*, keyboard, mouse, touchscreen, *etc.* and output devices, *e.g.*, softcopy or hardcopy output devices. The Operating System does all interfacing in the most appropriate manner regarding all kinds of inputs and outputs. For example, what output will come from the input given by the computer operator using devices such as mouse, keyboards or touchscreen. This management involves coordinating various input and output devices. It assigns the functions of those devices where one or more applications are executed.
6. **Secondary Storage Management:** The operating system manages storage devices such as magnetic, flash or optical disks. It decides where specific data will be stored and retrieved from the device. It helps store and retrieves data from the secondary storage. It allocates file to free spaces on the disk and ensure overall management. The creation of files, the creation of directories, the reading and writing of data from files and directories, as well as the copying of the contents of files and directories from one location to another are all included in storage management.

   The OS uses storage management for:
   (a) Improving the performance of the data storage resources.
   (b) It optimizes the use of various storage devices.
   (c) Free space management.
   (d) Assists businesses in storing more data on existing hardware, speeding up the data retrieval process, preventing data loss, meeting data retention regulations, and lowering IT costs.

CamScanner

7. **Network Management:** Network management is the process of administering, managing, and operating a computer network. A modern OS contains built-in software designed to simplify networking. Typical OS software includes an implementation of TCP/IP and related utility programs such as ping and traceroute, along with device drivers and other software to automatically enable the Ethernet or wireless interface for a device. Another common example of computer networking is a distributed system in which a server OS can ensure that there is uniformity between the distributed PCs. In such a system, a large number of independent and physically separated computers can connect through a single communication channel, and this is supported by the operating system.

8. **Security Management:** One of the important responsibilities of the operating system is to provide secure and protected runtime environment to the various process (both application process and system processes) which are being executed on the computer system. The operating system provides various techniques which assure the integrity and confidentiality of user data and secure environment. Following security measures are used to protect system against unauthorized access:
   (a) Protection against unauthorized access through login (*e.g.*, username and a password).
   (b) Creating secure accounts with required privileges only (*i.e.*, user management)
   (c) Protection against intrusion by keeping Firefall active.
   (d) Protecting the system memory against malicious access.
   (e) Displaying messages related to system vulnerabilities.

9. **Command Interpreter:** A command interpreter is the part of a computer operating system that reads the instructions given by the user. The command interpreter then translates these instructions into the context of the OS followed by its execution. The Command interpreter is also known as *shell*.

   A command interpreter is an interface between system and the user. There are two types of user interface.
   (a) Command Line Interface.
   (b) Graphical User Interface (GUI).

   Command Line: With a command line user interface, the user interact with the Operating System by typing commands to perform specific tasks.

   GUI Interface: With a graphical user interface (GUI), the user interacts with the Operating System by using a mouse or a touchscreen to access windows icons and menus.

Process Management

Program vs. Process: A process is a program in execution. For example, when we write a program in C++ and compile it, the compiler creates binary code. The original code and binary code are both programs. When we execute the binary code, it becomes a *process*. When a program is loaded into the memory and becomes a process, it can be divided into four sections—stack, heap, data and text. Fig. 3.5 shows a simplified layout of a process inside main memory.

CamScanner

1. *Stack:* The process Stack contains the temporary data such as method/function parameters, return address and local variables.
2. *Heap:* This is dynamically allocated memory to a process during its run time.
3. *Data:* This section contains the global and static variables.
4. *Text:* This includes the current activity represented by the value of Program Counter and the contents of the processor's registers.

A process is an *active* entity whereas a program is considered as a *passive* entity. A single program can create many processes when run multiple times; for example, when we open a .exe or binary file multiple times, multiple instances begin (multiple processes are created).

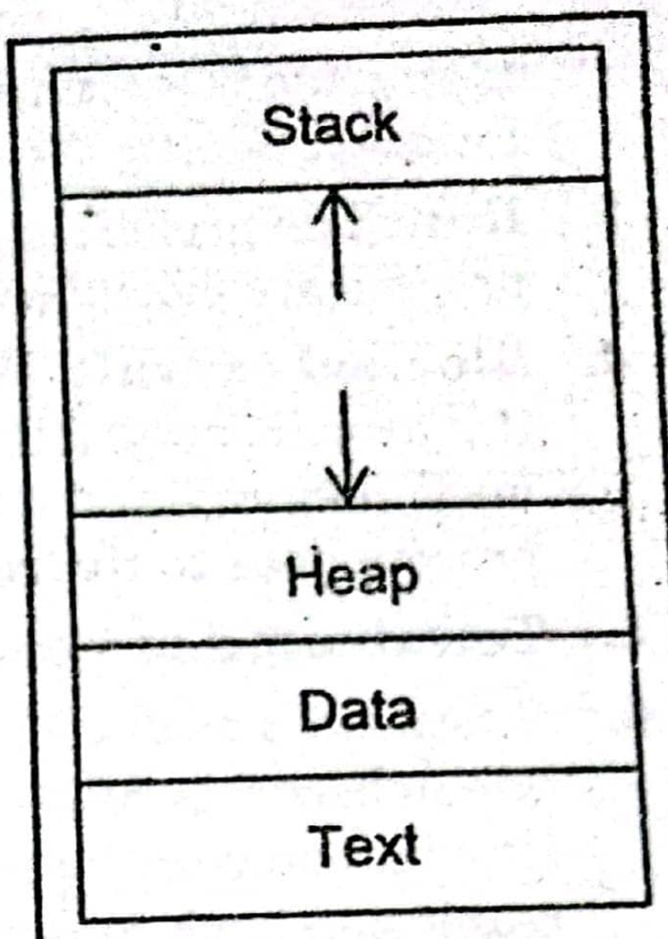


Fig. 3.5. A typical memory layout of a running process

States of Process in Operating Systems

When a process executes, it passes through different states. These stages may differ in different operating systems, and the names of these states are also not standardized. In general, a process can have one of the following seven states at a time (see Fig. 3.6).

1. **New (Create):** In this step, the process is about to be created but not yet created, it is the program which is present in secondary memory that will be picked up by OS to create the process.
2. **Ready:** Ready to run. After the creation of a process, the process enters the ready state, *i.e.*, the process is loaded into the main memory. The process here is ready

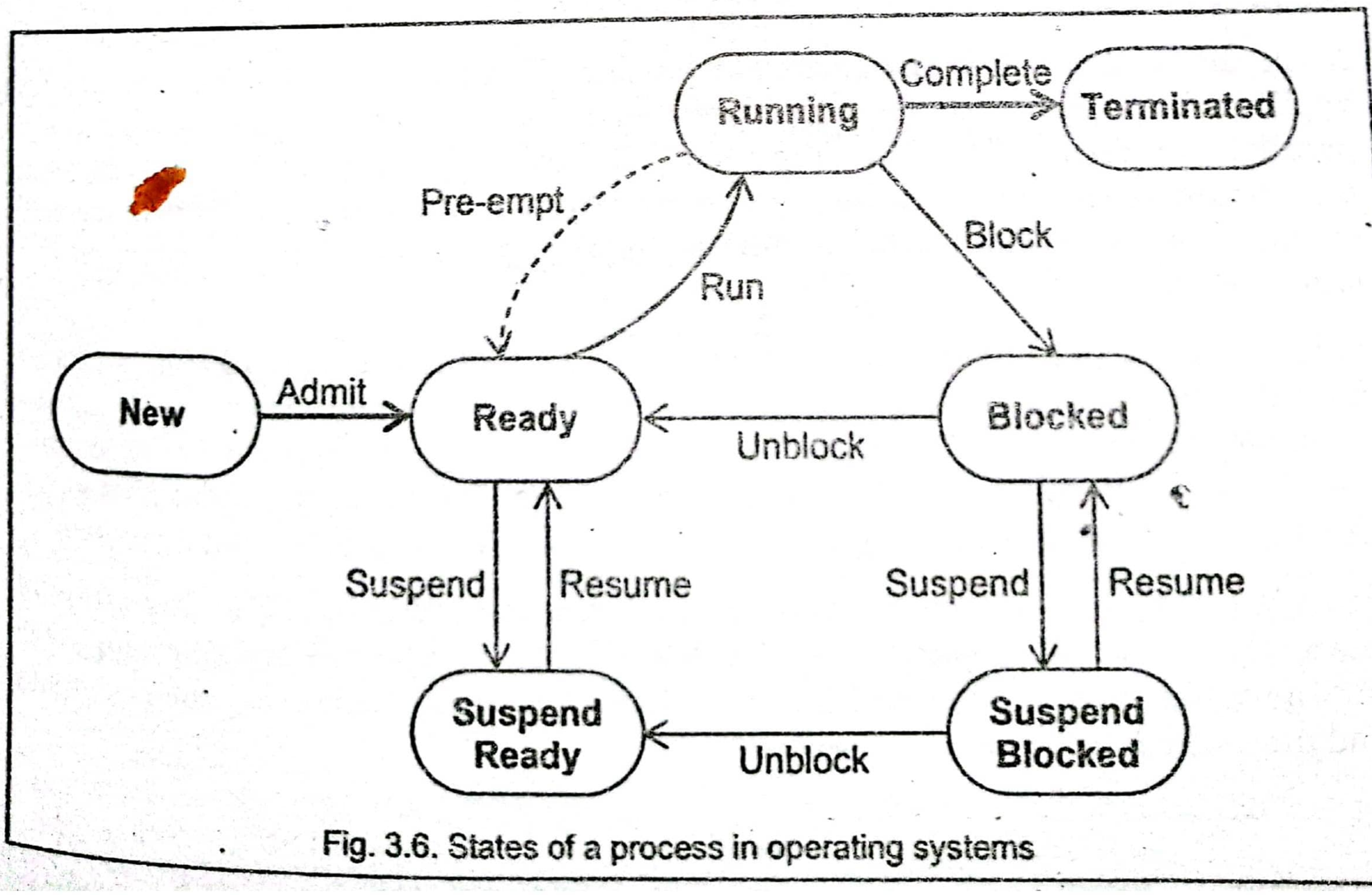


Fig. 3.6. States of a process in operating systems

CamScanner

to run and is waiting to get the CPU time for its execution. Processes that are ready for execution by the CPU are maintained in a queue for ready processes.

3. **Run:** The process is chosen by CPU for execution and the instructions within the process are executed by any one of the available CPU's cores.
4. **Blocked or wait:** Whenever the process requests I/O access or needs input from the user, it enters the blocked or wait state. The process continues to wait in the main memory and does not require CPU. Once the I/O operation is completed the process goes to the ready state.
5. **Terminated or completed:** The process completed its execution.
6. **Suspend ready:** Process that was initially in the ready state but if the ready queue becomes full, some processes are moved out of main memory and placed onto external storage by scheduler is said to be in *suspend ready state*. The process will transition back to ready state whenever the process is again brought onto the main memory.
7. **Suspend blocked or suspend wait:** Similar to suspend ready but uses the process which was performing I/O operation and lack of main memory caused them to move to secondary memory. When work is finished, it may go to suspend ready.

What is Thread?

A thread is a *flow of execution* through the process *code*, with its own program *counter* that keeps track of which instruction to execute next, system *registers* which hold its current working variables, and a *stack* which contains the execution history (see Fig. 3.7*a*). A process is defined as the *collection of threads*.

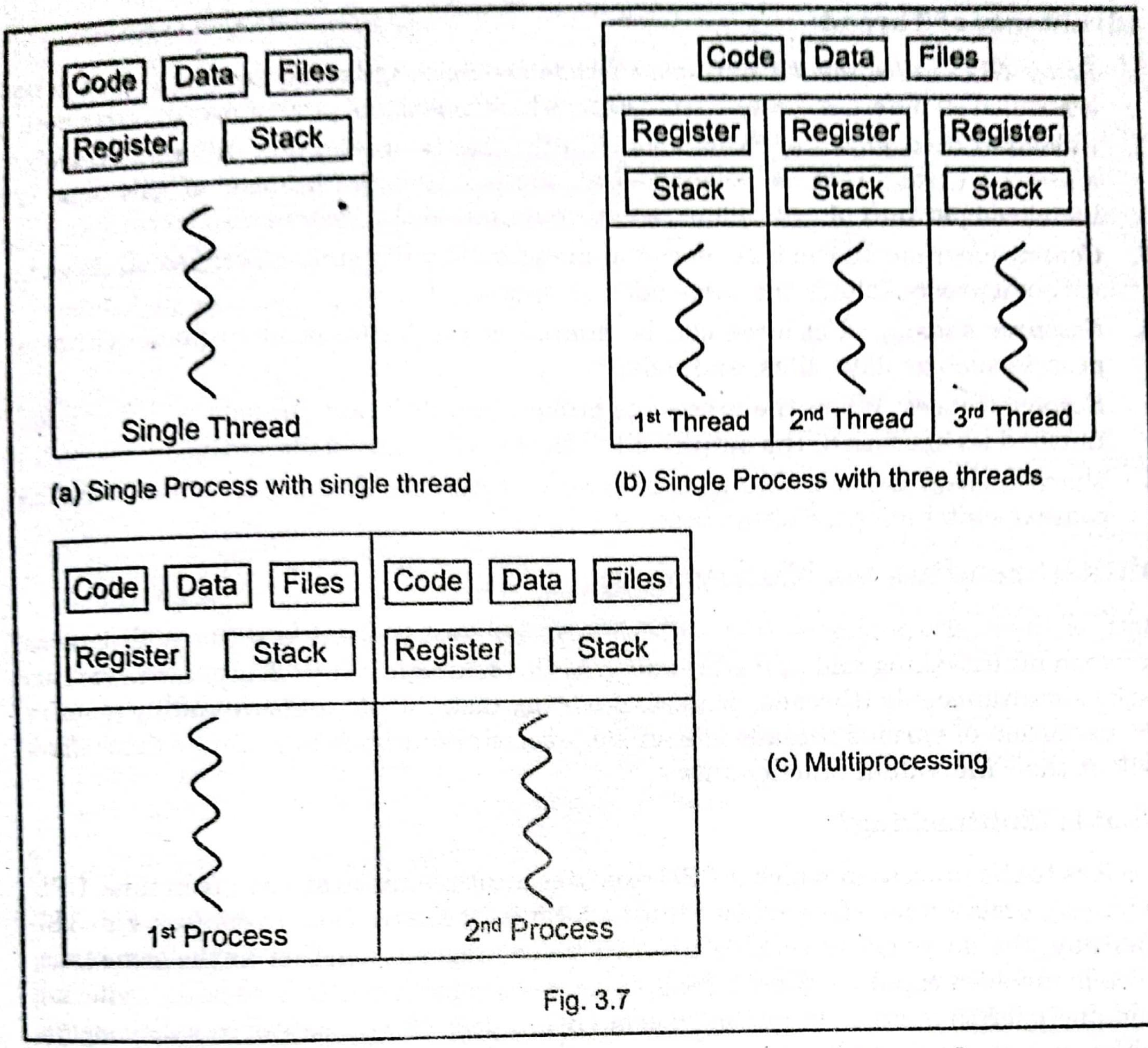
A thread shares with its peer threads few information like *code* segment, *data* segment and open *files*. When one thread alters a code segment memory item, all other threads see that.

A thread is also called a *lightweight process*. Threads provide a way to improve application performance through parallelism because threads are executed in a parallel manner. Threads represent a software approach to improving performance/speed of operating system by reducing the overhead thread. Threads provide a suitable foundation for running applications in parallel on shared-memory multiprocessors.

Each thread belongs to exactly one process and no thread can exist outside a process. Each thread represents a separate flow of control. Threads have been successfully used in implementing network servers and web server. They also provide a suitable foundation for parallel execution of applications on shared memory multiprocessors. The Fig. 3.7 shows the working of a single-threaded and a multithreaded process.

From the diagram, we can see that in multithreading (Fig. 3.7*b*), multiple threads share the same code, data, and files but run on a different register and stack. Multiprocessing (Fig. 3.7*c*) multiplies a single processor—replicating the code, data, and files, which incurs more overhead.

CamScanner



(a) Single Process with single thread

(b) Single Process with three threads

(c) Multiprocessing

Fig. 3.7

Table 3.2. Difference between Process and Thread

Process	Thread
Process is heavy weight or resource intensive.	Thread is light weight, taking lesser resources than a process.
Process switching needs interaction with operating system.	Thread switching does not need to interact with operating system.
In multiple processing environments, each process executes the same code but has its own memory and file resources.	All threads can share same set of open files, child processes.
If one process is blocked, then no other process can execute until the first process is unblocked.	While one thread is blocked and waiting, a second thread in the same task can run.
Multiple processes without using threads use more resources.	Multiple threaded processes use fewer resources.
In multiple processes each process operates independently of the others.	One thread can read, write or change another thread's data.

CamScanner

Advantages of Thread

1. *Better CPU utilization:* If multiple threads are existing in one process, they can be scheduled on different cores or processors, which makes the process execution quicker.
2. *Increased throughput of the system:* If a process is broken into different threads, and each thread task is considered as one job, and the number of jobs done is increased per unit of time. Thus, the throughput of the system also increases.
3. *Communication:* In the thread, communication is quite simple because all threads within a process share the same address space.
4. *Resource sharing:* Resource can be shared easily between all threads within a process such as data, files, and code.
5. *Responsiveness:* When the process is broken into different threads and if any file finished its execution, the output of the thread is returned instantly.
6. *Quick Context Switch:* The time of context switching in threads is less than the context switching time of the process.

Multithreading vs. Multitasking

Both of these are processes that a CPU performs, but there is a primary difference between multitasking and multithreading. Multitasking lets the CPU perform various tasks simultaneously (threads, process, program, task), while multithreading helps in the execution of various threads in a single process simultaneously. Let us first take a look at their individual functionalities.

What is Multitasking?

It refers to the process in which a CPU executes multiple tasks at any given time. CPU switching occurs very often when multitasking between various tasks (see Fig. 3.8). This way, the users get to collaborate with every program together at the same time. Since it involves rapid CPU switching, it requires some time. It is because switching from one user to another might need some resources. The processes in multitasking, unlike multithreading, share separate resources and memories.

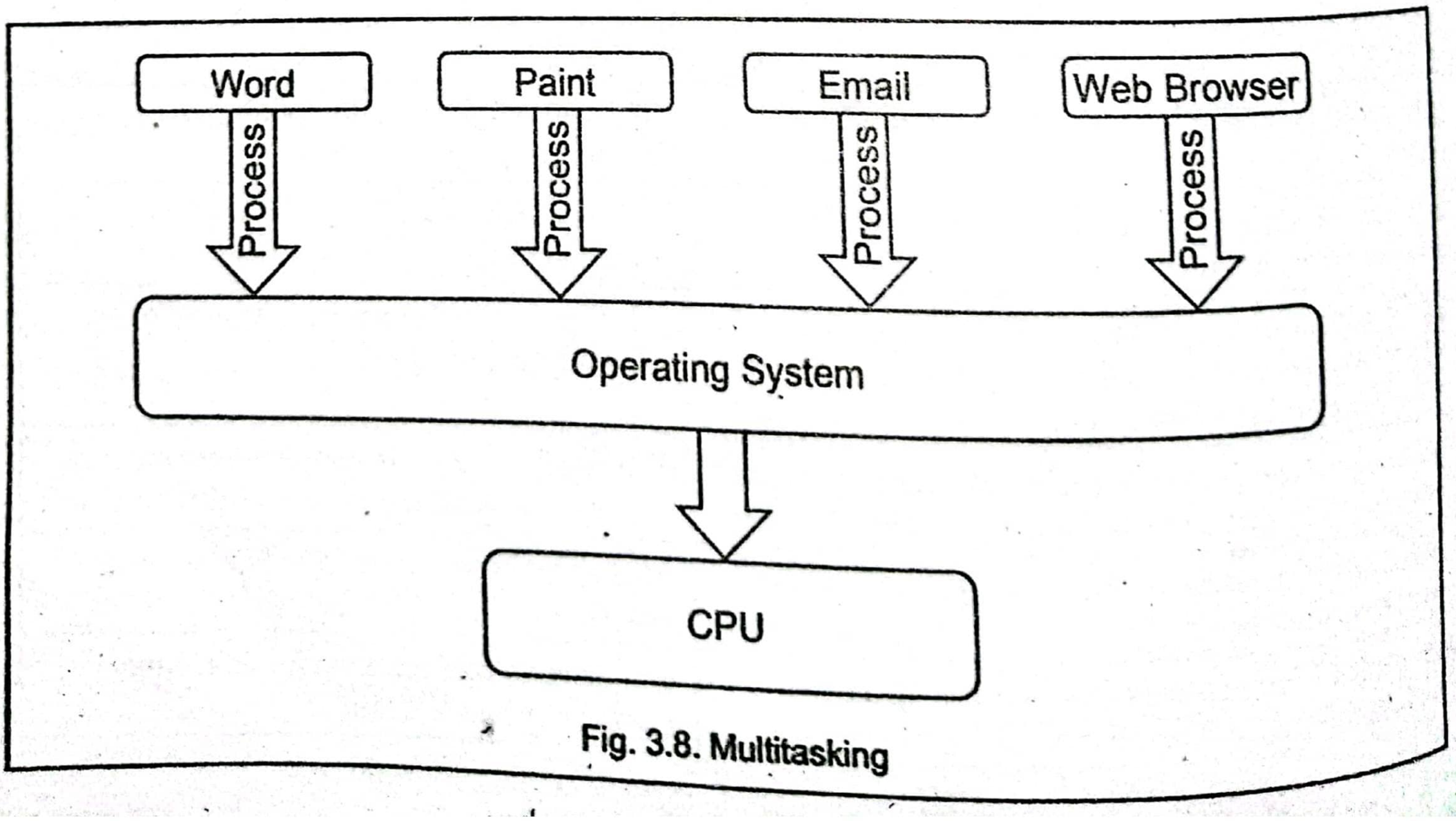


Fig. 3.8. Multitasking

CamScanner

Advantages of Multitasking

1. The main concept and advantage of multitasking is time-sharing. There is no CPU waiting time, and each task is given an appropriate amount of time.
2. Any program with a long waiting time due to an I/O interrupt is transferred to virtual memory. The program returns to RAM after the I/O operation is completed.
3. When using multitasking, a user can run a single program or multiple programs without any problem.
4. Multiple programs, such as MS Word, MS Excel, Photoshop, a browser, games, and a calculator, can operate concurrently.

Disadvantages of Multitasking

1. If the computer's processor is slow, it will process programs slowly and managing many programs will take longer. Because they require more processing power, some heavy programs cannot operate well on a slow processor.
2. When a computer user starts many programs simultaneously, the computer slows down. Because numerous applications are put into the main memory, the CPU is unable to provide an appropriate time for each program and the reaction time for performing the operation increases. It is a common problem with PCs that do not have enough RAM.
3. Multitasking keeps the processor active all the time and increases CPU temperature. You need to have efficient CPU's cooling system. It typically occurs when you play graphically demanding PC games.

What is Multithreading?

It is a system that creates many threads out of a single process to increase the overall power and working capacity of a computer (see Fig. 3.9). In the process of multithreading, we use a CPU for executing many threads out of a single process at any given time. Unlike multitasking, multithreading provides the same memory and resources to the processes for execution.

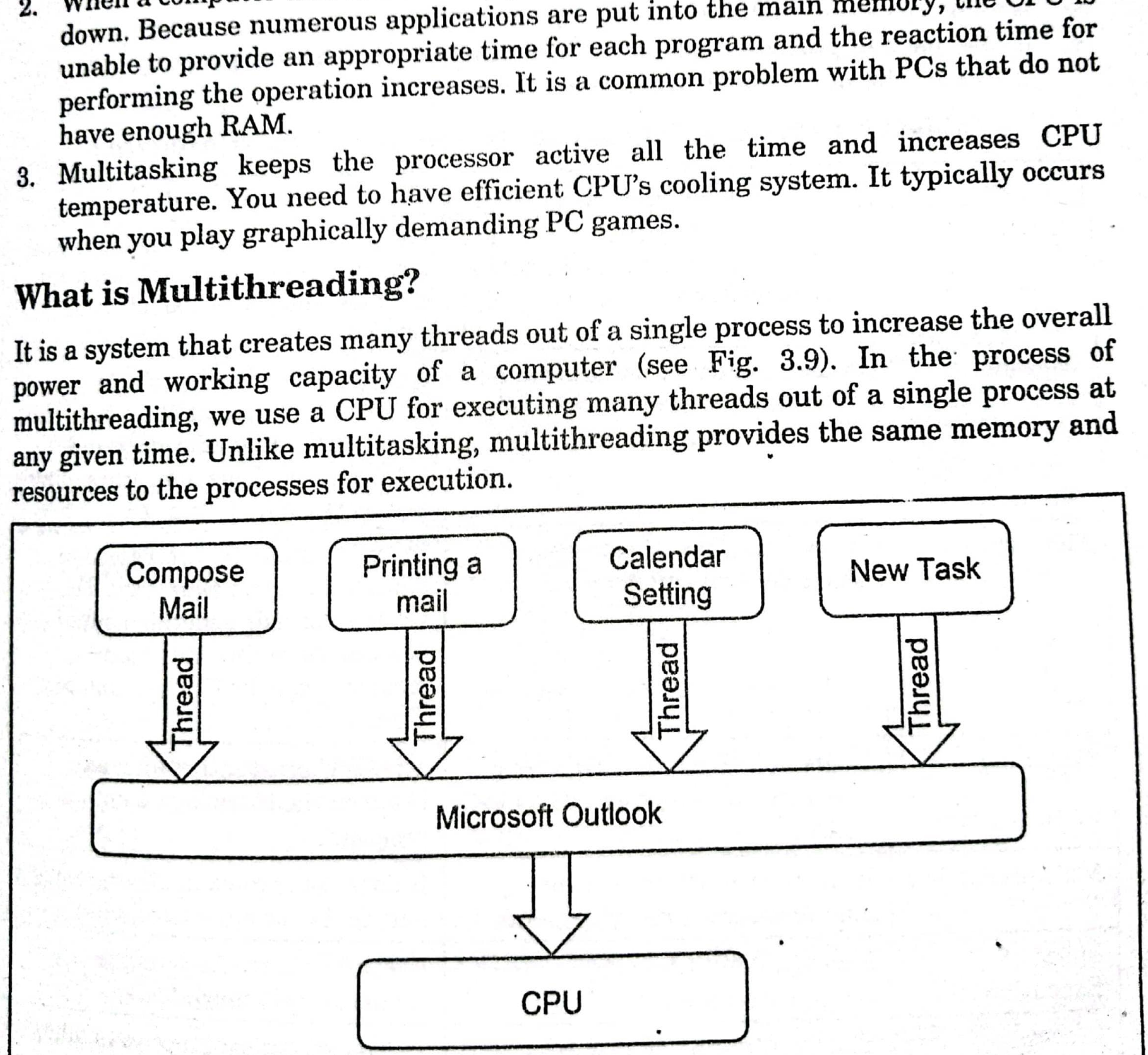


Fig. 3.9. Multithreading

CamScanner

Advantages of Multithreading

1. It is economical and takes less time to create threads.
2. It allows resource sharing among threads like data, memory, files, etc. Thus, multiple threads can share the same address space.
3. It increases the responsiveness towards the user as it allows the program to continue running irrespective of the situation.
4. It increases parallelism on multiple CPU machines and enhances performance of the multi-processor machines.
5. It uses CPU resources more efficiently

Disadvantages of Multithreading

1. It requires more accurate synchronization.
2. If many threads try to access the same data, then there is a chance that the situation of thread starvation may arise.
3. It adds overhead to context switching.
4. If the parent process requires several threads to function properly, the child processes should be multithreaded as well.

Table 3.3. Difference Between Multitasking and Multithreading

	Multitasking	Multithreading
Basics	The process of multitasking lets a CPU execute various tasks at the very same time.	The process of multithreading lets a CPU generate multiple threads out of a task and process all of them simultaneously.
Working	A user can easily perform various tasks simultaneously with their CPU using multitasking.	A CPU gets to divide a single program into various threads so that it can work more efficiently and conveniently. Thus, multithreading increases computer power.
Memory	In multitasking, the processes share separate memory.	In multithreading, processes are allocated the same memory. The various threads generated out of a process share that very same memory that the CPU allocated to them.
Switching	There is the constant switching between various programs by the CPU.	The CPU constantly switches between the threads and not programs.
Multiprocessing	It involves multiprocessing among the various components.	It does not involve multiprocessing among its various components.
Speed of Execution	Executing multitasking is comparatively slower.	Executing multithreading is comparatively much faster.
Process Termination	The termination of a process takes up comparatively more time in multitasking.	The termination of a process takes up comparatively less time in multithreading.

CamScanner

What is Windows?

Windows is an operating system developed by Microsoft Corporation and it dominates the PC environment. The feature that makes Windows so easy to use is a *graphical user interface* (GUI), in which users click the desired app or program to run it rather than with keyed-in commands.

Windows 11 contains some new features and clever ways to help you work faster and get more out of your computer. Some of the features will be familiar to anyone that has used a Windows computer, but there are also new features and ways of working too. Here is a brief description of parts of Windows 11 operating system.

Start menu

Windows 11 introduces new interfaces in almost every area of the desktop experience, and that includes the Start menu. The Taskbar icons are centered and the *Start button* is to the left of the other app icons (see Fig. 3.10). Clicking the Start button opens the Start Menu. In Windows 11, the Start menu contains the following four parts:

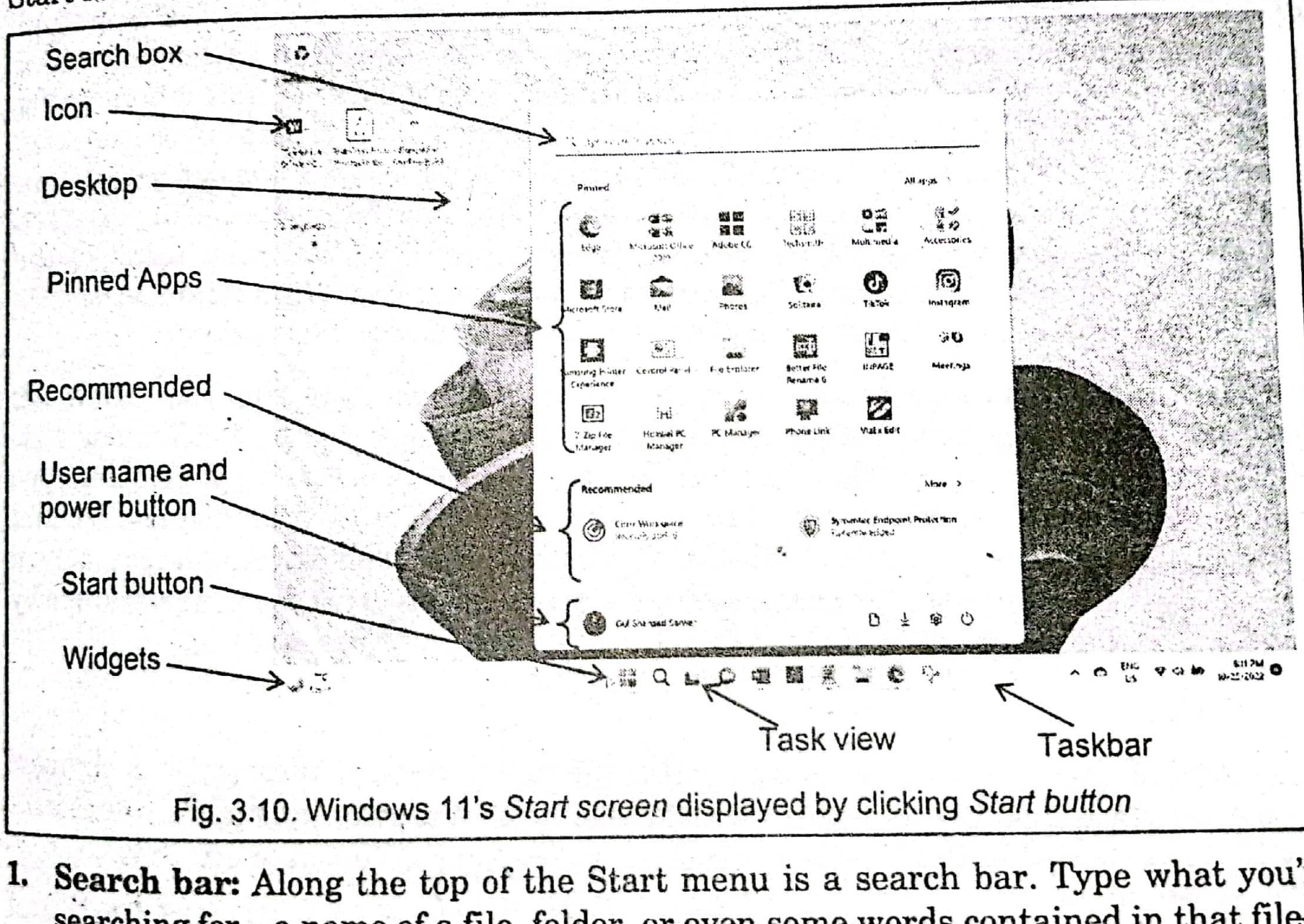


Fig. 3.10. Windows 11's *Start screen* displayed by clicking *Start button*

1. **Search bar:** Along the top of the Start menu is a search bar. Type what you're searching for—a name of a file, folder, or even some words contained in that file—into the Search bar, and Windows will try to find it, whether it's on your PC or the internet.
2. **Pinned:** Windows displays a selective list of *pinned* apps in this section. To see *all* your apps, click the *All Apps* icon in the section's upper-right corner. You can pin your own favorite apps here.
3. **Recommended:** Below your pinned apps is a new *Recommended* area that acts as a *recent menu* for things like documents and installed apps. Whenever you install

CamScanner

a new app or open an Office document, it will appear directly in this Recommended area for quick access. That means any documents, whether they be photos, Word documents, Excel spreadsheets, or even random files in some cases, can show up there.

4. **User Account name and Power:** The Start menu's bottom section lists your user account name. Opposite the name, you have a power button. Click it to put your computer to sleep, restart it, or shut it down.

Start menu has a grid of icons that you can pin, unpin, and reorganize. To remove an unwanted or unused icons from the Start menu, just right-click it and choose *Unpin from Start* from the pop-up menu. The icon disappears without fuss, freeing up some prime space. (This doesn't uninstall the app or program; any unpinned items can still be found in the Start menu's *All Apps* area.)

The Desktop

After you turn on your PC and type in your username and password, the Windows desktop fills the screen, ready for work. The desktop lets you run several apps and programs simultaneously, each living within its own little *window*. That separation lets you spread several programs across the screen, sharing bits of information among them. Windows 11 introduces new interfaces in almost every area of the desktop experience. The Windows 11 desktop may contain a *background, one active window, one or more inactive windows and icons* (see Fig. 3.10). The Windows desktop and the objects on the desktop resembles a work area in an office. You may think of the Windows desktop as an electronic version of the top of your desk. You can move objects around on the desktop, look at them and then put them aside, and so on.

Microsoft also made several key changes and improvements to the Task View UI, which is where many go to see an overview of all their running apps. On Windows 11, Microsoft has removed the old Timeline feature, instead prioritizing your open apps and Virtual Desktops, which now appear along the bottom of your display. Virtual Desktops are much more customizable now, with abilities such as renaming and even setting custom wallpapers for each desktop. You can also reorganize your desktops by clicking and dragging.

The Taskbar

A big area of change on Windows 11 is with the new *Taskbar*. Microsoft has changed the layout of the Taskbar so system buttons and pinned or running apps are centered. This is a big change to the Taskbar, which has always been left-aligned.

All of the system icons (those being Start, Search, Task View, Teams Chat, and Widgets) have cute little animations that play when you click on them. And your pinned or running apps also have new animations that play when you click on them.

The System Tray has been re-done on Windows 11 in an attempt to simplify it as much as possible (see Fig. 3.11). Microsoft has split up the Action Center into two separate flyouts: one for notifications and the other for quick settings. Clicking on the date and time button will open up your calendar view and notifications, and clicking on either Wi-Fi, Volume, or Battery will open the new Quick Settings panel.

CamScanner

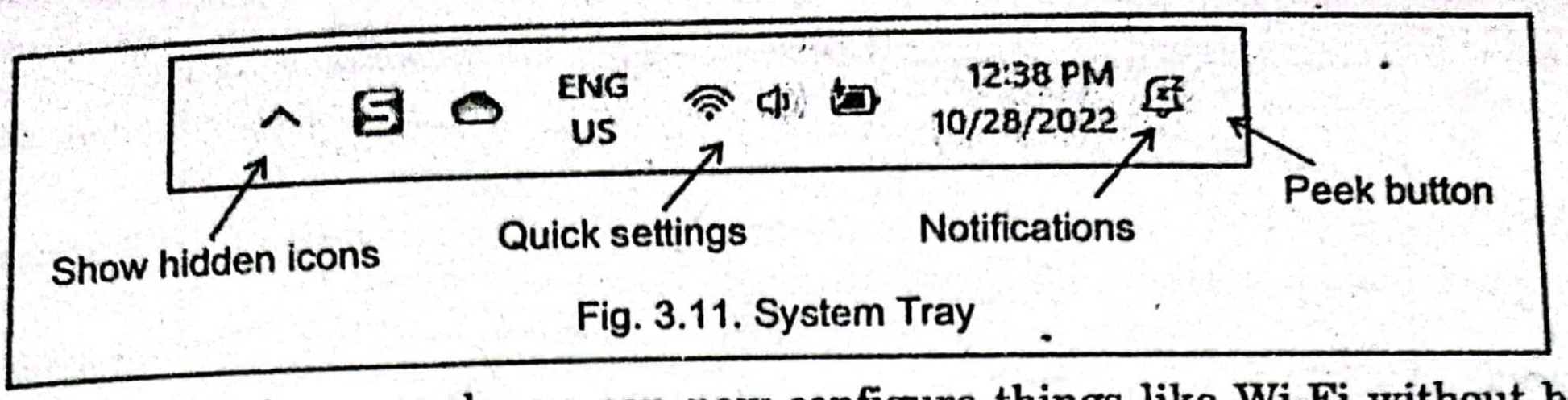


Fig. 3.11. System Tray

Using *Quick Settings* panel, you can now configure things like Wi-Fi without being sent into the Settings app first. Some of the toggles have additional menus that let you configure them directly within the Quick Settings panel, which keeps you in your flow and doesn't get in the way of your currently open app. That said, not all of the toggles can be configured directly from the Quick Settings panel, such as Bluetooth, which still takes you to the main *Settings* app. Having all these options in one panel makes the user experience feel much less cluttered, which is the whole point of Windows 11.

To open an application from the taskbar, click the application's icon. To search your computer for files and apps, or the Web for additional information, click search button and type your query. To view all open applications in thumbnail form, click the Task View button.

To minimize all open applications, click the slim Peek button at the far right of the taskbar.

Widgets

A new feature that Microsoft is trying to push on Windows 11 is *Widgets*, which exists as a hidden panel that flies out above your desktop from the left side of the screen. Widgets are a personalized feed powered by artificial intelligence and Microsoft Edge. There is a dedicated button for it on the Taskbar, or you can access it by swiping in from the left edge of your display. The panel consists of a widgets area at the top that has a handful of customizable widgets to choose from.

As of February 2022, Microsoft has updated this feature with a new entry point on the Taskbar. It is now accessible via a *weather* button placed in the far left of the Taskbar where the Start button used to be. This button will present you with the current weather condition, and clicking it will open up the full Widgets panel for access to all of the Widgets that are available to you.

File Explorer

The File Explorer plays a key part of your Windows 11 experience. The main section displays the contents of your folders while the others help you navigate your way around the contents of your PC.

Browse files and folders: The main function of File Explorer is to browse, open and manage the files stored on your hard drive and OneDrive (see Fig. 3.12). Microsoft recently added a new OneDrive status icon into the File Explorer to get the details of the OneDrive in the file explorer itself. You can navigate to different folders in the file explorer and get the details of the OneDrive accounts that those folders are synced, etc., using the new OneDrive icon.

CamScanner

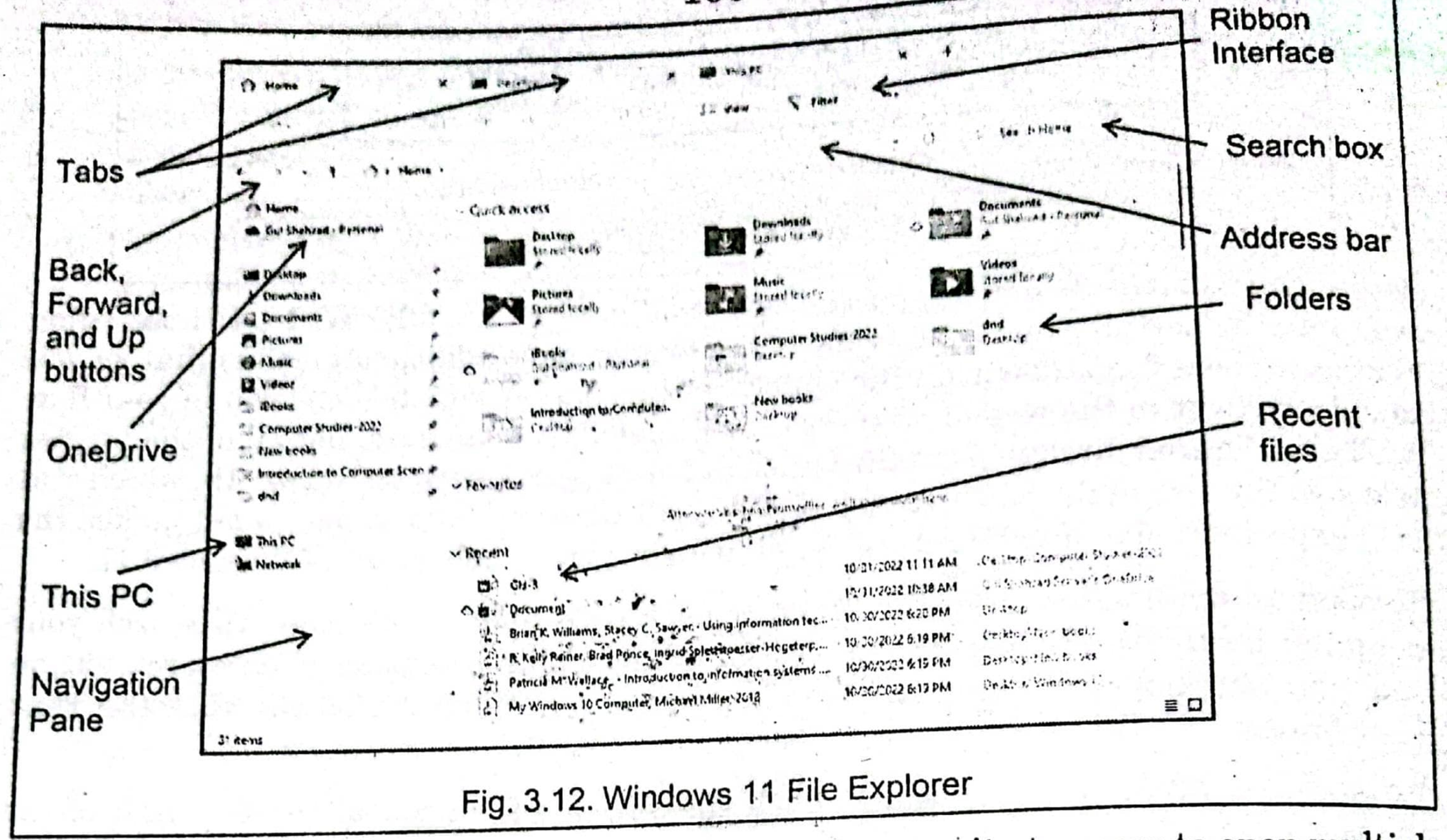


Fig. 3.12. Windows 11 File Explorer

Windows 11 File Explorer Tabs Feature gives an opportunity to users to open multiple tabs in the same Windows file explorer window. This helps users from opening multiple file explorer windows to perform operations such copy and paste between two folders. Many features and use cases are available for File Explorer Tabs, similar to Microsoft Edge or Chrome browser multiple tab options.

Use of Windows 11 New File Explorer: Let's check what the uses of Windows file explorer are. You can see the default functionalities of the explorer is available with Windows 11. File explorer's use cases are given below, and these basic functionalities are not changed when you compare them with Windows 10.

1. *Interact* with stored files in Windows.
2. Organize your *personal* files.
3. Create *Folders* to organize files.
4. Interact with files stored in *cloud-like* OneDrive.
5. A search *bar* option is available.
6. *Path bar* is available for navigation through different files and folders.

File Management: Your hard drive is essentially made up of files which reside within folders. Folders are a way of keeping everything organized on your hard drive. Within a folder you can have any number of files, as well as more folders—known as sub-folders. When a new program is installed, you will find that it will be placed into its own dedicated folder within your Program Files folder (also known as a directory). When arranging files yourself, you can also create your own folders and name them.

You can move and copy files around your hard drive to arrange them into your folders. Moving a file is known as cutting; it involves taking a file from one location and placing it into another. Copying a file is similar, but the outcome is that a duplicate version of

CamScanner

the file is created in another location while the original file remains where it was. By right-clicking a file, you can also use the 'Send To' tool to quickly move a file to another location.

Windows 11 File Explorer Home Page: Let's check *Windows 11 File Explorer Home Page* option available with the latest version. The Quick Access folder is replaced automatically with the home folder on Windows 11 File Explorer.

New Toolbar: You can check the new toolbar options with Windows 11 file explorer. You don't see ribbon options in the new file explorer. You don't have options like *File, Computer*, and *View* ribbon options in Windows 11 explorer. The new toolbar comes with *very simplified* icons. These icons are there for the following operations: (a) Cut, (b) Copy, (c) Rename, (d) Share, (e) Delete.

These functions can also be accessed by right-clicking on a file or folder and selecting from the menu that appears.

Change display options: File Explorer has a variety of ways in which it can display your files and folders. It can also arrange them into orders that go beyond alphabetical sorting.

Starting with the visual aspect, your files can be shown in a list, as tiled icons or thumbnail images. These views can be chosen by clicking the *View* icon at the top of window, followed by clicking an entry from the selection underneath (You can have an option to change the view of file explorer using the icons on the *right-side bottom corner* too.). There is also a *Details* view that shows your files in a table along with detailed information (such as file size, date modified etc.). When in Detail view, you can decide what details are visible by right-clicking on the column titles at the top of the display. You can also left-click on these column titles to reorder the currently viewed files based on the clicked topic. If you are not in Details view, you can still reorder the files by clicking the *Sort* icon at the top and selecting an option from the menu. Windows 11 also has a handy *Group by* option (in *Sort* menu) that can bunch together files based on a certain measurement; for example, you can batch photos together based on what day they were taken.

You can also use the right-hand side as a method of previewing a file without having to open it. This can show images and documents that you have created.

Sort and Group Operations: You can see the new option to *sort, and group operations* are available near to delete button. The following options are available in the new explorer: (a) Name, (b) Type, (c) Total Size, (d) More, (e) Ascending, (f) Descending, (g) Group by.

Layout and View Options: The new *layoit and view options* are available in the Windows 11 new file explorer option. You can change the view of file explorer with Extra large icons, Large icons, Medium icons, Small icons, List, Details, Tiles, Content, Compact view, and Show options.

Search for files: Windows 11 file explorer includes a very useful search function that is just the thing for tracking down files. You will find it towards the top-right area of the File Explorer window. Simply start typing in the box and any matching results will

CamScanner

appear in the main file window. The results will be matches found in the currently selected folder.

Context menu: There is a new, simplified context menu design (see Fig. 3.13) which has all the older legacy context menu options into a secondary menu for those who may still need access to it. This can become quite annoying if you are using apps that put their options into that secondary menu, such as 7-Zip. Developers can update their apps to put their options into the new modern context menu. The most common tasks such as copy, paste, cut, and delete, are represented as icons that flow horizontally along the top of the menu. Then, all the other options are presented as a vertical list.

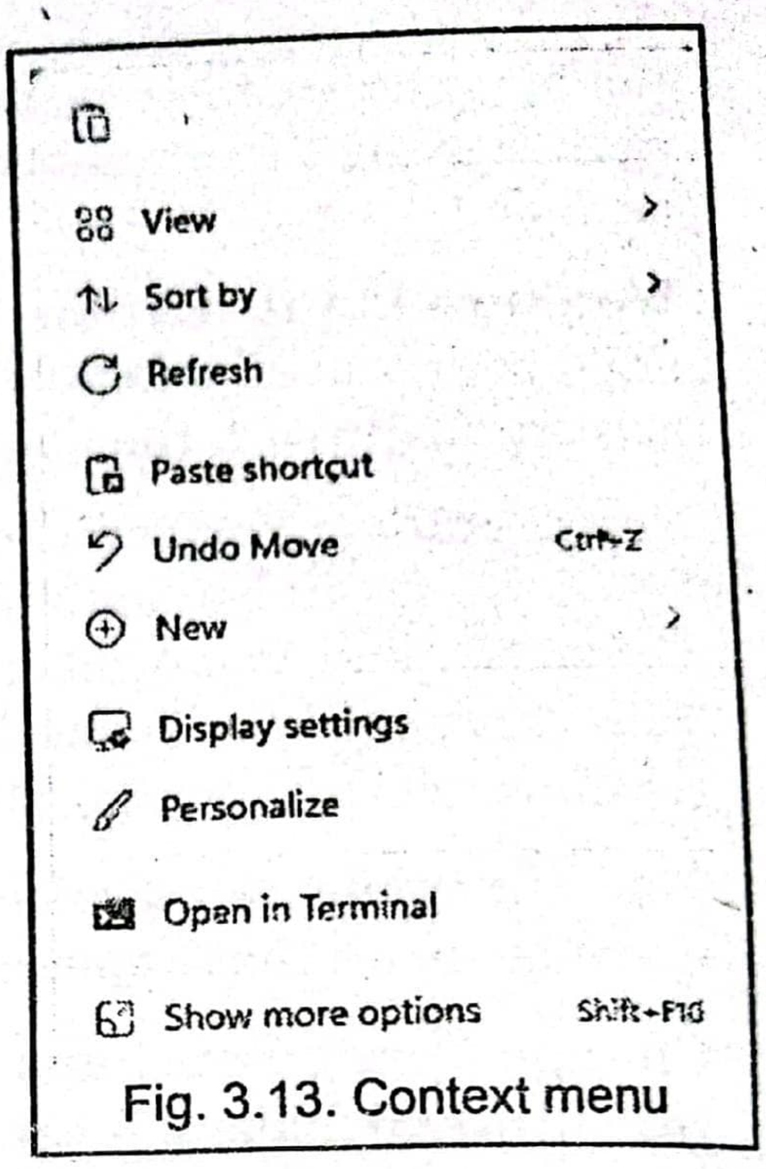


Fig. 3.13. Context menu

Settings

The brand-new Settings app in Windows 11 (click Start → Settings) enables you to customize the way Windows looks and works. It means that you can personalize it and make it work in a way that best suits the way your work on your computer or the leisure activities it is used for (see Fig. 3.14). Simple examples of this are the options to change the desktop background and colors, and the lock screen image. You do not have to accept the default image and color scheme and you can select your own.

Settings is where you set up devices like printers, scanners, a Bluetooth mouse or keyboard. It provides access to user accounts where you can manager your own, add another person, change permissions and passwords, and more. A privacy and security section contains controls for setting the amount of information you share, and windows update section enables you to get Windows updates and recover from problems.

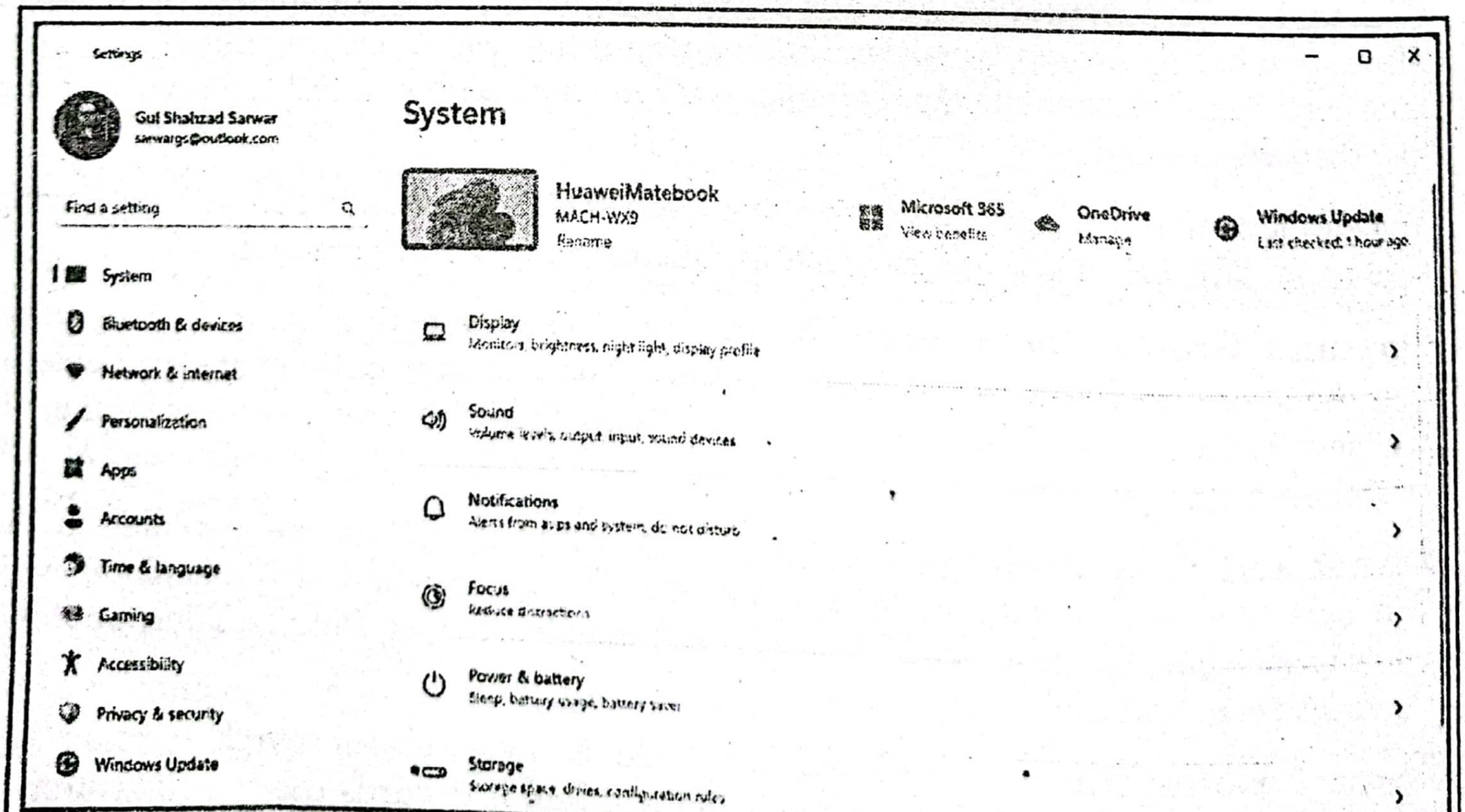


Fig. 3.14. Settings are used to customize the hardware and software

CamScanner

DirectX Diagnostic Tool (Dxdiag.exe)

`DxDiag` ("DirectX Diagnostic Tool") is a diagnostics tool used to test DirectX functionality and troubleshoot video- or sound-related hardware problems (see Fig. 3.15). DirectX Diagnostic can save text files with the scan results. These files are often posted in tech forums or attached to support emails in order to give support personnel a better idea of the PC the requester is using in case the error is due to a hardware failure or incompatibility.

`DxDiag` is located in `C:\Windows\System32`. Starting from Windows Vista, `DxDiag` only shows information; it is no longer possible to test the hardware and the various DirectX components.

This tool is used to collect info about devices to help troubleshoot problems with DirectX sound and video. A support person might ask for this info, or you might post it in a forum when you ask for help.

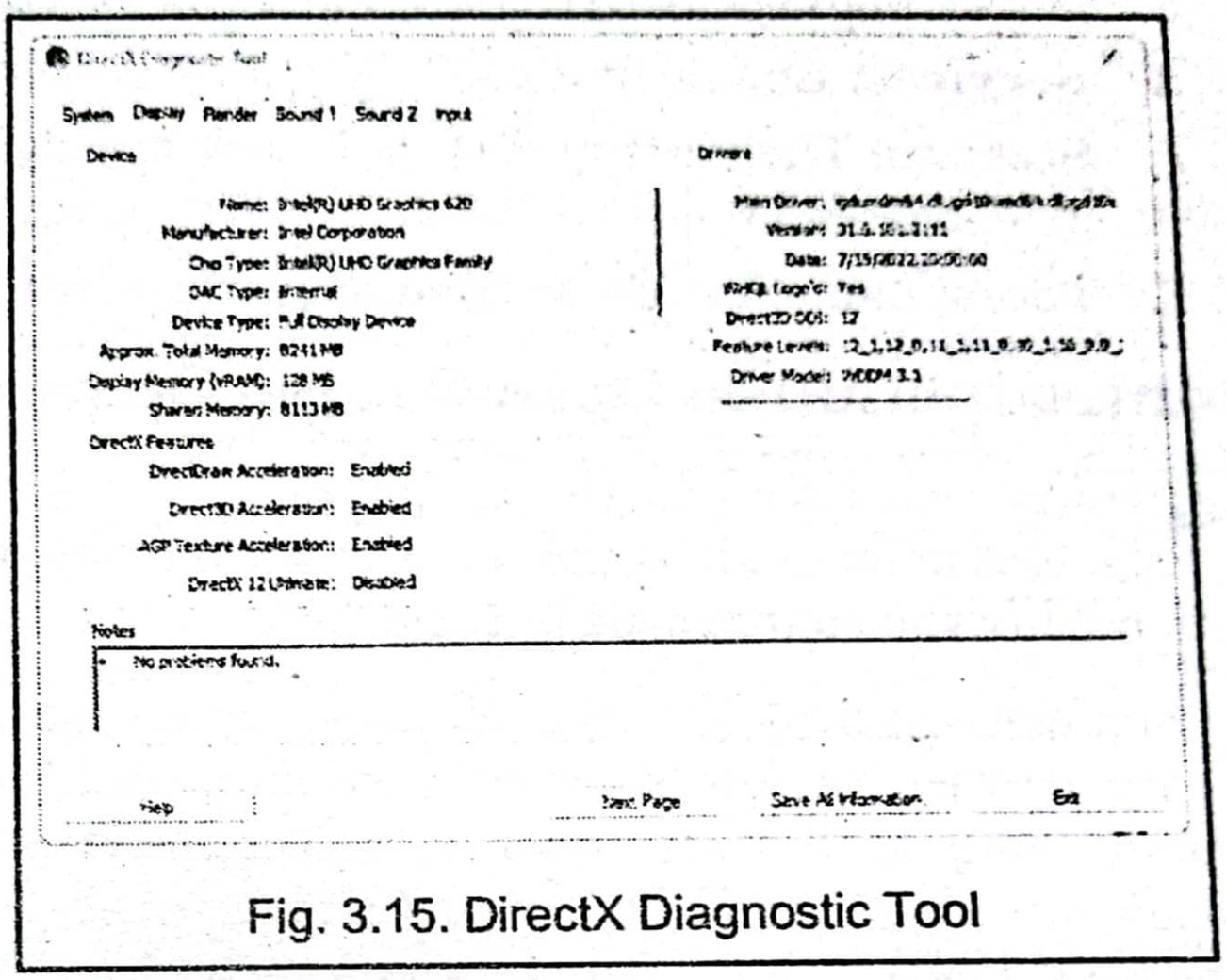


Fig. 3.15. DirectX Diagnostic Tool

To run the tool, click Start button and enter `dxdiag` into the search box. Select `dxdiag` from the results. In the tool, select Save All Information for an easily shared format

System Configuration Utility (`msconfig`)

`msconfig`, also known as the System Configuration Utility, is a system utility designed to troubleshoot the Microsoft Windows startup process. It can disable or re-enable software, device drivers or Windows services that run at startup, and it can change boot parameters.

`msconfig` is easy to use and helps you troubleshoot a lot of Windows 11 boot problems. You can easily modify how the system boots (boot files) and the system services on either a temporary or a permanent basis. This makes `msconfig` a very useful troubleshooting utility. It can be used to speed up your Boot times and to streamline what loads where and when as your system starts up.

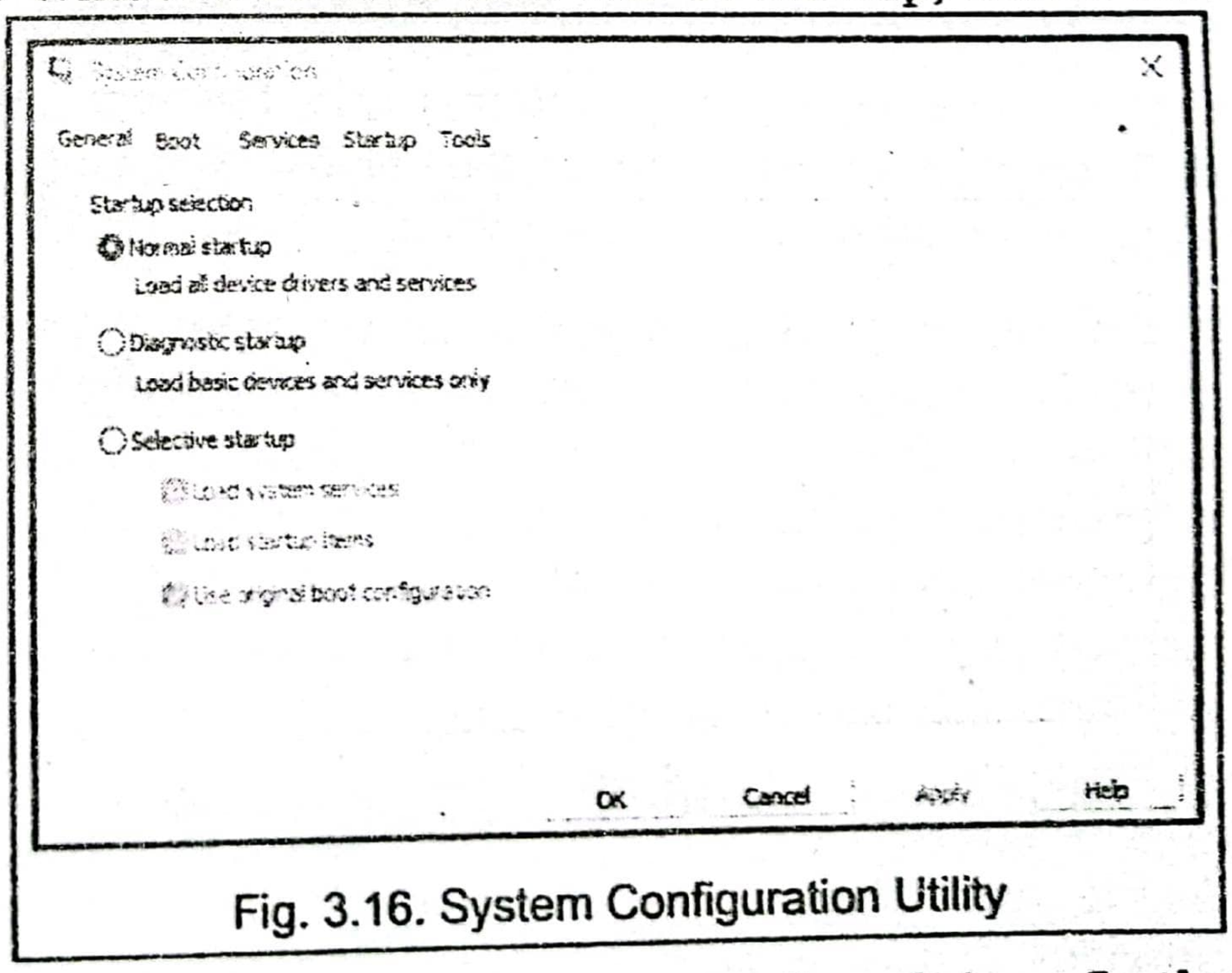


Fig. 3.16. System Configuration Utility

How to Open `msconfig`: On the Windows 11 taskbar, click the Search icon. In the text field at the top of the search window, type `msconfig` and press Enter. This

CamScanner

launches Microsoft's System Configuration Utility, as shown in the Fig. 3.16. Alternately, open Run Prompt (Win+R), and type `msconfig` and press the Enter key. It will launch the System Configuration Utility. It will display five tabs:

1. **General:** Allows you to boot Windows in diagnostic or selective mode when necessary.
2. **Boot:** Manage everything related to Windows boot, including Safe mode.
3. **Services:** Enable or disable Windows and other services.
4. **Startup:** The startup section is now managed via Task Manager or Settings app in Windows 11.
5. **Tools:** Launch popular System services from here.

Introduction to Open-Source Operating System

Open source refers to the computer software or applications where the owners or copyright holders allow the users or third party to see, use, share and provide the right to modify the source code of the product.

On a closed-source OS like Windows, that code can only be altered by Microsoft, and only viewed by few selected customers like big companies. In contrast, the code underlying open-source operating systems is not only freely available for anyone to view but also to modify, use and share, under the terms of open-source licenses such as MIT, GNU Public License, and Apache 2.0.

Closed-source Operating Systems are built with numerous codes and complex programming and that is called *source code*. This source code is kept secret by the respective companies (owners) and inaccessible to third parties. By doing so, they ensure the safety and secure the Operating System and computer from any threats. The generally known Operating Systems like Microsoft's Windows, Apple's macOS and iOS, are closed-source Operating system.

How Open-Source Operating System Works?

Open-Source Operating System works the same as the closed-source ones; the only difference is that the source code or the whole application is modifiable by the user. There is no difference in performance, but there can be a difference in functioning.

For example, in a proprietary (closed-source) operating system, the information is packed and stored. The same happens in the open source. But since the source code is visible to a user, the user can understand the software and alter the way information is processed. While the former is secure and hassle-free, the latter needs some technical knowledge, but you can customize and increase performance. There is no defined way or framework for the open-source operating system working; it can be customized based on the user needs.

Types of Open-Source Operating System

At the heart of most open-source operating systems today is the Linux kernel. Linux provides the core of these operating systems, the kernel that interacts with a computer's hardware, and was developed by Linus Torvalds in 1991.

CamScanner

This kernel provides the core functionality any OS needs, controlling how data is processed and parceled out into memory, how the system handles files, how it interacts with devices plugged into the computer and other fundamental tasks.

Operating system makers then build tools that plug into this Linux kernel to create an operating system. These tools range from the windowing systems that powers graphical desktops to the systems managing services running in the background. The development of the Linux kernel continues, with version 5.0 of the Linux kernel recently released.

While Linux is at the heart of operating systems, there is a huge variety in the look and feel of Linux-based operating systems. Some focus on simplicity and usability, such as Ubuntu, Linux Mint and Elementary OS; others on security, such as Tails; others on providing powerful tools for technical experts, such as Debian and Gentoo. And while many have similarities with the Windows desktop, others borrow design cues from macOS, while others offer no graphical interface at all by default, just a bare command line.

There are also some open-source operating systems that are *not based on Linux*. One notable example is FreeBSD, a free OS that is not built around the Linux kernel.

Advantages of Open-Source Operating Systems

1. *Cost-efficient:* Most of the open-source OS are free and some of them are available at a very cheap rate than the commercial proprietary OS.
2. *Reliable and efficient:* Most of them are monitored by thousands of eyes since the source code is public. So, if there is any vulnerability or bugs, they are fixed by the best developers around the world.
3. *Flexibility:* You can customize it as per your requirements and there is creative freedom.

Disadvantages of Open-Source Operating Systems

1. *Security risk:* Though the bugs are identified, there is a risk of attacks as the source code is available to the attackers.
2. *Complicated:* It is not user-friendly as the closed-source ones. You need to have the minimum technical knowledge to use open-source OS.
3. *No support:* If you run across an issue, there is no customer support available to assist you.

Android OS—An example of the most popular open-source OS

The most popular open-source operating system is Android, which is based on a modified version of the Linux kernel. It primarily runs on smartphones and tablets. The Android platform includes an operating system based upon the Linux kernel, a GUI, a web browser and end-user applications that can be downloaded. Android was released under the Apache v2 open-source license; this allows for many variations of the OS to be developed for other devices, such as gaming consoles and digital cameras. Android is based on open-source software, but most Android devices come preinstalled with a suite of closed-source Google-branded apps, such as Google Maps, YouTube, Google Chrome and Gmail.

CamScanner

History and Development

Android began its life as a Palo Alto-based startup company called Android Inc., in 2003. Originally, the company set out to develop an operating system for digital cameras, but it abandoned those efforts in lieu of reaching a broader market.

Google acquired Android Inc. and its key employees in 2005. Google marketed the early mobile platform to handset manufacturers and mobile carriers with its major benefits as flexibility and upgradability. Google released the public beta version of Android 1.0 for developers in November 2007.

Android OS Features

The default UI of Android relies on direct manipulation inputs such as tapping, swiping and pinching to initiate actions. The device provides haptic feedback to the user via alerts such as vibrations to respond to actions. If a user presses a navigation button, for example, the device vibrates.

When a user boots a device, Android OS displays the home screen, which is the primary navigation hub for Android devices and is comprised of widgets and app icons. Widgets are informational displays that automatically update content such as weather or news. The home screen display can differ based on the device manufacturer that is running the OS. Users can also choose different themes for the home screen via third-party apps on Google Play Store.

A status bar at the top of the home screen displays information about the device and its connectivity, such as the Wi-Fi network that the device is connected to or signal strength. Users can pull down the status bar with a swipe of a finger to view a notification screen.

Android OS also includes features to save battery usage. The OS suspends applications that aren't in use to conserve battery power and CPU usage. Android includes memory management features that automatically close inactive processes stored in its memory.

Android OS Versions

Google makes incremental changes to the OS with each release. This often includes security patches and performance improvements. Some latest versions of Android are:

- *Android 10 (Android Q):* Released Sept. 3, 2019. Abandoned the Back button in favor of a swipe-based approach to navigation. Introduced a dark theme and Focus Mode, which enables users to limit distractions from certain apps.
- *Android 11 (Red Velvet Cake):* Released Sept. 8, 2020. Added built-in screen recording. Created a single location to view and respond to conversations across multiple messaging apps. This version also updated the chat bubbles so users can pin conversations to the top of apps and screens.
- *Android 12 (Snow Cone):* Released Oct. 4, 2021. Added customization options for the user interface. The conversation widget let users store preferred contacts on their home screens. Added more privacy options, including sharing when apps access information such as camera, photos or microphone.
- *Android 12L:* Released March 7, 2022. The L stands for larger screens. This update

CamScanner

aimed to improve the user interface and optimize for the larger screen of a tablet, foldable or Chromebook. This update added a dual-panel notification center for tablets and foldables.

- *Android 13 (Tiramisu):* Released Aug. 15, 2022. Included more customizable options including color, theme, language and music. Security updates included control over information apps can access, notification permission required for all apps and clearing of personal information on clipboard. This update enables multitasking by sharing of messages, chats, links and photos across multiple Android devices—including phones, tablets and Chromebooks.

Hardware

Android uses ARM for its hardware platform; later versions of Android OS support x86 and x86-64 architectures. Starting in 2012, device manufacturers released Android smartphones and tablets with Intel processors.

Google releases documentation with hardware requirements that original equipment manufacturers (OEMs) must meet for a device to be "Google Approved," which means that it will ship with official Google apps. The open-source nature of Android, however, means that it can also run on lesser hardware, and vice versa.

Comparisons with other mobile OSes

Initially, Android's creators believed that the OS would compete with other mobile operating systems such as Symbian and Microsoft Windows Mobile.

Symbian was a closed OS with a microkernel and a UI that provided the graphical shell. Many mobile manufacturers used Symbian OS, including Nokia, Samsung and Motorola. Symbian was a popular OS worldwide, but it did not gain major popularity in North America. Symbian's design was not as simple as Android and iOS, however, and the OS was difficult to program. Symbian OS development was discontinued in 2014.

Windows Mobile originated from Windows CE, an embedded OS, and first appeared on a Pocket PC 2000. Microsoft marketed the mobile OS toward businesses. Competition from Android and iOS forced Microsoft to make changes; the company replaced Windows Mobile with Windows Phone in 2010, aimed at the consumer market. Microsoft phased out Windows Phone in favor of Windows 10 Mobile, but that OS was also discontinued; Microsoft declared its end of life for Jan. 14, 2020.

Android's main competitor is Apple iOS. Both iOS and Android OS offer comparative features. Apple iOS is a proprietary OS with a fixed interface, whereas Android is an open-source OS that offers more flexibility and customization.

CamScanner

Questions

3.1. What is an operating system? Why is it necessary for a computer system?
3.2. List out the various functions normally performed by an operating system.
3.3. "The operating system tends to isolate the hardware from the users". Discuss the statement.
3.4. Define the term user interface. Distinguish between a command-line interface and graphical user interface, stating one advantage of each.
3.5. In the Windows 11 environment, how is an item, such as an application program or a menu option, selected with a mouse? How is the item opened?
3.6. What are the principal functions performed by an operating system?
3.7. List four elements of the Windows application window.
3.8. Write brief notes on MS-DOS and UNIX.
3.9. What is batch processing feature of an OS?
3.10. Which of the following is an example of batch processing?
Example are: Research Segment, Billing System, and Transactions Process.
3.11. What is time sharing technique in an OS?
3.12. What is the other name for time sharing in an OS?
3.13. What is quantum?
3.14. What is an embedded operating system?
3.15. In the Windows environment, is the active window highlighted in the foreground or the background?
3.16. What is displayed in a Windows 11 taskbar?
3.17. Define file management system. How can you use File Explorer for file management?
3.18. What type of jobs can be performed using File Explorer?
3.19. What is open-source software?

CamScanner

Objectives and MCQs

Each of the following questions has four alternative answers, of which one or more may be correct. Pick up the correct alternative / alternatives.

3.1. Through a user interface, the user is able to
- (a) control the computer
- (b) request information from the computer
- (c) respond to messages displayed by the computer
- (d) all of the above

3.2. Which of the following are functions of the operating system?
- (a) file management
- (b) CPU management
- (c) task management
- (d) booting
- (e) all of these

3.3. Which of the following was the first major personal computer OS?
- (a) macOS
- (b) Windows
- (c) DOS
- (d) Unix
- (e) Linux

3.4. A shortcut menu displays when you _____ a(n) _____.
- (a) right-click, object
- (b) click, menu name on the menu bar
- (c) click, submenu
- (d) click, recessed button in the taskbar button area

3.5. To shut down Windows 11,
- (a) click the Start button, select Power button, and then click Shut Down
- (b) click File on the menu bar and then click Shut Down
- (c) right-click the taskbar, click Shut Down on the shortcut menu, and then click the OK button
- (d) press the F10 key and then click the OK button

3.6. The _____ contains the Desktop icon in Quick access.
- (a) Navigation pane
- (b) Folders pane
- (c) status bar
- (d) Standard Buttons toolbar

3.7. Which of the following refers to the execution of two or more programs by one user almost at the same time on the same computer with one central processor?
- (a) multitasking
- (b) multiprocessing

CamScanner

(c) time-sharing
(d) multiprogramming
(e) coprocessing

3.8. What is the common consumer computer interface used today?
(a) command-driven interface
(b) graphical user interface
(c) menu-driven interface
(d) electronic user interface
(e) biometric user interface

Objectives

3.1. A user interface is a combination of computer hardware and computer software.
3.2. System software enables the computer to perform essential operating tasks and makes it possible for application software to run.
3.3. You can maximize a window by clicking maximize button on title bar.
3.4. File Explorer is an application you can use to organize and work with the files and folders on the computer.
3.5. A folder that is contained within another folder is called a subfolder.

CamScanner

CHAPTER 4

ALGORITHM AND FLOWCHARTS

By: Sir Ahad

Algorithm

The word algorithm is derived from the well-known ninth century mathematician of Iran Abu Jafar Ibn Musa Al-Khwarizmi, who developed methods to solve problems using specific instructions step-by-step. Originally the term was *algorism* but later on, it changed to *algorithm*.

Consequently, the *algorithm* can be defined as a finite sequence of unambiguous instructions designed to solve a particular problem and these instructions ultimately terminate if followed. You may think that the latter part of this definition (requiring a finite sequence of instructions to terminate) is extra. However, the fact that a list of directions contains only a finite number of instructions does not necessarily mean that the described task will ever be completed if followed. For example, the single instruction that reads "Do this step again" results in an endless process, whereas "If you have done this step four times, then stop; otherwise, do it again" terminates. In summary, the latter part of the definition of an algorithm means that an algorithm must be both definable in a finite number of instructions and executable in a finite amount of time.

The use of the term *unambiguous* in the definition of an algorithm means that at each step the action to be performed next must be uniquely determined by the instruction and the data available at that time. Observe that this does not mean that the instruction alone must be enough to determine the desired action. Merely knowing

CamScanner

that a person is about to execute the "if" instruction from the previous paragraph is not enough information to determine what that person's action is going to be. We must also know how many times the person has already executed the instruction.

The requirement that each instruction in an algorithm be executable eliminates the possibility of an algorithm containing instructions whose execution is impossible. For instance, the sequence of instructions

1. Make a list of all the positive integers.
2. Arrange this list in descending order (from largest to smallest).
3. Extract the first integer from the resulting list.
4. Stop.

is not an algorithm because the execution of some of its instructions is impossible. One cannot make a list of all the positive integers as requested by instruction 1, and the positive integers cannot be arranged in descending order as requested by instruction 2.

To gain insight into algorithms, let us consider a simple example.

Example 4.1

There are 50 students in a class who appeared in their final examination. Their mark-sheets have been given to you. Write an algorithm to calculate and print the total number of students who passed in first division.

Algorithm 4.1

Step 1. Initialize TOTAL FIRST DIVISION and TOTAL MARKS SHEETS CHECKED to zero.

Step 2. Take the mark-sheet of the next student.

Step 3. Check the division column of the mark-sheet to see if it is FIRST DIVISION: if no, go to step 5.

Step 4. Add 1 to TOTAL FIRST DIVISION.

Step 5. Add 1 to TOTAL MARK-SHEETS CHECKED.

Step 6. Is TOTAL MARK-SHEETS CHECKED = 50: if no, go to step 2.

Step 7. Write TOTAL FIRST DIVISION.

Step 8. Exit.

It must be clear to the readers from this example that even for very simple problem, the development of algorithms is not so simple as it might initially appear and requires some thinking. It may also be noted from the given example that in order to solve a given problem, each and every instruction must be strictly carried out in a particular sequence. It is this fact which a beginner to problem solving by computers finds difficult to appreciate.

Example 4.2

Compute the distance travelled by a train in a specified time and speed. You are given with the data that a train travels at a speed of 160 km per hour and reaches its

CamScanner

destination in one hour and forty-five minutes. Write an algorithm to compute the distance travelled by train in the specific time.

Algorithm 4.2

Step 1. Start the calculation.
Step 2. Enter a value for SPEED.
Step 3. Enter a value for TIME.
Step 4. Calculate DISTANCE by using formula SPEED × TIME.
Step 5. Write the value of DISTANCE.
Step 6. Exit.

Pseudocode

Pseudocode is an English-like way of representing the solution to a problem. The prefix *pseudo* means *fake;* pseudocode, therefore literally means *fake code*—that is, not the code that is actually entered into the computer. It is considered a "first draft" because the pseudocode eventually has to be translated into a programming language. In general, a pseudocode is a notational system in which ideas can be expressed informally during the algorithm development process.

When using pseudocode to plan a program, you can concentrate on the logic and not worry about the rules of a specific language. It is also easy to change pseudocode if you discover a mistake in your logic—once it is coded in a programming language, most people find that it is more difficult to change logic. Pseudocode can be translated into a variety of programming languages, such as C, C++, Pascal or COBOL. Therefore, pseudocode instructions can be considered as midway between English and high-level languages but free from ambiguities of English language. Now from this point onward we will develop all the algorithms in pseudocodes.

Mathematical Notations and Functions

This section gives various mathematical functions which appear very often in the analysis of algorithms and in computer science in general, together with their notation.

Remainder Function

Let k be any integer and let M be a positive integer, then

$$k(\text{MOD } M) = \text{remainder}$$

(read k *modulo* M) will denote the integer remainder when k is divided by M.

Example 4.3

11(MOD 2) = 1
22(MOD 3) = 1
24(MOD 7) = 3
3(MOD 7) = 3
20(MOD 4) = 0
4(MOD 10) = 4

Arithmetic modulo M refers to the arithmetic operations of additions, multiplication and subtraction where the arithmetic value is replaced by its equivalent value in the set

$$\{1, 2, 3, \ldots, M\}$$

CamScanner

For example, in arithmetic modulo 12, sometimes called 'clock' arithmetic,

$$6 + 9 \equiv 3 \qquad 7 \times 5 \equiv 11 \qquad 1 - 5 \equiv 8 \qquad 2 + 10 \equiv 0 \equiv 12$$

(The use of 0 or M depends on the application.)

Integer Value Function

Let X be any real number then the *integer value* of X, written as INT(X), converts X into an integer by deleting (truncating) the fractional part of the number. Thus

INT(4.32) = 4 INT(5.392) = 5 INT(−8.7) = −8 INT(9) = 9

Absolute Value Function

The absolute value of the real number X, written as ABS(X) or $|X|$, is defined as the greater of X or $-X$. Hence

ABS (0) = 0
ABS (X) = X
ABS ($-X$) = X

Thus

ABS(−15) = 15 ABS(6) = 6 ABS(3.86) = 3.86 ABS(−0.21) = 0.21

Note that $|X| = |-X|$ and, for $X \neq 0$, $|X|$ is positive.

Factorial Function

The product of the positive integers from 1 to n is denoted by $n!$ (read 'n factorial'). That is,

$$n! = 1 \times 2 \times 3 \times \ldots \times (n-2) \times (n-1) \times n$$

It is also convenient to define 0! = 1.

Example 4.4

2! = 1 × 2 = 2 3! = 1 × 2 × 3 = 6 4! = 1 × 2 × 3 × 4 = 24

Algorithmic Notations

An algorithm, as stated above, is a finite step-by-step list of well-define instructions for solving a particular problem. Different symbols are used to define the different activities and which can be understood easily by the following formal presentation of an algorithm.

Algorithm 4.3

This algorithm takes two numbers X and Y and output their sum.

1. Read X. [To get first Number.]
2. Read Y. [To get second Number.]
3. Set SUM = X + Y. [Summing two Numbers.]
4. Write SUM. [Displaying result on Screen.]
5. Exit. [Termination of Program.]

CamScanner

The format for the formal presentation of an algorithm consists of two parts. The first part is a paragraph which tells the purpose of the algorithm, identifies the variables which occur in the algorithm and lists the input data. The second part of the algorithm consists of the list of steps that is to be executed.

Following paragraphs summarizes certain points and terminology used in making algorithm.

1. *Algorithm Name:* Each algorithm must be given a unique name for identification (*e.g.*, Algorithm 4.3 refer to the third algorithm in chapter 2).
2. *Step Number:* Each statement of the algorithm must be given a unique number for identification (*e.g.*, 1, 2, 3, etc.).
3. *Variable Names:* Variable names will use capital letters, as *X*, *Y*, SUM.
4. *Getting Input from the User:* 'Read' is used to get input from the user (computer ask the user for value).
5. *Assigning Value to a Variable:* 'Set' is used to assign value to a variable (computer itself sets the value for variable).
6. *Algorithm Termination:* 'Exit' is used to terminate the algorithm.
7. *Comments:* '[' and ']' are used for providing comments or remarks.
8. *Control Transfer:* Use 'Goto Step *n*' to transfer control to step *n*.

Flowcharts

If the planning done for the programs is retained as a part of the program documentation, it can then be used as a basis for future changes in the program. Once a computer program has been written to solve a particular problem, that program will continue to solve the problem in the same way for as long as it is used. A computer solution will continue to be a good solution until the nature of the problem changes. Unfortunately, the nature of all organizations is to be in a continual state of change. With the passage of time, the computer program will be solving problems that no longer exist—at least not in the same form as when the program was written.

When the time comes for a modification of the computer solution that will match the evolved problem, the original program plans can be used by the programmer to identify those portions of the program that require changes.

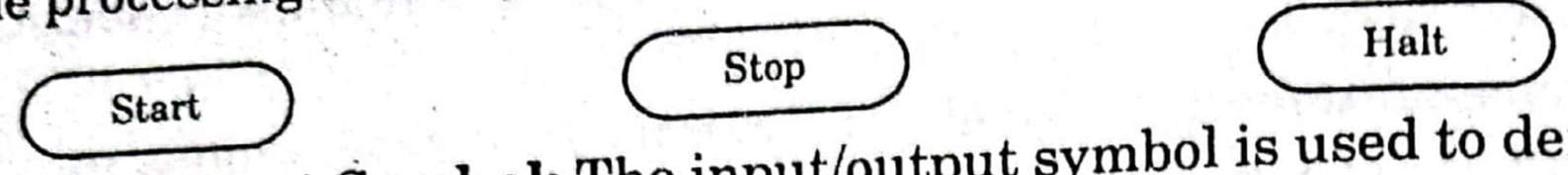
Flowcharting is one technique used in program planning. A *flowchart* is a pictorial representation of logic used in a computer program. It illustrates the major program elements and how they are logically integrated. The flowchart uses *arrows* to represent the direction of the program flow and *boxes* and other shapes to display actions. The *American National Standard Institute* (ANSI) has established standards for the symbols and boxes that are used in flowcharting. The shape of the boxes represents the type of activity that is to be performed. These symbols and examples of their use are illustrated below. Although many more symbols are included in the standard set, those shown below are encountered most frequently.

CamScanner

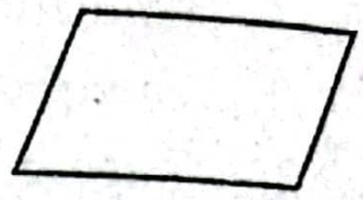
Terminal Symbol: The terminal symbol is used to indicate the beginning (START), ending (STOP) and pauses (HALT) in the program logic flow. It is the first symbol and the last symbol in the program logic. In addition, if the program logic calls for a pause in the program, that also is indicated with a terminal symbol. A pause is normally used in the program logic under some error conditions or if forms had to be changed in the computer's line printer during the processing of that program.



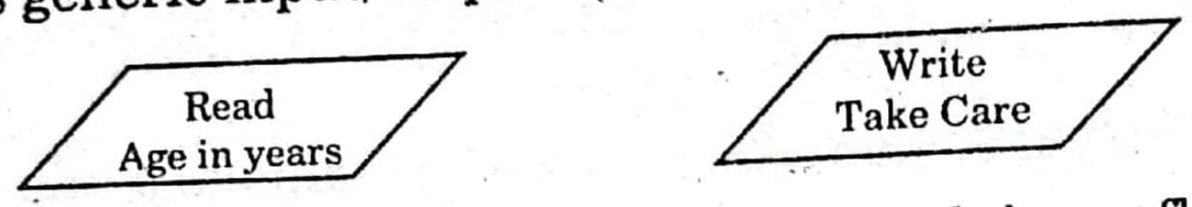
(Oval)



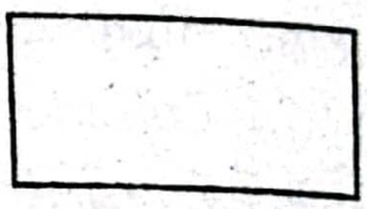
Input/Output Symbol: The input/output symbol is used to denote any type of input data or output information. This symbol represents generic input/output functions of all kinds.



(Parallelogram)



Process Symbol: A process symbol is used in a flowchart to represent an operation involved in actual processing of data. The process symbol is used for all arithmetic and data transfer (MOVE) operations. When more than one arithmetic and data movement instructions are to be executed consecutively, they are normally placed in the same processing box and they are assumed to be executed in the order of their appearance.



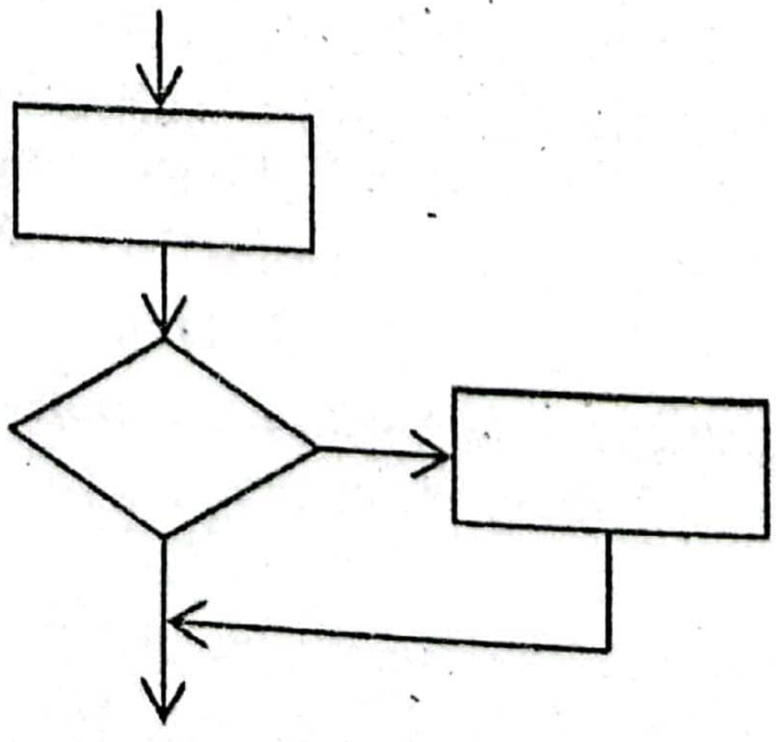
(Rectangle)



Flowlines: Flowlines with arrowheads are used to indicate the flow of operation, that is, the exact sequence in which the instructions are to be executed. The normal flow of flowchart is from top to bottom and left to right. Arrowheads are required only when the normal top to bottom flow is not to be followed. However, as a good practice and in order to avoid ambiguity, flowlines are usually drawn with an arrowhead at the point of entry to a symbol. Good practice also dictates that flowlines should not cross each other and that such intersections should be avoided whenever possible.

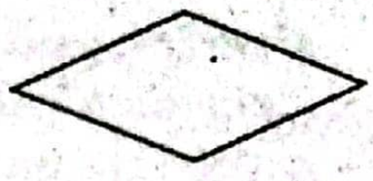


(Arrow)

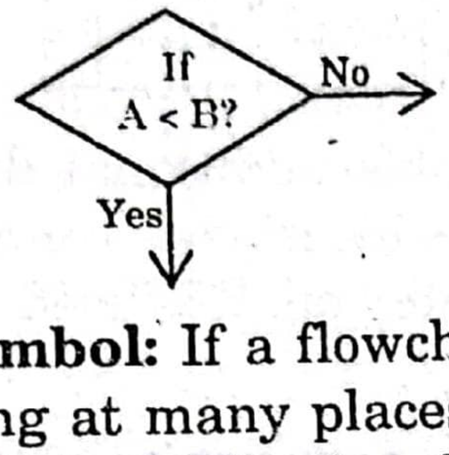


CamScanner

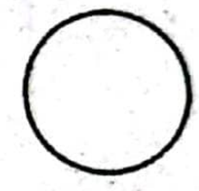
Decision Symbol: The decision symbol is used in a flowchart to indicate a point at which a decision has to be made and a branch to one of two or more alternative points is possible. The criterion for making the decision should be indicated clearly within the decision box. Moreover, the condition upon which each of the possible exit paths will be executed should be identified and all the possible paths should be accounted for. During execution, the appropriate path is followed depending upon the result of the decision.



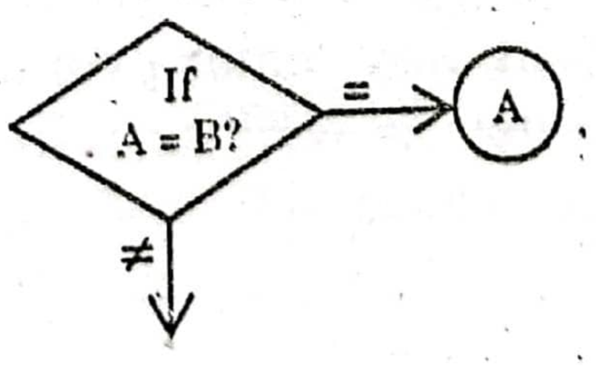
(Diamond)



On-Page Connector Symbol: If a flowchart becomes very long, the flowlines start crossing at many places that causes confusion and reduces understandability of the flowchart. In this situation it is useful to utilize the connector symbol as a substitute for flowlines. This symbol represents an entry from, or an exit to another part of the flowchart. An on-page connector symbol is represented by a circle and a letter or digit is placed within the circle to indicate the link. A pair of identically labelled connector symbols is commonly used to indicate a continued flow when the use of a line is confusing. So, two connectors with identical labels serve the same function as a long flowline. That is, they show an exit to some other chart section, or they indicate an entry from another part of the chart. How is it possible to determine if an on-page connector is used as an entry or an exit point? It is very simple: if an arrow enters but does not leave a connector, it is an exit point and program control is transferred to the identically labelled connector that does have an outlet. It may be noted that connectors do not represent any operation and their use in a flowchart is only for the sake of convenience and clarity.

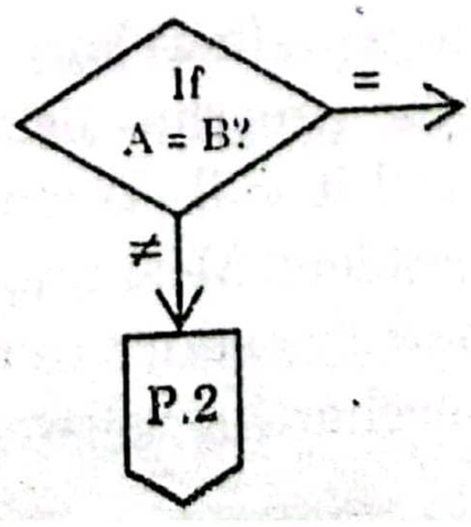


(Circle)



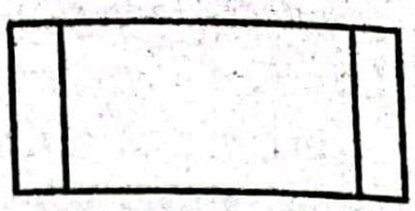


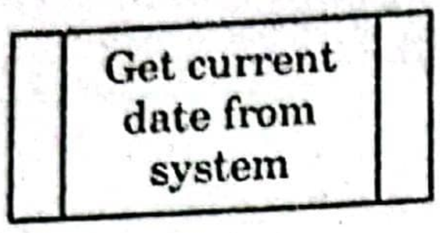
Off-Page Connector Symbol: This symbol is used to connect parts of the flowchart from one page to another.

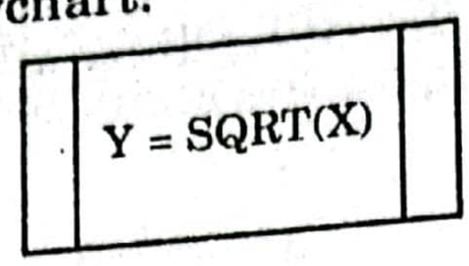


CamScanner

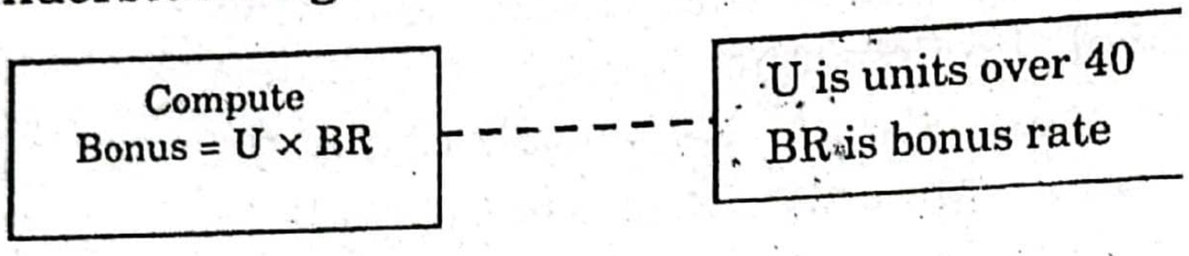
Predefined Process Symbol: This symbol is used to access ("perform" or "call") a subroutine not detailed at that particular point but usually defined in a separate micro flowchart. May be either procedures and functions that are used repeatedly or steps that would overly complicate the flowchart.





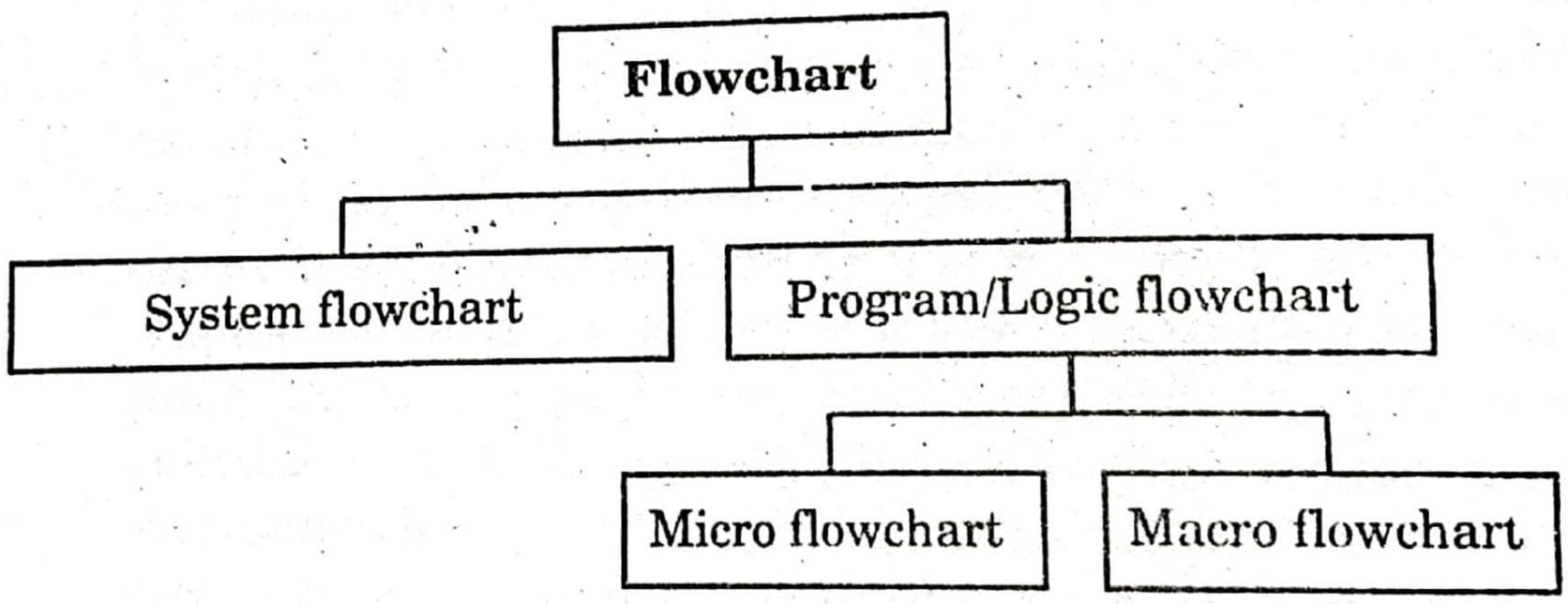


Annotation Symbol: This symbol holds additional descriptive comments, explanatory notes, or clarifications; connected by dashed line to the symbol being commented on. It is used as needed to aid understanding.



Categories of Flowcharts

The flowchart may be categorized as



Note that in this discussion we are talking about the *program/logic flowchart*, a flowchart that represents the flow of logic in a program. This flowchart is further classified as *micro*, a flowchart in which only major events are shown, and *macro*, a flowchart in which all the events (major as well as minor) are shown. A program flowchart is different from a *system flowchart*, which shows the flow of data through an entire computer system.

Flowcharts, which present a map of a solution, were the primary planning device for many years. They were favored over other methods because logic is easier to follow with pictures than with words. But flowcharts have some drawbacks: They are not easy to change, and they tend to be too detailed. A programmer draws a flowchart, then writes a program based on that flowchart; both the flowchart and the program listing are part of the program documentation. As the program changes—and it will—corresponding changes should be made on the flowchart. And therein lies the problem: Most programmers simply do not keep flowcharts up-to-date. However, there is a new procedure that makes flowcharts easier to plot: They can be drawn and revised using flowcharting software.

CamScanner

Composition of a flowchart requires consideration of the following four points. The first, and most obvious, is that the objective or problem, must be well defined. The next is the choice of starting point, followed by determination of the sequence of events leading towards the end result. Finally, there will be decisions resulting in more than one possible action.

Control Structures

In a program, *control structures* control how the program executes. Algorithms, flowcharts and their equivalent computer programs are more easily understood if they mainly use self-contained modules and three types of logic, or flow of control, called

1. Sequence logic, or sequential flow.
2. Selection logic, or conditional flow.
3. Iteration logic, or repetitive flow.

These three are considered the basic building blocks of all program construction.

Sequence

The *sequence control structure* is the most straightforward: One module simply follows another in sequence. The sequence may be presented by means of numbered steps or by the order in which the modules are written. Generally, most of the problems follow this elementary flow pattern.



Fig. 4.1. Flowchart general format of sequence control structure

We say that such an algorithm is a sequence of steps, meaning that

(a) At a time only one step is to be carried out.
(b) Every step is to be carried out once and only once; none is repeated and none is omitted.
(c) The order of executing the steps is the same as that in which they are written.
(d) Termination of the last step indicates the end of the algorithm.

Fig. 4.1 shows the general format of a sequence control structure as it is used in flowcharting: The three steps follow in sequence.

Example 4.5

This is the algorithm required to input three numbers from the keyboard of the personal computer and output the result.

CamScanner

1. Read X.
2. Read Y.
3. Read Z.
4. Set SUM = X + Y + Z.
5. Write SUM.
6. Exit.

This program starts with a declaration of variables followed by an instruction *Read* which tells the computer to take the numbers from the keyboard and store them as the variables X, Y, Z.

The next line instructs the computer to add together the contents of the stores identified by the name *SUM*. Lastly the instruction *Write SUM* outs the answer on the monitor.

Selection

The *selection control structure* employs a number of conditions which lead to a selection of one out of several alternative modules. The structures which implement this logic are called conditional structures or IF structures. The end of IF structure is frequently indicated by the statement "[End of IF structure.]" or some equivalent statement.

This control structure has two forms: IF and IF-ELSE. The IF selection is less complicated: IF the condition is true, then do something—but if it is not true, then do nothing. For example, IF the shift worked is shift 2, then add bonus of Rs. 500/-.

This structure has the form

IF condition, then
[Module A.]
[End of IF structure.]

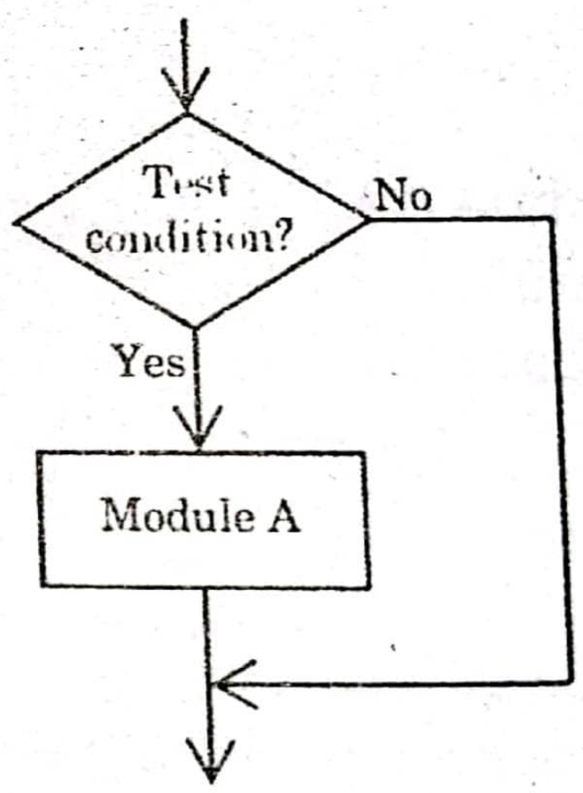


Fig. 4.2. Flowchart general format of IF structure

If the condition holds, then Module A, which may consist of one or more statements, is executed; otherwise Module A is skipped and control transfers to the next step of the algorithm. Fig. 4.2 shows the general format of IF structure in a flowchart.

IF-ELSE is more complicated than IF. The IF-ELSE control structure works as follows: IF (a condition is true), then (do something), ELSE (do something different). For example, IF a student is a resident, then the fee equals number of subjects times Rs. 700/-, ELSE fee is number of subjects times Rs. 900/-.

This structure has the form

CamScanner

```
IF condition, then
        [Module A.]
ELSE
        [Module B.]
[End of IF structure.]
```

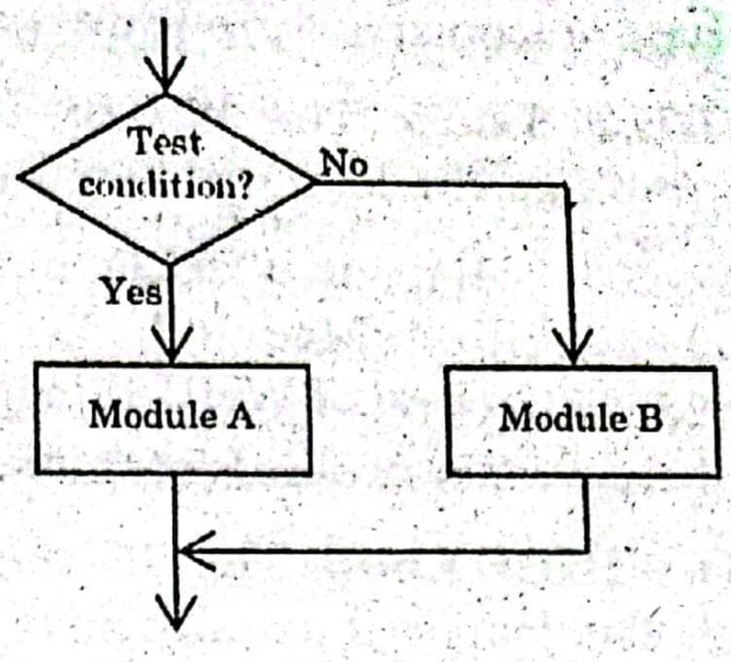


Fig. 4.3. Flowchart general format of IF-ELSE structure

If the condition holds, then Module A is executed; otherwise B is executed. Fig. 4.3 shows the general format of IF-ELSE structure in a flowchart.

Example 4.6

The problem is to input an examination mark and test it for the award of a grade. The mark is a whole number between 1 and 100. Grades are awarded according to the following criteria:

```
>= 80 distinction
>= 60 merit
>= 40 pass
< 40 fail
```

The algorithm is

```
1. Read mark.
2. IF mark >= 80 then
       Write distinction.
   ELSE IF mark >= 60 AND mark < 80 then
       Write merit.
   ELSE IF mark >= 40 AND mark < 60 then
       Write pass.
   ELSE mark < 40
       Write fail.
   [End of IF structure.]
3. Exit.
```

Iteration

Iteration is often referred to as a loop, because the program will keep repeating the activity until the condition becomes false. Loops simplify programming because they enable the programmer to specify certain instructions only once to have the computer execute them many times. WHILE, DO-WHILE, and FOR are three types of structures involved in loops. Each type begins with a Repeat statement and is followed by a module, called the *body of the loop*. The end of structure is frequently indicated by the statement "[End of loop.]" or some equivalent statement.

CamScanner

Each type of loop structure is discussed below:

(1) WHILE Loop: The *WHILE loop* uses a condition to control the loop at the beginning. The loop will usually have the form

```
Repeat WHILE condition
    [Module.]
[End of WHILE loop.]
```

Fig. 4.4 shows the general format of WHILE structure in a flowchart.

(2) DO-WHILE Loop: The *DO-WHILE loop* uses a condition to control the loop at the end. The loop will usually have the form

```
DO
    [Module.]
Repeat WHILE condition
[End of WHILE loop.]
```

(3) FOR Loop: The *FOR loop* uses an index variable, such as K, to control the loop. The loop will usually have the form:

```
FOR K = R to S by T
    [Module.]
[End of FOR loop.]
```

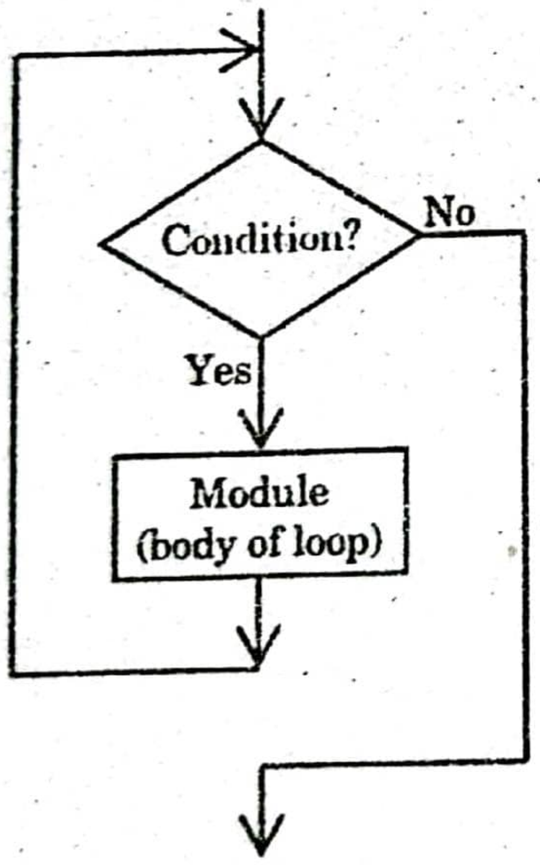


Fig. 4.4. Flowchart format of WHILE loop

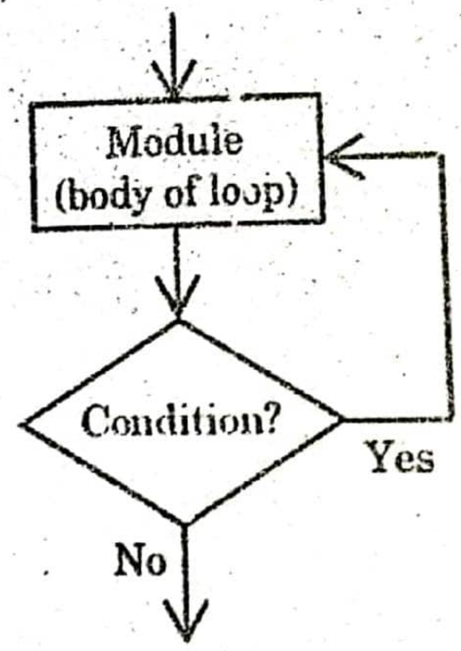


Fig. 4.5. Flowchart format of DO-WHILE loop

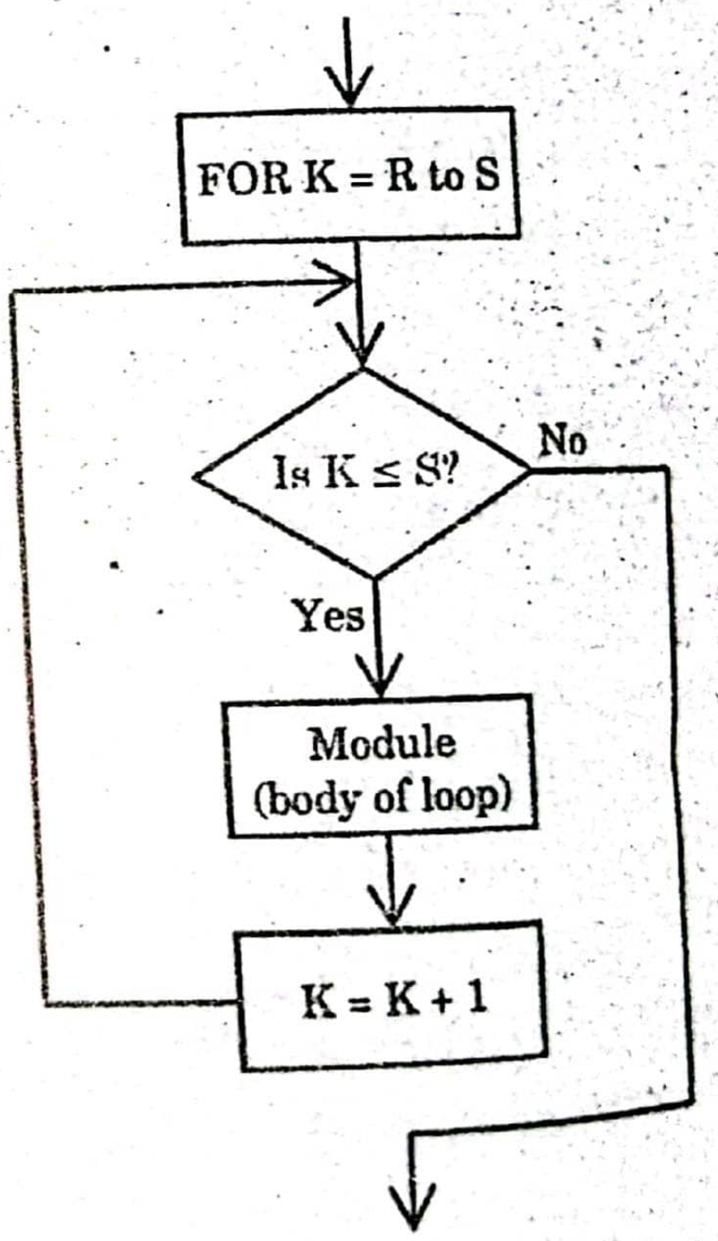


Fig. 4.6. Flowchart format of FOR loop

The WHILE and DO-WHILE loops are used for *indefinite repetitions*. That is, they do not define the number of repetitions that will occur when the loop is executed. They merely give conditions for looping to stop. The FOR loop is used for *definite repetition*. It defines the number of repetitions that will occur during execution. In Fig. 4.6 R is called the initial value, S the end value or test value. Observe that the body of the loop is executed first with K = R, then with K = R + 1, then with K = R + 2, and so on. The cycling ends when K = S.

CamScanner

If we put the loop-ending decision at the beginning, it is called a *leading decision;* if we put it at the end, it is called a *trailing decision*. The position of the decision constitutes the basic difference between WHILE and DO-WHILE structures. As Fig. 4.4 shows, WHILE tests at the beginning of the loop—the diamond-shaped decision box is the first action of the loop process. The DO-WHILE loop tests at the end, as we can see in Fig. 4.5. The DO-WHILE loop, by the way, guarantees that the loop statements are executed *at least once* because the loop statements are executed before we make any test about whether to get out. This guarantee is not desirable, depending on the program logic.

Example 4.7

Write the algorithm to add ten numbers using WHILE, DO-WHILE and FOR loops.

(a) Using **WHILE** loop.

1. Set X = 0 and S = 0.
2. Repeat WHILE X ≤ 10
   Read N.
   Set S = S + N.
   Set X = X + 1.
   [End of WHILE loop.]
3. Write S.
4. Exit.

(b) Using **DO-WHILE** loop.

1. Set X = 0 and S = 0.
2. DO
   Read N.
   Set S = S + N.
   Set X = X + 1.
   WHILE X ≤ 10
   [End of WHILE loop.]
3. Write S.
4. Exit.

(c) Using **FOR** loop.

1. Set S = 0.
2. FOR I = 1 to 10 by 1
   Read N.
   Set S = S + N.
   [End of FOR loop.]
3. Write S.
4. Exit.

CamScanner

Types of Flowcharts

There are four main types of flowcharts which can be combined in different way to represent any given problem. Each type can be recognized by the features given in the following descriptions.

Sequential Flowchart: This is the simplest type of flowchart, involving no decisions, branches or loops. An example is given in Fig. 4.7, which relates to a diner buying in a restaurant.

Branch (or Jump) Flowchart: In this type, one or more decision boxes are introduced, and the action taken depends upon whether the result of the decision is 'yes' or 'no'. In the example shown in Fig. 4.8, by the introduction of the extra three boxes: the 'leave a tip' operation is jumped over if the 'no' branch is taken from the decision box.

Loop Flowchart: A loop represents a procedure which may be repeated a definite or indefinite number of times. An example is given in Fig. 4.9, where the definition of a loop will be read over and over again until it is fully understood. This may require only one or two cycles, or if the reader has exceptional difficulty, a large number of cycles.

Nested Loops in a Flowchart: Consider the following problem. A man has a number of metallic rods, each of which is greater than two meters in length. Each rod is to be cut into lengths of two meters: the objective is to construct a flowchart which will represent a logical and systematic method of carrying out this task.

The appropriate flowchart is given by Fig. 4.10, where inner loop represents the procedure involved in cutting one rod into several lengths of two meters, and the outer loop represents the procedure involved in taking a new rod for the inner loop to process.

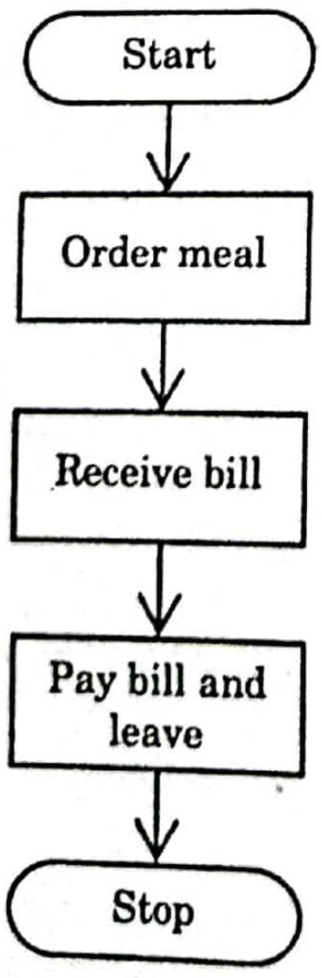


Fig. 4.7. Sequential flowchart

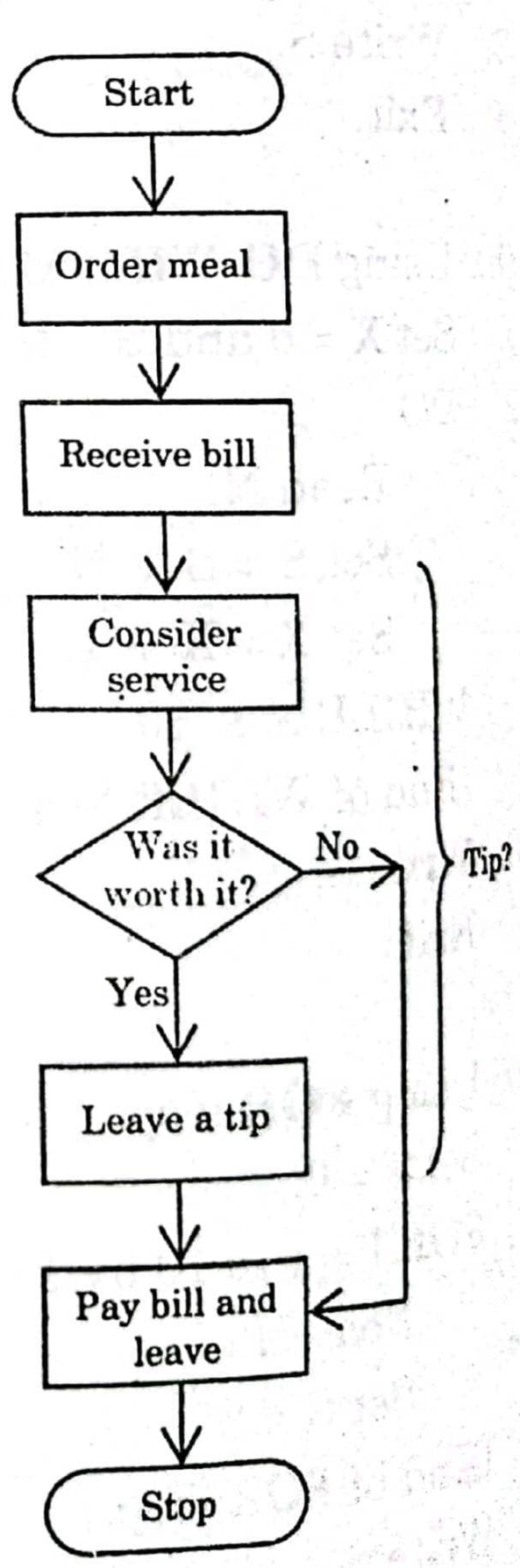


Fig. 4.8. Branch flowchart

CamScanner

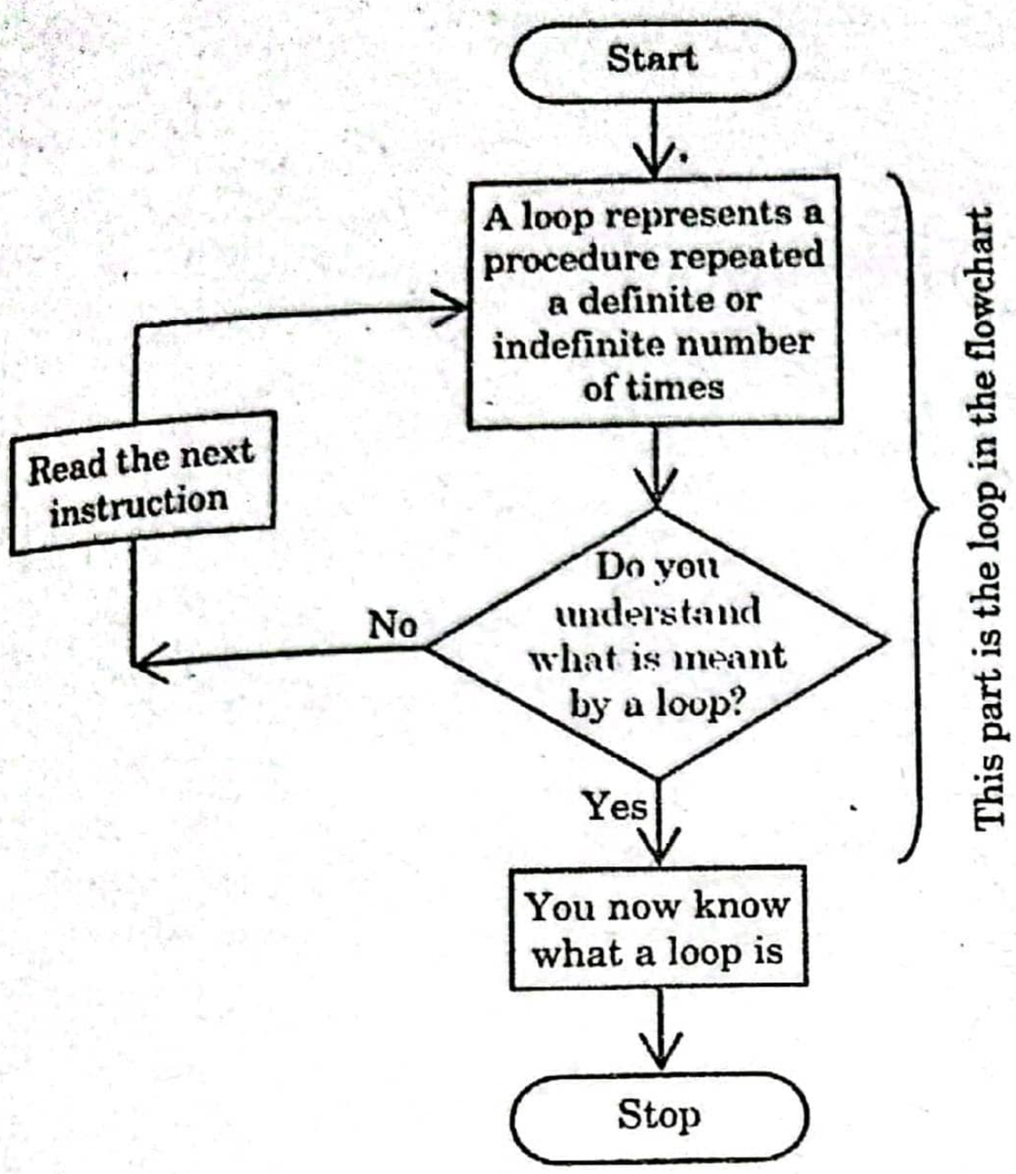


Fig. 4.9. Loop flowchart

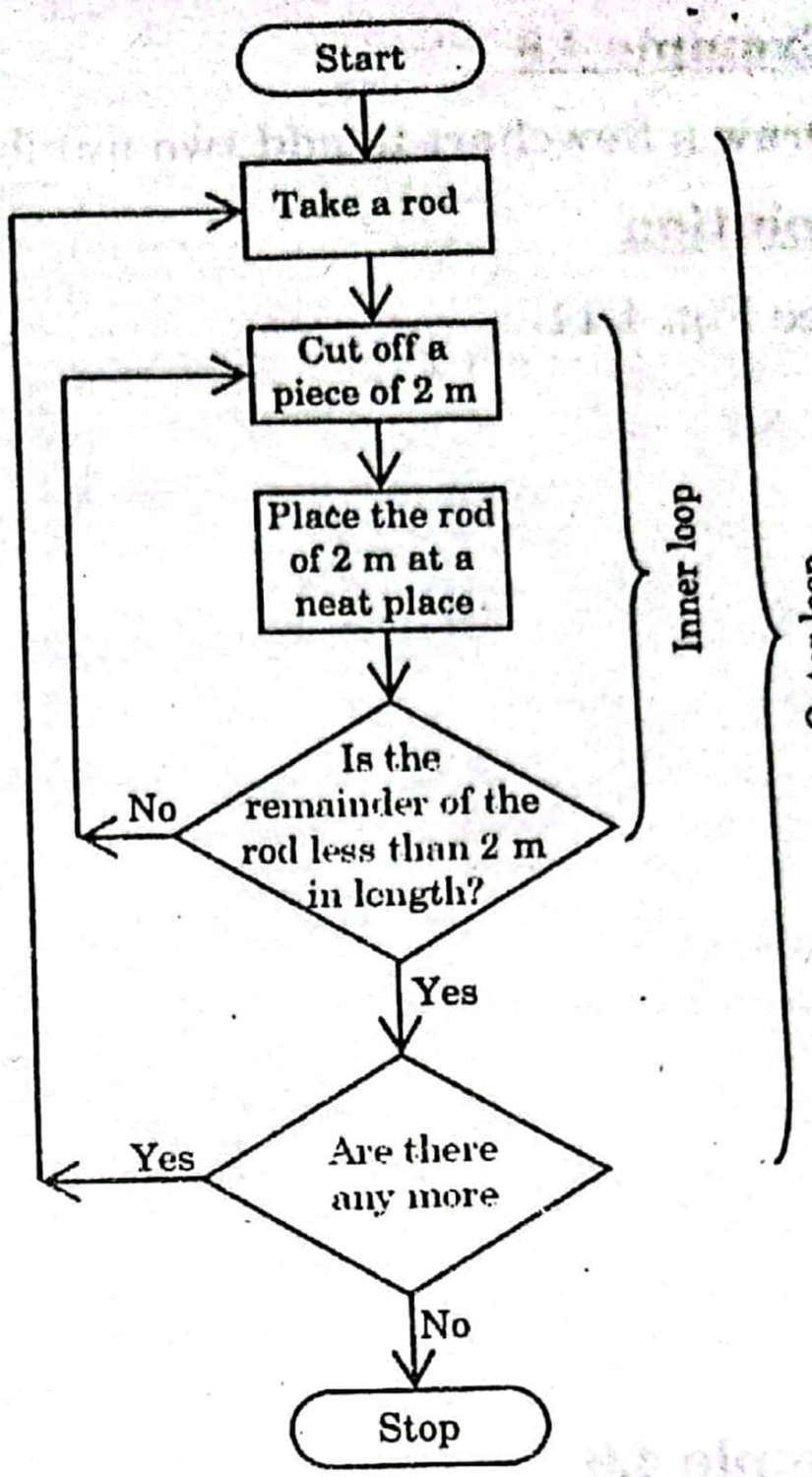


Fig. 4.10. Nested loops in a flowchart

Flowcharting Rules

While programmers have a good deal of freedom in creating flowcharts, there are a number of general rules and guidelines recommended by the *American National Standards Institute* (ANSI) to help standardize the flowcharting process. Various computer manufacturers and data processing departments usually have similar flowcharting standards. Some of these rules and guidelines are as follows:

1. First chart the main line of logic, then incorporate detail.
2. Maintain a consistent level of detail for a given flowchart.
3. Do not chart every detail or the flowchart will only be a graphic representation, step by step, of the program. A reader who is interested in greater details can refer to the program itself.
4. Words in the flowchart symbols should be common statements and easy to understand. It is recommended to use descriptive titles written in designer's own language rather than in machine oriented language.
5. Be consistent in using names and variables in the flowchart.
6. Go from left to right and top to bottom in constructing flowcharts.
7. Keep the flowchart as simple as possible. The crossing of flow-lines should be avoided as far as practicable.
8. If a new flowcharting page is needed, it is recommended that the flowchart be broken at an input or output point. Moreover, properly labelled connectors should be used to link the portions of the flowchart on different pages.

CamScanner

Example 4.8

Draw a flowchart to add two numbers.

Solution

See Fig. 4.11.

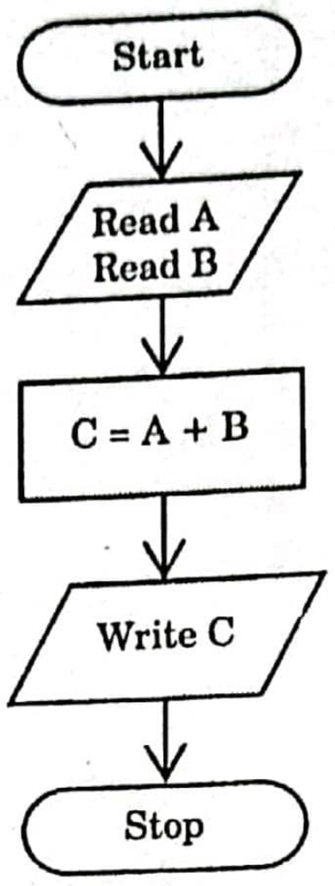


Fig. 4.11

Example 4.9

Draw a flowchart to generate numbers from 1 to 10.

Solution

See Fig. 4.12.

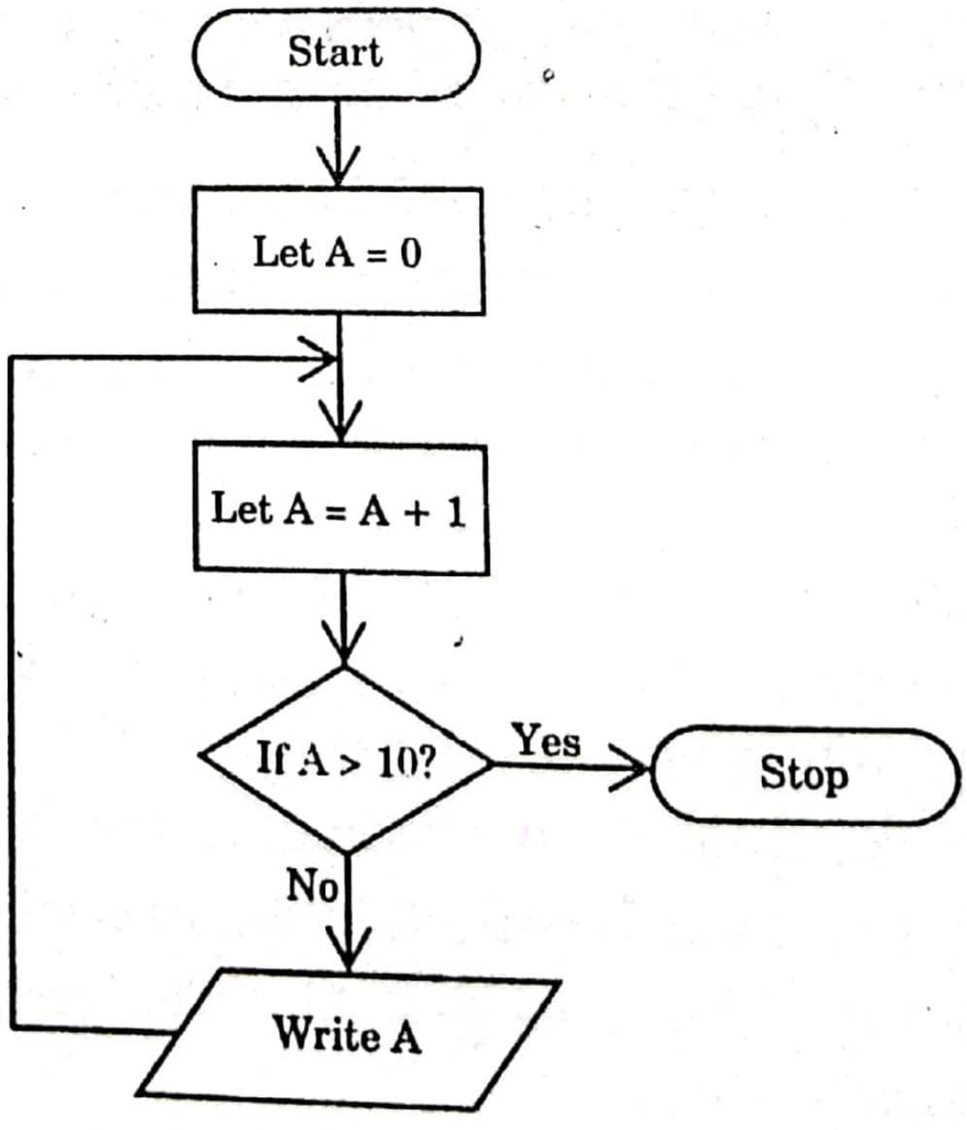


Fig. 4.12

CamScanner

Example 4.10

Draw a flowchart for summing the first thirty integers.

Solution

Fig. 4.13 is the flowchart for summing the first thirty integers. There we use SUM as the accumulator, and the variable K to produce the numbers 1, 2, 3, ..., 30. Observe that

SUM = 0 first assigns 0 to SUM.
When K = 1 SUM = SUM + K assigns 0 + 1 = 1 to SUM.
When K = 2 SUM = SUM + K assigns 1 + 2 = 3 to SUM.
When K = 3 SUM = SUM + K assigns 3 + 3 = 6 to SUM, and so on.
When K = 30, the value of SUM is 435, so

SUM = SUM + K assigns 435 + 30 = 465 to SUM.

This is the value of SUM when the Write statement is executed, so the output of the program is 465, which is the sum of the integers from 1 to 30.

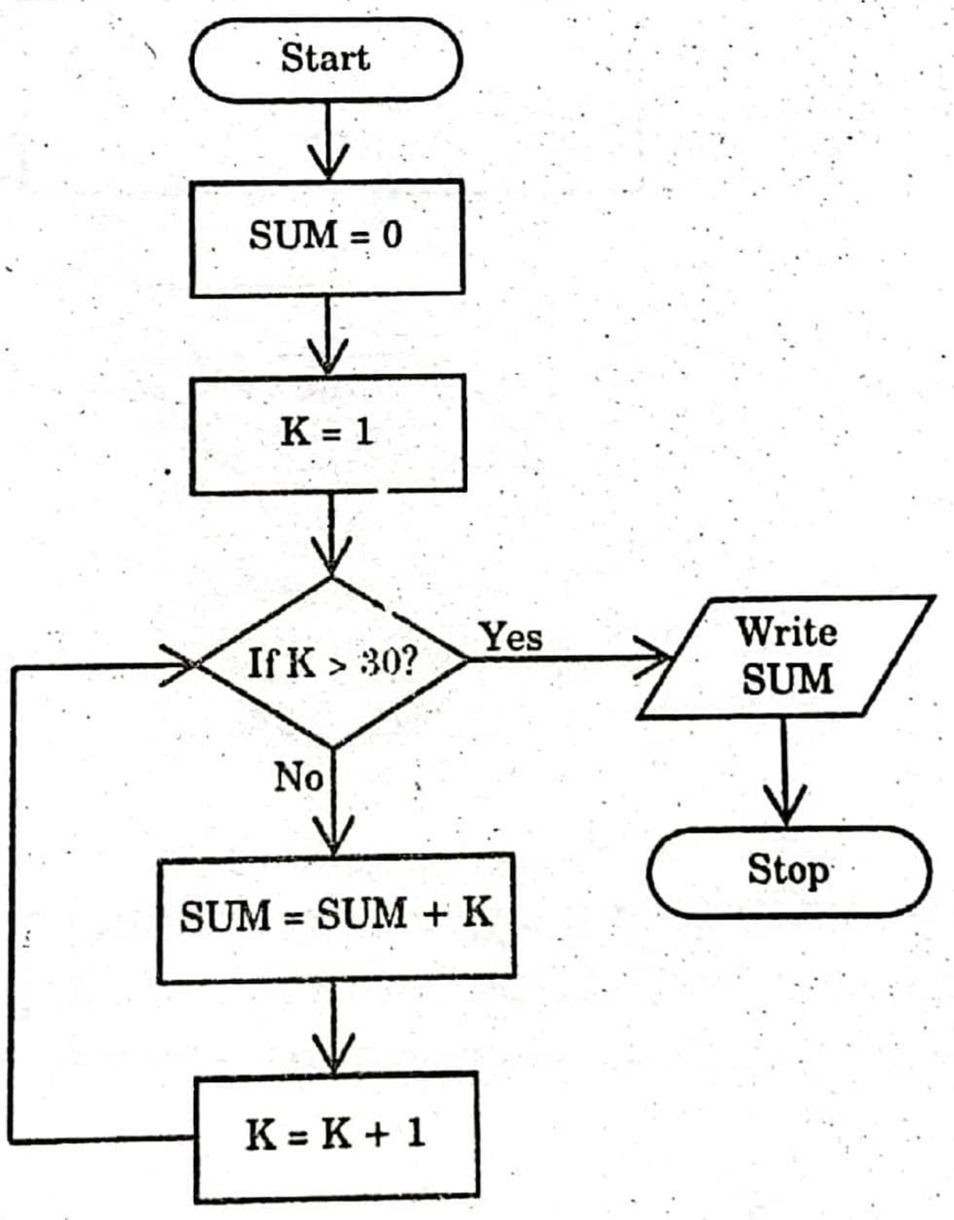


Fig. 4.13

Example 4.11

Draw a flowchart as follows:

- Start.
- Input age in years.
- if age < 13, display "Young".
- if age < 20 (but not < 13), display "Teenager".
- if age not < 20, display "Grownup".
- in the end, display "Take Care".
- Stop.

CamScanner

Solution

See Fig. 4.14.

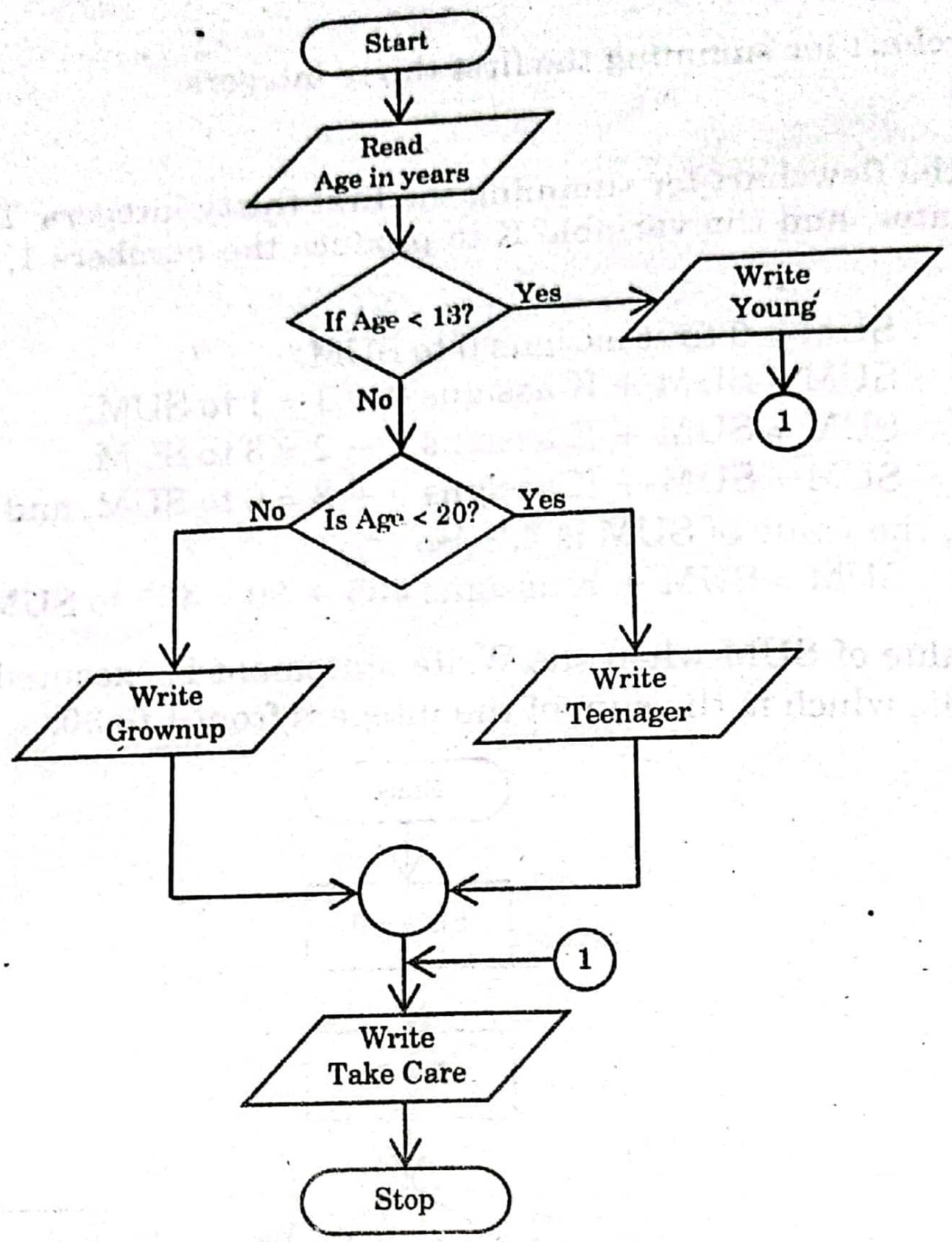


Fig. 4.14

Example 4.12

Compute the Net Pay (NP) of employees of a certain company. Data input for problem is Emp. Name and Basic Pay. Output is required Emp. Name, Basic Pay, Gross Pay (GP), Net Pay (NP), House Rent, Conveyance Allowance, Tax and Medical Allowance. Formula applied here is

1. House Rent = 20% × Basic Pay
2. Conveyance Allowance = 5% × Basic Pay
3. Medical Allowance = 10% × Basic Pay
4. Gross Pay = Basic Pay + House Rent + Conveyance Allowance + Medical Allowance
5. Tax depends upon Gross Pay as:

   If Gross Pay < 2000 then Tax = 0
   If Gross Pay > 2000 then Tax = 2% of Gross Pay
   If Gross Pay > 3000 then Tax = 3% of Gross Pay
   If Gross Pay > 4000 then Tax = 4% of Gross Pay
   If Gross Pay > 5000 then Tax = 5% of Gross Pay

CamScanner

Solution

See Fig. 4.15.
or Fig. 4.16.

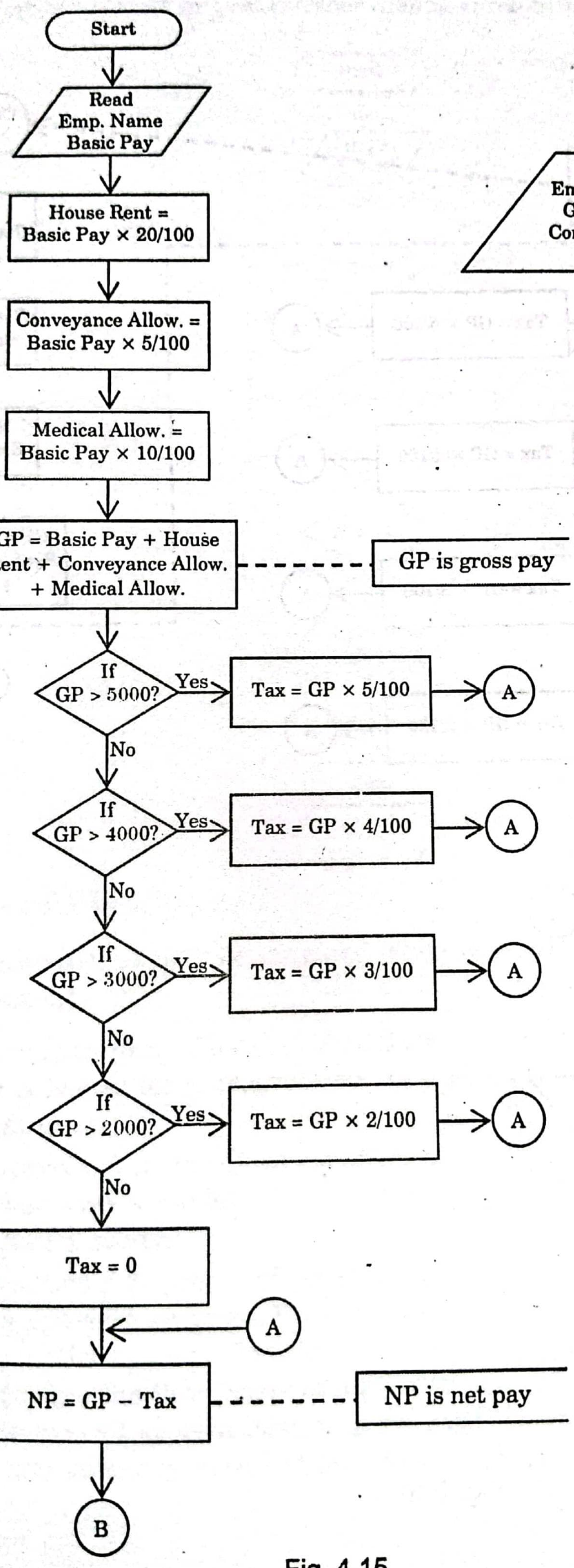


Fig. 4.15

CamScanner

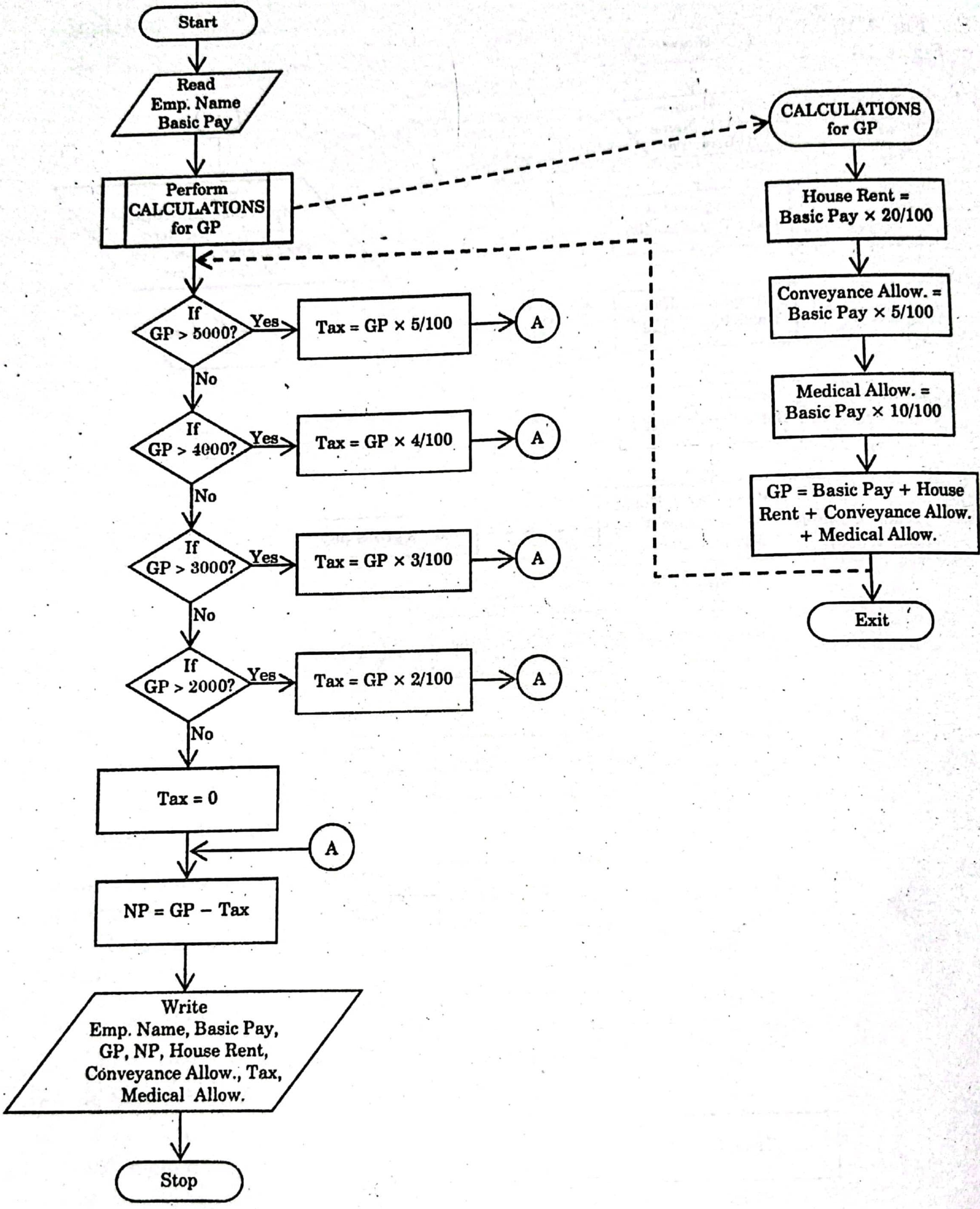


Fig. 4.16

CamScanner

Example 4.13

Draw a flowchart to identify largest number among three numbers?

Solution

See Fig. 4.17.

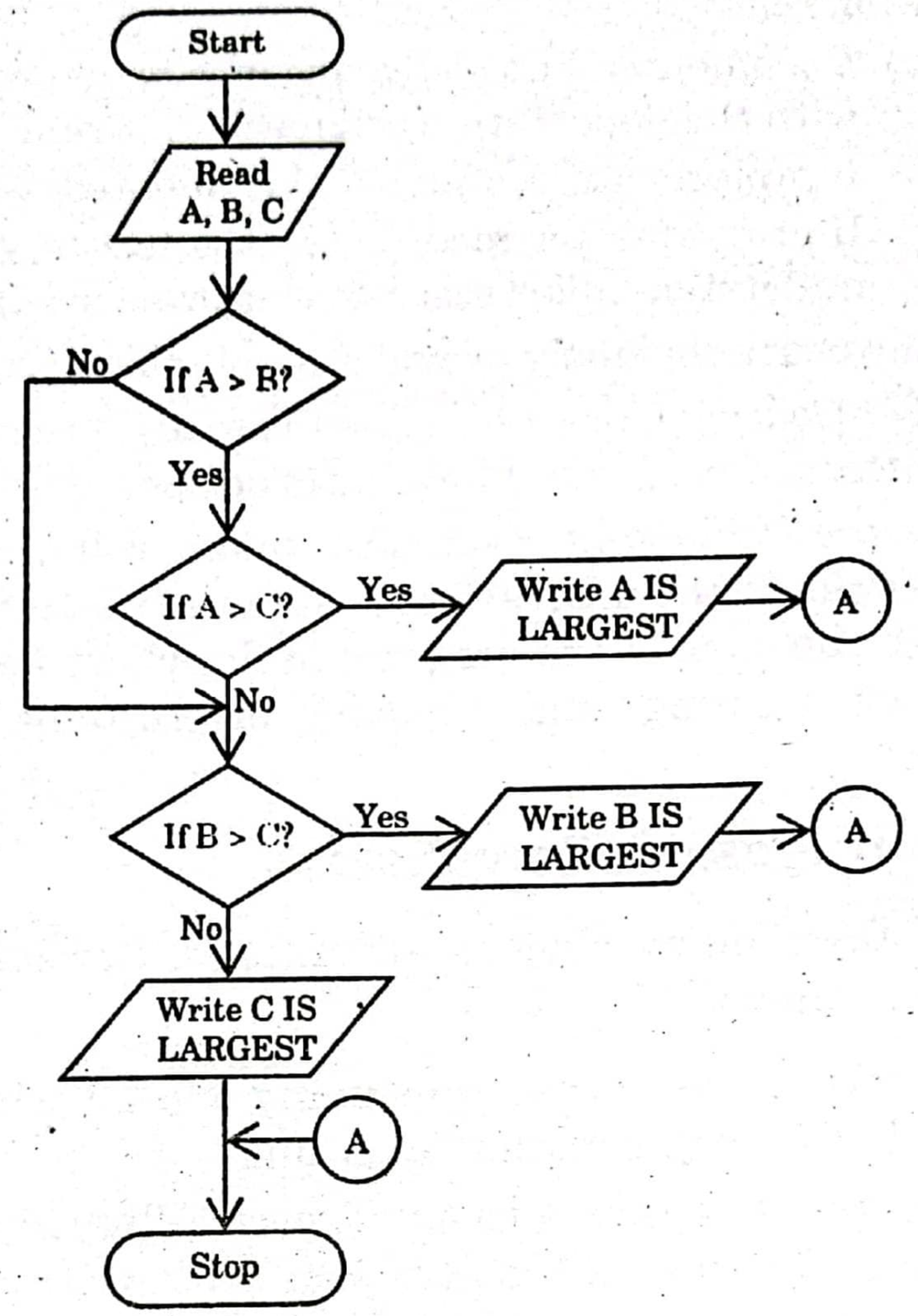


Fig. 4.17

Advantages of Flowcharts

The following advantages may be obtained when flowcharts are used for the purpose of program planning:

1. *Better Communication:* Since, a flowchart is a pictorial representation of a program, it is easier for a programmer to explain the logic of a program to some other programmer.
2. *Effective Analysis:* A macro-flowchart that presents the main lines of logic used in a program becomes a model of a program/system that can be broken down into detailed parts for study.
3. *Effective Synthesis:* A group of programmers are normally associated with the design of big software systems. Each programmer is responsible for designing only a part of the entire system. So initially, if each programmer draws a flowchart for his part of design, the flowcharts of all the programmers can be placed together to visualize the overall system design. Any problem in linking the various parts of the system can be easily detected at this stage and the design can be accordingly modified.

CamScanner

4. *Proper Program Documentation:* Program documentation involves collecting, organizing and storing a complete historical record of programs and the other documents associated with a system. Good documentation is needed for the following reasons:
   (a) Documented knowledge belongs to an organization and does not disappear with the departure (resignation/retirement) of a programmer.
   (b) If projects are postponed, documented work will not have to be duplicated.
   (c) If programs are modified in the future, the programmer will have a more understandable record of what was originally done.

   The flowchart obviously provides a valuable documentation support.
5. *Efficient Coding:* Once a flowchart is ready, programmers find it very easy to write the concerned program. Flowcharts act as a guide during program writing stage.
6. *Systematic Debugging:* Even after taking full care in program design, some errors may remain in the program. It is easier to find out the reason for the error by going through the logic of the program in flowchart form. A flowchart is very helpful in detecting, locating, and removing errors (bugs) in a program in a systematic manner.

Disadvantages of Flowcharts

In spite of their many obvious advantages, flowcharts have several disadvantages which are as follows:

1. Flowcharts are very time consuming and laborious to draw with proper symbols especially for large complex programs.
2. Flowchart tend to take up a lot of space, filling multiple sheets of paper to capacity. This can make them difficult to read and understand.
3. Owing to the symbol-string nature of flowcharting, any changes or modifications in the program logic will usually require a completely new flowchart.
4. There are no standards determining the amount of detail that should be included in a flowchart.

Because of these disadvantages, many organizations are now reducing the amount of flowcharting used.

CamScanner

Exercise

4.1. What is an algorithm?

4.2. What are the three control structures used in coding?

4.3. Explain the difference between the two loops (iteration) structures, WHILE and DO-WHILE.

4.4. Write algorithm showing the logic required to calculate and print the sales tax on a purchase of four items.

4.5. Write the algorithm for each of the following tasks listed sequentially:

(a) Write algorithm to generate numbers from 1 to 10. (using WHILE loop).

1. Set X = 0.
2. WHILE X < 10
   Set X = X + 1.
   Write X.
   [End of WHILE loop.]
3. Exit.

(b) Write algorithm to generate numbers from 1 to 10. (using IF-ELSE structure).

1. Set X = 0.
2. Set X = X + 1.
3. Write X.
4. IF X = 10, then
   Exit.
   ELSE
   Goto step 2.
   [End of IF structure.]

(c) Write algorithm to write any number 10 times.

1. Read N.
2. FOR K = 1 to 10 by 1
   Write N.
   [End of FOR loop.]
3. Exit.

(d) Write algorithm to add numbers till input zero is given.

1. Set S = 0.
2. WHILE N ≠ 0
   Read N.
   Set S = S + N.
   [End of WHILE loop.]
3. Write S.
4. Exit.

CamScanner

(e) Write algorithm to write odd numbers from 1 to 11.

1. FOR K = 1 to 11 by 2
   Write K.
   [End of FOR loop.]
2. Exit.

(f) Write algorithm to take sum of array of 10-elements (first take input and then add).

1. FOR K = 1 to 10
   Read N[K].
   [End of FOR loop.]
2. Set S = 0.
3. FOR K = 1 to 10
   Set S = S + N[K].
   [End of FOR loop.]
4. Write S.
5. Exit.

4.6. What is a flowchart and how many types of flowcharts are there?

4.7. What are the various symbols used in flowcharting? Give their pictorial representation.

4.8. Describe the function of the various flowcharting symbols.

4.9. Determine the appropriate flowchart symbol in which each of the following instructions or questions should be placed:

(a) Read A, B, C
(b) Is A = 0?
(c) Add B to C giving A
(d) Start
(e) Is A is less than B?

Ans.

(a) Parallelogram, since it is an input instruction.
(b) Diamond, since it asks a question.
(c) Rectangle, since it is a process (calculation).
(d) Oval.
(e) Diamond, since it asks a question.

4.10. Determine the minimum number of arrows connected to a diamond in a flowchart.

Ans. There must be at least one arrow going to the diamond, and at least two arrows leaving the diamond; hence there are at least three arrows connected to a diamond.

4.11. Why are there standards for the symbols used in drawing flowcharts?

4.12. How does a flowchart help a programmer in program development?

4.13. What are the various guidelines to be followed while drawing a flowchart?

4.14. What are the advantages of flowcharts when they are used to solve a problem?

CamScanner

4.15. What are the disadvantages of flowcharts?

4.16. How can a counter be used to keep track of the number of times a loop has been executed?

4.17. Is it possible to have more than one flowchart for a given problem? Give reasons for your answer.

4.18. Three numbers, denoted by the variables A, B, and C are supplied as input data. Draw a flowchart for the logic to pick and print the smallest of the three numbers.

4.19. Analyze the flowchart shown in Fig. 4.18 and list the final output.

4.20. Consider the flowchart in Fig. 4.19. Find the output if the input is (a) A = 5, B = 10, C = 15 and (b) A = 15, B = 10, C = 5.

Ans.

(a) Since A is less than B, BONUS = 100 is executed, and 100 is outputted.

(b) A is greater than B, and B is greater than C; so BONUS = 300 is executed, and 300 is outputted.

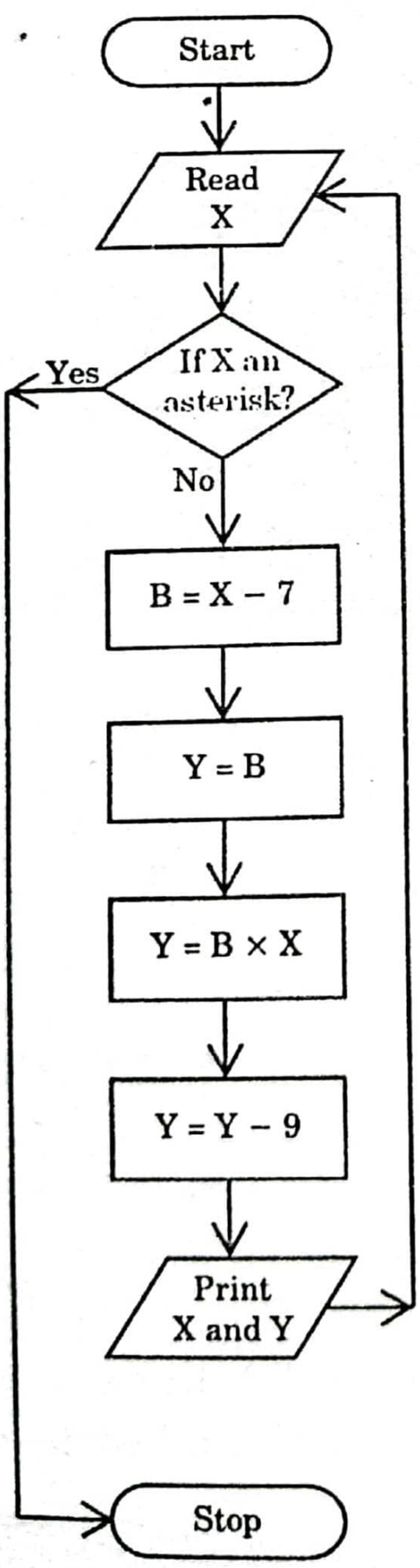


Fig. 4.18

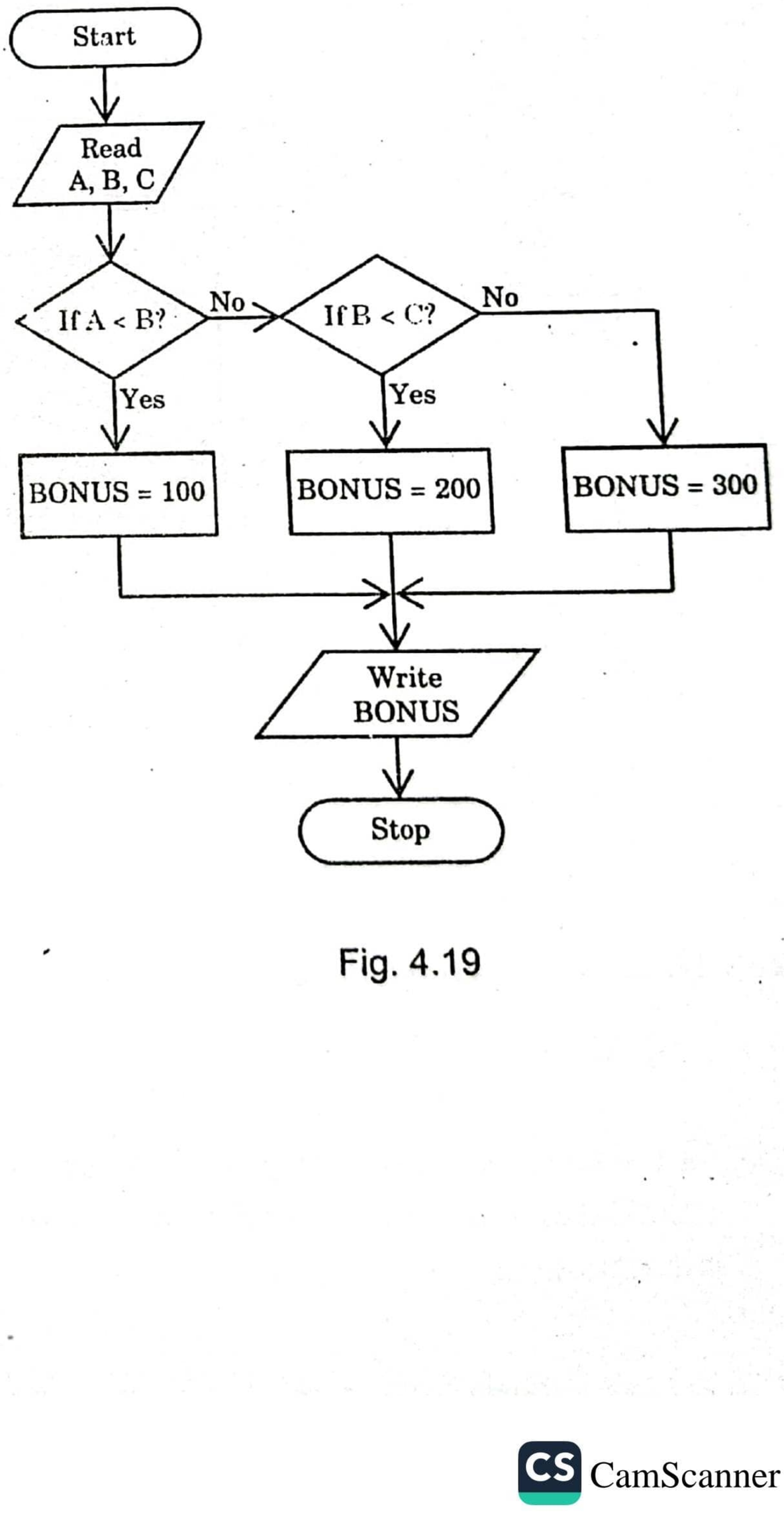


Fig. 4.19

CamScanner

4.21. The first 20 records in a data set are to be read and printed. Draw a flowchart to do this job. Make sure that the processing stops after the twentieth record.

4.22. Draw a flowchart for the logic to find out whether a given triangle ABC is a right-angled triangle. Assume that the sides of the triangle are supplied as input data. Print the answer as yes or no.

4.23. Consider Fig. 4.20. Find the output if the input is (a) A = 5, B = 10; (b) A = 20, B = 10.

Ans.

(a) 40; (b) 40

4.24. Consider the flowchart in Fig. 4.21. Find the output if the input is (a) A = 5, B = 10, C = 20; (b) A = 30, B = 10, C = 20.

Ans.

(a) 40; (b) 70

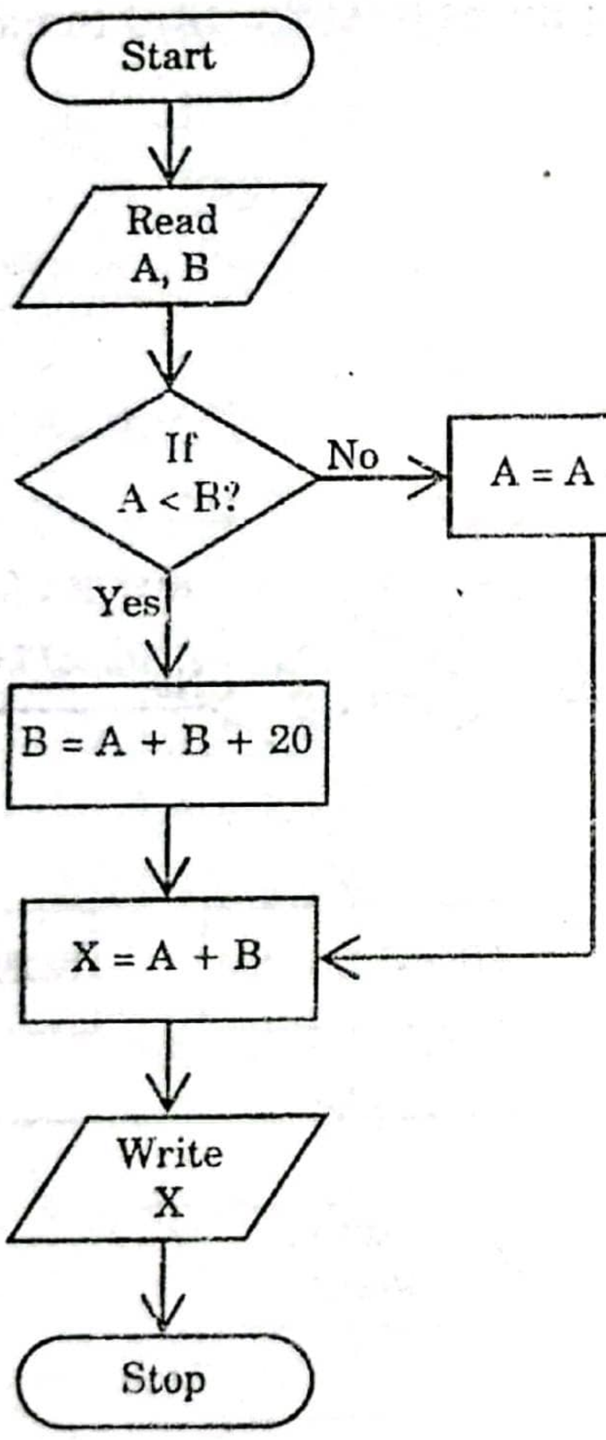


Fig. 4.20

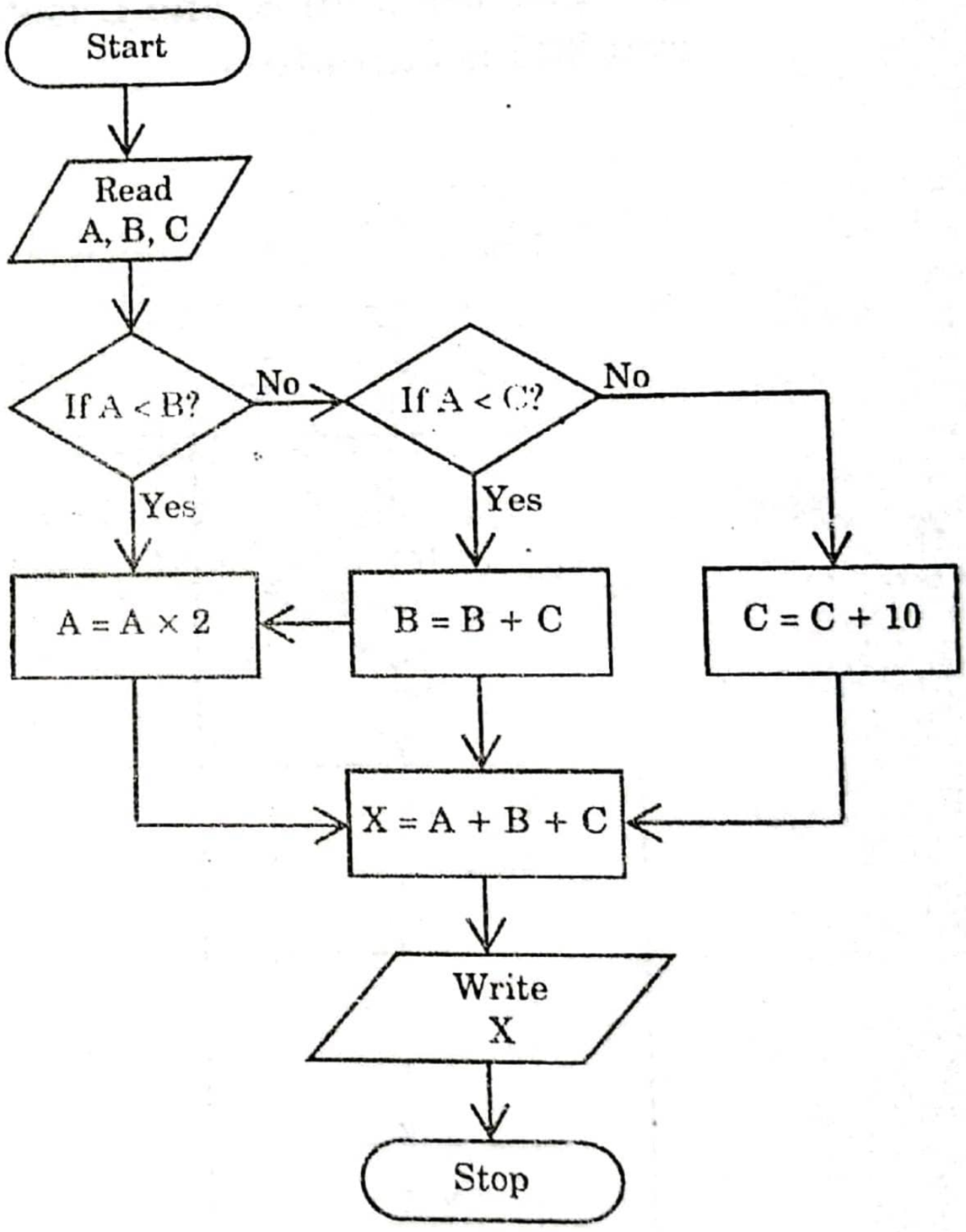


Fig. 4.21

4.25. Draw a flowchart that calculates:

$$\text{SUM} = \frac{1}{1} + \frac{1}{3} + \frac{1}{5} + \ \ldots \ + \frac{1}{25}$$

4.26. Consider the equation $y = x^4 + 2x^3 - 4x^2 - 16x + 32$. Draw a flowchart that calculates y for values of x from -4 to 4 in step of 0.25, and write each x and its corresponding y.

CamScanner

Objectives and MCQs

Choose the answer that best completes the statement.

4.1. A pictorial representation of a problem is called
- (a) a flowchart
- (b) an algorithm
- (c) a pseudocode
- (d) None of the above

4.2. Which of the following is *not* one of the three program logic constructs?
- (a) simple sequence
- (b) condition
- (c) loop
- (d) go to

4.3. What do you call the step-by-step solution to a programming problem?
- (a) recipe
- (b) structure chart
- (c) syntax
- (d) algorithm

4.4. What does the ◇ symbol in a flowchart represent?
- (a) decision
- (b) process
- (c) start/stop
- (d) predefined process

4.5. What does the ▱ symbol in a flowchart represent?
- (a) process
- (b) terminal
- (c) input/output procedure
- (d) predefined process

4.6. Which of the following control structures use DO-WHILE?
- (a) sequence
- (b) selection
- (c) iteration
- (d) None of the above

CamScanner

Objectives

4.1. A pictorial representation of a problem is made by a flowchart.

4.2. In micro flowchart only major events are shown.

4.3. In macro flowchart all the events (major as well as minor) are shown.

4.4. The oval symbol is used to represent the beginning of a program.

4.5. The rectangle symbol in a flowchart represents process.

4.6. At least two arrows leave the diamond in a flowchart.

4.7. The selection control structure is frequently shown in algorithm as an IF-ELSE statement.

4.8. Match the following symbols with their functions.

(a) Oval	(a) Decision
(b) Parallelogram	(b) Connector
(c) Circle	(c) Terminal
(d) Rectangle	(d) Input/output of data
(e) Diamond	(e) Process

CamScanner

CHAPTER 5

AN OVERVIEW OF C++ LANGUAGE

Introduction to C++

C++ is a powerful programming language developed by Bjarne Stroustrup at Bell (now AT & T's Bell) Laboratories of USA in 1983, and is basically a superset of C language. Hence all standard C features are available within C++. However, C++ is not merely an extension of C. Rather it incorporates several new fundamental concepts that form a basis for object-oriented programming (OOP)—a new programming paradigm that is of interest to professional programmers. C++ mainly developed to provide a programming tool, for designing complex software projects, having the following characteristics:

1. Be helpful in reducing the complexity while solving problems.
2. Ensure correctness of results.
3. Be affordable in terms of hardware and resources.
4. Be open—easy and cheap to integrate existing software libraries and facilities.
5. Be portable.

The rapid expansion of C++ over various types of computers prompted many software companies to release their own complier of C++. Dev-C++, an Integrated Development Environment (IDE) for the C and C++ programming languages, was created by Colin Laplace and released in 1998. Dev-C++ quickly gained popularity among beginner programmers due to its simplicity and user-friendly interface. One of the key features

CamScanner

of Dev-C++ was its integration with the MinGW compiler, which allowed developers to compile and run C and C++ programs without the need for additional software.

Advantages of C++

C++ has many advantages over other programming languages. Among them are:

1. Object-Oriented: C++ is an object-oriented programming language which means that the main focus is on objects and manipulations around these objects. This makes it much easier to manipulate code, unlike procedural or structured programming which requires a series of computational steps to be carried out.

2. Portability: C++ offers the feature of portability or platform independence which allows the user to run the same program on different operating systems or interfaces at ease.

Suppose you write a program in Linux OS and for some apparent reason you switch to Windows OS, you would be able to run the same program in windows as well without any error. This feature proves to be of great convenience to the programmer.

3. Speed: When speed is a critical metric, C++ is the most preferred choice. The compilation and execution time of a C++ program is much faster than most general-purpose programming languages.

4. Compiled: Unlike other programming languages where no compilation is required, every C++ code has to be first compiled to a low-level language and then executed.

5. Rich Library Support: The C++ Standard Template Library (STL) has many functions available to help write code quickly.

6. Multi-paradigm: C++ is a multi-paradigm programming language. The term 'Paradigm' refers to the style of programming. It includes logic, structure, and procedure of the program.

7. Pointer Support: C++ also supports pointers which are often not available in other programming languages.

8. Closer to Hardware: C++ is closer to hardware than most general-purpose programming languages. This makes it very useful in those areas where hardware and software are closely coupled together, and low-level support is needed at the software level.

9. Memory Management: C++ gives the programmer the provision of total control over memory management. This can be considered both as an asset and a liability as this increases the responsibility of the user to manage memory rather than it being managed by the Garbage collector. This concept is implemented with the help of DMA (Dynamic memory allocation) using pointers.

10. Large Community Support: C++ has a large community that supports it by providing online courses and lectures, both paid and unpaid. Statistically speaking, C++ is the 6th most used and followed tag on StackOverflow and GitHub.

11. Compatibility with C: C++ is pretty much compatible with C. Virtually, every error-free C program is a valid C++ program. Depending on the compiler used, every

CamScanner

program of C++ can run on a file with .cpp extension.

12. Scalability: Scalability refers to the ability of a program to scale. It means that the C++ program is capable of running on a small scale as well as a large scale of data. We can also build applications that are resource intensive.

The IDE of Dev-C++ Compiler

Dev-C++, developed by Embarcadero since 2020, is a fully featured graphical IDE (Integrated Development Environment) and code editor for the C/C++ programming language. It uses MinGw (Minimalist GNU for Windows) compiler system to create Windows as well as Console based C/C++ applications. It can also be used with any other GCC (GNU Compiler Collection) based compiler like Cygwin. Dev-C++ is free software and is distributed under the GNU General Public License. Thus, we can distribute or modify the IDE freely. Fig. 5.1 shows Dev-C++ version 6.3 IDE.

To download Dev-C++ installer, go to the following link:

`https://sourceforge.net/projects/embarcadero-devcpp/files/`

OR

`https://www.dropbox.com/s/9xluz5mgcuk023e/Dev_C_v6.3.zip?dl=0`

You should let the installer put Dev-C++ in the default directory of `C:\Program Files(x86)\Embarcadero\Dev-Cpp`, as it will make it easier to later install add-ons or upgrades.

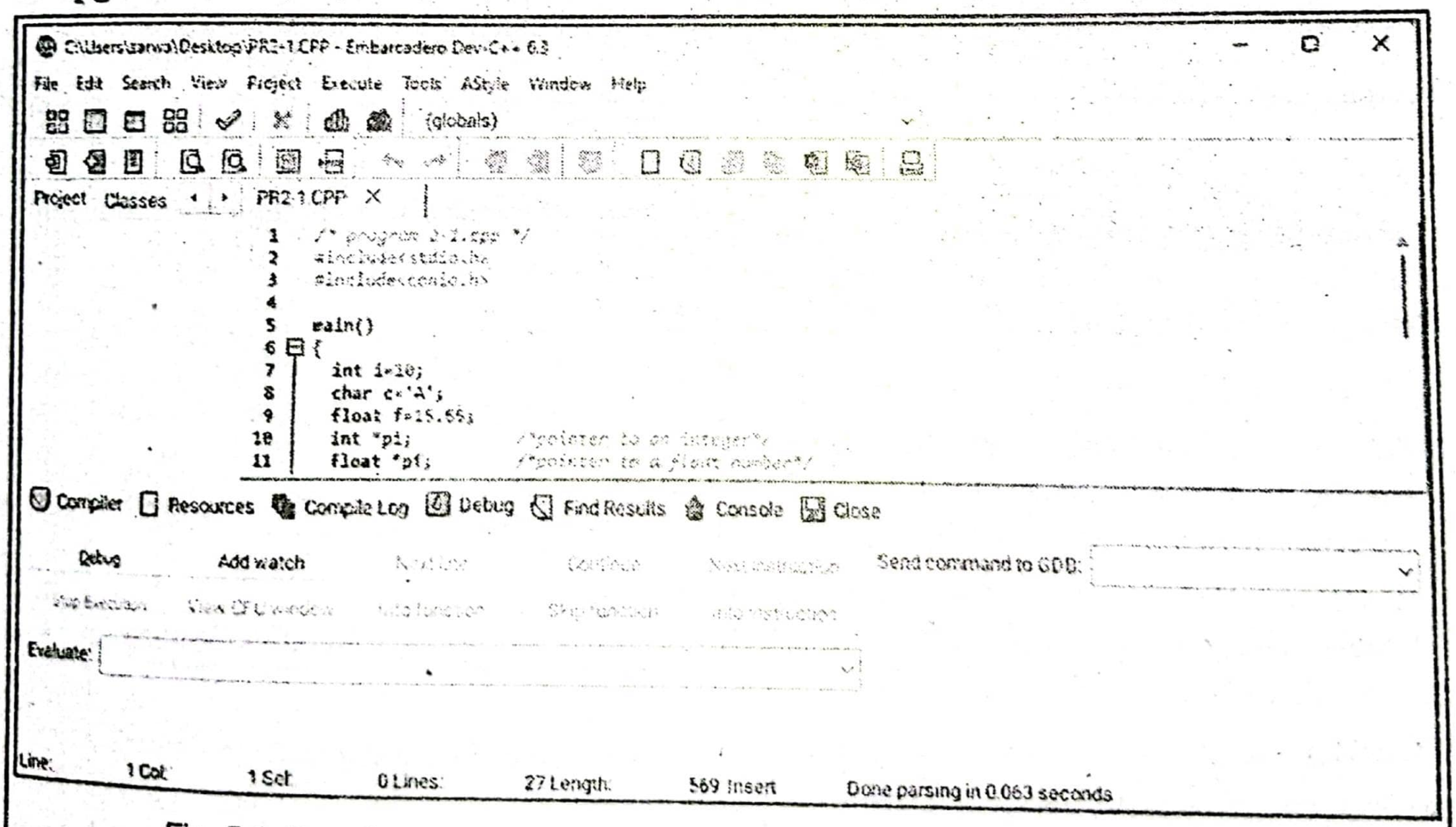


Fig. 5.1. Dev-C++ version 6.3 Integrated Development Environment (IDE).

Features of Dev-C++ IDE

Enlisted below are some of the features of this IDE that help us develop efficient and user-friendly C++ applications.

CamScanner

1. Dev-C++ supports GCC-based compilers including Cygwin, MinGW, etc. We can either install a Dev-C++ IDE along with the compiler integrated or just an IDE if we already have a compiler on our system.
2. We can use integrated debugging (using GNU Debugger) with this IDE. The debugger allows us to perform all the general debugging operations on source code.
3. It has a localization feature that provides support for multiple languages. We can select the language the first time when we open the IDE after installing it. We can also change the language anytime using settings.
4. Like the other IDEs, this IDE also provides the 'Auto-Completion' feature for the code we write.
5. It comes with customizable syntax highlighting editor that can make the source code more readable.
6. Allows to Edit and Compile the resource files.
7. Has a Tool Manager that contains various tools that can be used in the project.
8. This IDE also has inbuilt Find and Replace facilities.
9. Using Dev-C++ IDE, we can create various types of applications be it Windows, Console, Static libraries or DLLs.
10. We can also create our own project templates to create our own project types.
11. It has a Project Manager that helps us to manage various projects.
12. Also provides print support through its interface.
13. We can easily install the add-on libraries using the package manager provided by the IDE.

Using Dev-C++

Step 1: Configure Dev-C++

After starting the IDE, the first thing we need to ensure is the setting for debugging information to be generated.

- Go to the Tools → Compiler Options.
- In the 'Settings' tab, click on 'Linker' tab, and change 'Generate debugging information' to 'Yes' (see Fig. 5.2).
- Also, in the 'Settings' tab, click on 'Code Generation' tab, and select 'Language standard' to 'ISO C++11'.
- Click 'OK'.

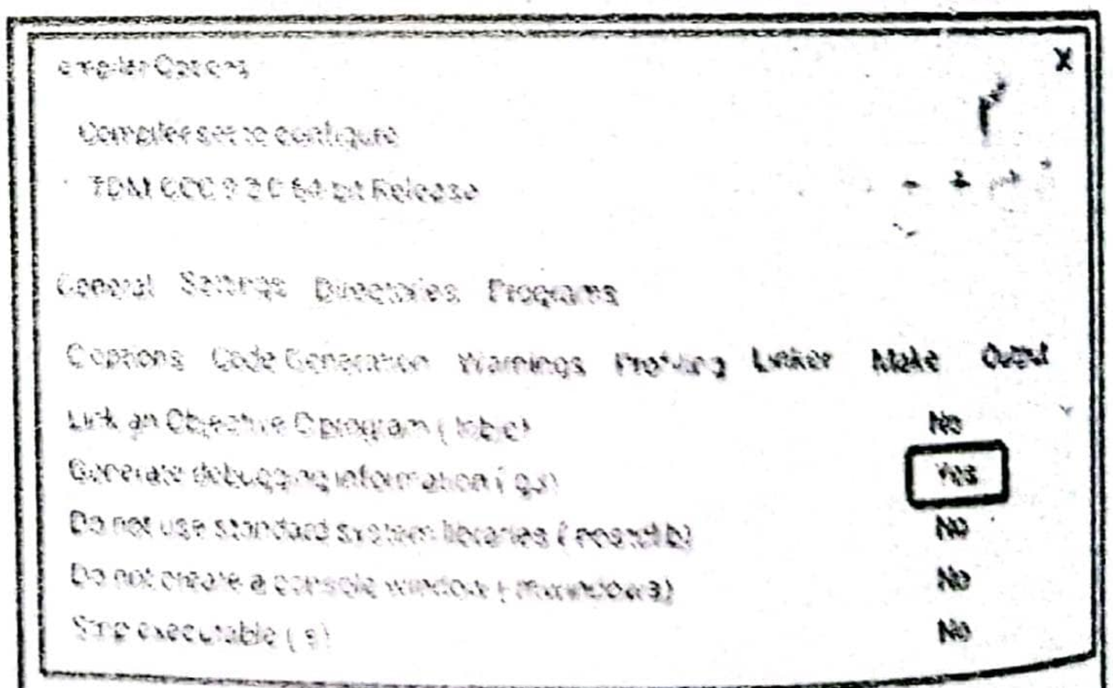


Fig. 5.2. Setting of Compiler Options

Step 2: Create a new Source File

With Dev-C++ IDE, you can create an individual source file or develop a project and *integrate as many source files as you need*. A 'project' can be considered as a container that is used to store all the elements that are required to compile a program. To create a new source file to write a C++ program:

- Click File → New → Source File. (Or just press Ctrl+N)

CamScanner

- Dev-C++ has its own built-in text editor. The files you create with your editor are called *source files*, and for Dev-C++ they typically are named with the extension .cpp. Note that Dev-C++ will not ask for a filename for any new source file until you attempt to:
    - Compile
    - Save the source file
    - Exit Dev-C++

Step 3: Compile and Execute

Once you have entered all of your source code, you are ready to compile.

- Go to the 'Execute' menu and select 'Compile' (or just press F9). After your source code is compiled, the compiler produces an executable program with the .exe extension.
- We can see the compilation status in the 'Compile Log' tab in the workspace.
  If there are any errors whether syntax or linker errors, then they will appear in the 'Compiler' tab. You can double-click on any error to take you to the place in the source code where it occurred.
- Once the Source File is compiled successfully, we need to run it.
- Click on Execute → Run. (Or click F10)
- The Terminal window that gives us the output will be shown as in Fig. 5.3.
- If there are command line parameters to be passed to the program, we click on Execute → Parameters. This will open a dialog using which we can pass parameters.

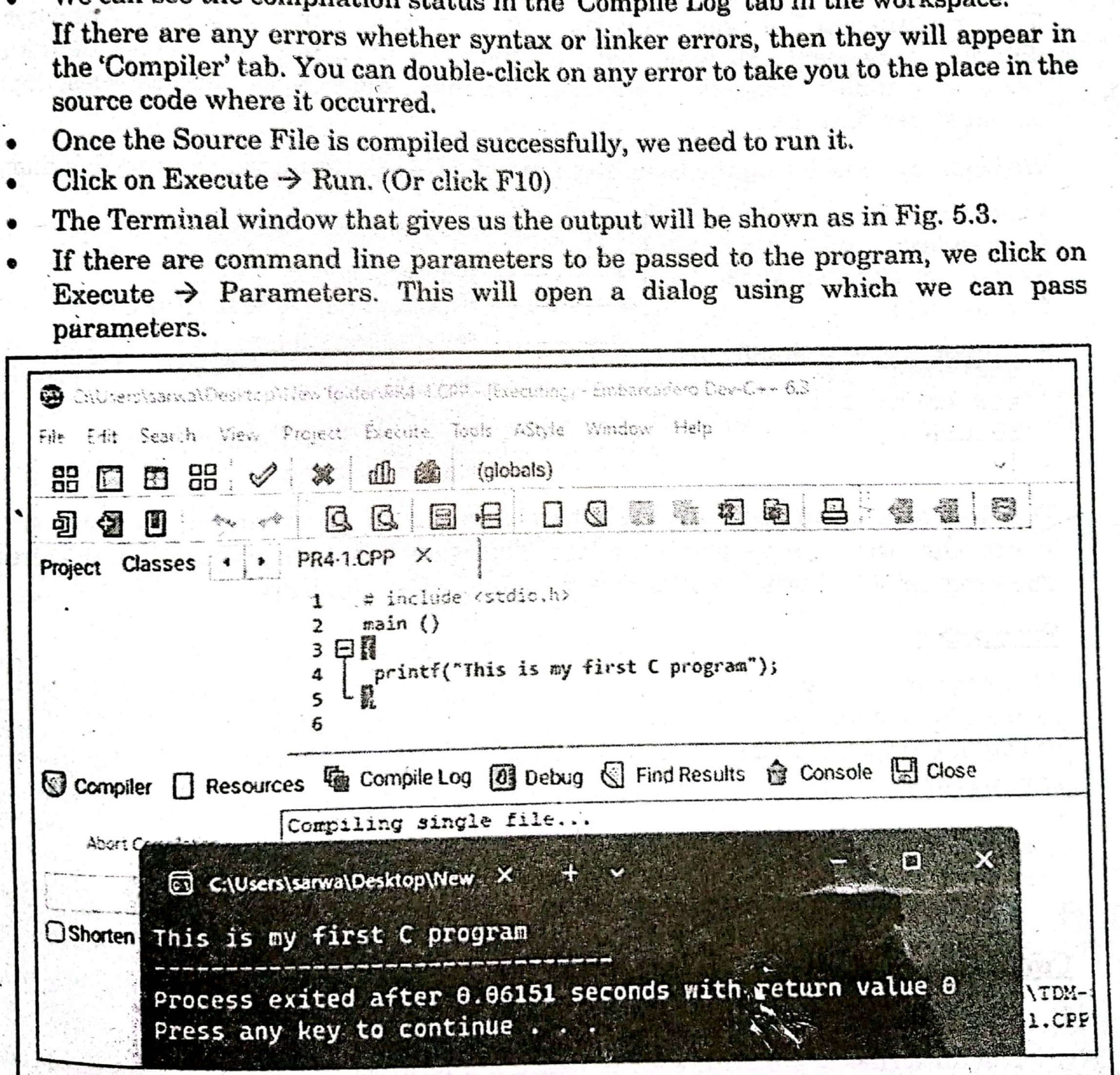


Fig. 5.3. Running a source file

CamScanner

Step 4: Debugging in Dev-C++ IDE

Sometimes we may not get the desired output from our program although the program is syntactically correct. In such a situation, we can debug the program. The Dev-C++ IDE provides the inbuilt debugger.

Follow the below steps to debug the program using Dev-C++ IDE:

- Click Execute → Debug. (Or click F5).
- Once the debug is clicked, we get the debug menu in the IDE.
- Before debugging we can toggle breakpoints using F4 at a particular line of code.
- Using the debug menu, we can use options like add watches, run to cursor, into function, etc. to efficiently debug our program.

The Basic Structure of a C++ Program

The structure of a C++ program is made up of multiple source code files that cater to different components such as main function, member functions, class definition, headers/ standard headers, comments, variables, data types, namespaces, input/ output statements, etc.

We begin by considering the basic structure of a C++ program which may look like:

```
#include<...>                    ┐
#include<...>                    ┘ ←——— Header files
using namespace std;             ┘ ←——— Namespace std
int main()                       ┘ ←——— Function name, Main Function
{                                ┘ ←——— Start of function
   statement 1;                  ┐
   statement 2;...               ┘ ←——— Function body
   return 0,                     ┘ ←——— Main function Return Value
}                                ┘ ←——— End of function
```

The first task of anyone learning programming is to get computer to print on the screen. Our first example prints the text "This is my first C++ program" on the screen. The Program 4.1 shows the structure of complete C++ program;

Program 4.1

```
// program 4-1.cpp
#include <iostream>
using namespace std;
int main()
{
   cout << "This is my first C++ program" << endl;
   return 0;
}
```

Commentary on Program 4.1

1. `#include <iostream>`

   The lines that begin with a # are called *pre-processing directives*. This `#include` directive tells the compiler to use the information in the *header* file called

CamScanner

`iostream`. The file `<iostream>`, which is a standard file that should come with the C++ compiler, is short for *input-output streams*. This command contains predeclared function libraries for displaying and getting an input from the user. The angle brackets around `<iostream>` indicate that this file is to be found in the "usual place", which is system-dependent.

2. `using namespace std;`

`namespace` is a prefix that is applied to all the names in a certain set. The line `using namespace std`, tells the compiler to use standard namespace. `iostream` file defines two *names* used in this program—`cout` and `endl`. This code is saying: *Use the cout and endl tools from the std toolbox.*

Namespaces can be accessed in multiple ways using:

```
using namespace std;
std::cout;
```

3. `int main()`

C++ programs are collections of *functions* designed to perform a specific task and return a computed result. The line `int main()` is the main function where program execution begins. Its return type is `int`. The parentheses following `main` indicate to the compiler that it is a function.

4. `{ }`

The left brace begins the body of each function. A corresponding right brace must end the function.

5. `cout`

The `cout` object is used to display the output to the standard output device. It is defined in the `iostream` header file. The name `cout` is short for *character output*. The data needed to be displayed on the screen is inserted in the standard output stream (`cout`) using the *insertion operator* (<<).

6. `endl`

`endl` statement is used at the end of the line to break the line after displaying the string on the screen. You can also use `\n` (escape sequence character for new line) to break the line. You can use more than one `<< endl` or `\n` statement if you want to break more lines.

7. `return 0`

The `return` value for `int main()` function is used to indicate how the program exited. If the program execution is normal, a 0 return value is used. Abnormal termination (errors, invalid inputs, segmentation faults, etc.) is usually terminated by a non-zero return.

8. `;`

Every statement must end with semicolon (also known as the *statement terminator*). You can put several statements on a line. For clarity's sake usually one statement is put on a line.

9. `//`

If you want to insert comments in your program, for single line comments, use // before mentioning comment, and for multiple line comment, enclose the comment

CamScanner

between /* and */

After you have entered your source code, pull down the *File* menu and click *Save*. Type in the name of your program when the dialog box appears. The default Dev-C++'s "Untitled1" disappears as soon as you start typing. You don't have to type ".cpp" extension. Dev-C++ automatically assigns .cpp extension.

After you enter the filename, say FIRST and click Save, the filename, FIRST.cpp, appears on the tab instead of Untitled1.

Program 4.2

Here is another simple C++ program.

```
// program 4-2.cpp
#include <iostream>
using namespace std;
int main()
{
  cout << "This is";
  cout << " another";
  cout << " C++ program" << endl;
  return 0;
}
```

This program display *This is another C program* on the screen. The key point to this program is that statements are executed sequentially, beginning with the opening brace and ending with the closing brace.

Using Comments

A *comment* is a message to anyone reading the program. Adding comments makes your programs easier to understand. Comments can appear anywhere in a program. They are ignored by the compiler. In fact, they are never added to the executable program.

C++ supports two comment styles: *single line comment* and *multiline comment*. Single line comments are used to define line-by-line descriptions. Double slash (//) is used to represent single line comments. This type of comment stops at the end of a line, so you do not need to type an end-of-comment code. To understand the concept of single line comment, consider this statement.

```
// An example of single line comments
```

Multiline comments are used to define multiple lines descriptions. You begin a comment with the characters /* and end it with */, like this:

```
/* This program displays a word on the monitor */
#include <iostream>
using namespace std;
int main()          // Start of a function
{
  cout << "OK";     // This instruction prints a word
  return 0;
}
```

CamScanner

The /* indicates the start of the comment, and */ indicates the end of the comment. Dev-C++ will ignore everything between the two symbols. Programmers usually place a comment at the start of each program to explain its overall purpose. Comments within a program explain a particular instruction or point of logic, and you can even add a comment after the semicolon of an instruction.

When you include an instruction and comment on the same line, it is conventional to leave some blank space between the two. This simply makes the instruction and comment easier to read.

If you have lengthy comments, type as many lines as you want between the codes:

```
/*This program displays a word OK on the monitor and includes a
* int main() function to make it compatible with Dev-C++ compiler
* that require it.
*/
```

The additional asterisks are only for appearance. They, like everything else between /* and */, are ignored.

Escape Sequences

You can control the way the cursor moves on your monitor, and some other computer functions, by using special codes, called escape sequences. An escape sequence consists of the escape character (the backslash '\') and a second character. The backslash causes an escape from the normal interpretation of a string, so that the next character is recognized as one having a special meaning.

Though there are two characters in an escape sequence, they are considered as a single character. Some of them are shown in the following table.

Table 5.1. Escape sequence Characters

Escape Sequence	Meaning	Result at Execution Time
\n	New line	Move the cursor at the beginning of the next line
\t	Horizontal tab	Move the cursor to the next tab stop
\a	Alert	Produces a beep
\r	Carriage return	Moves the active position to the initial position of the next paragraph
\b	Backspace	Moves the active position to previous position on the current line.
\f	Form feed	Moves the active position to the initial position of the next logical page
\0	End of string	Null
\\	Backslash	Presents with a backslash \
\'	Single quote	Presents with a single quote
\"	Double quote	Presents with a double quote
\?	Question mark	Presents with a question mark

The escape sequence \n performs a new line command and is most commonly used a the end of an instruction. It can appear anywhere inside the double quotes and move the cursor to the start of the next line. These commands

CamScanner

```
cout << "This is\n";
cout << "my first\n";
cout << "C++ program";
```

or this command

```
cout << "This is\nmy first\nC++ program";
```

displays three lines of text:

```
This is
my first
C++ program
```

The \t tab command moves the cursor to the next preset tab stop. To see how the tab code works, use this command

```
cout << "This is\n\tmy first\n\t\tC++ program";
```

The command displays the output

```
This is
        my first
                C++ program
```

The \r code performs a carriage return, moving the cursor to the start of the same line, without moving down to the next line.

Fig. 5.4 shows the difference between a carriage return and a new line command.

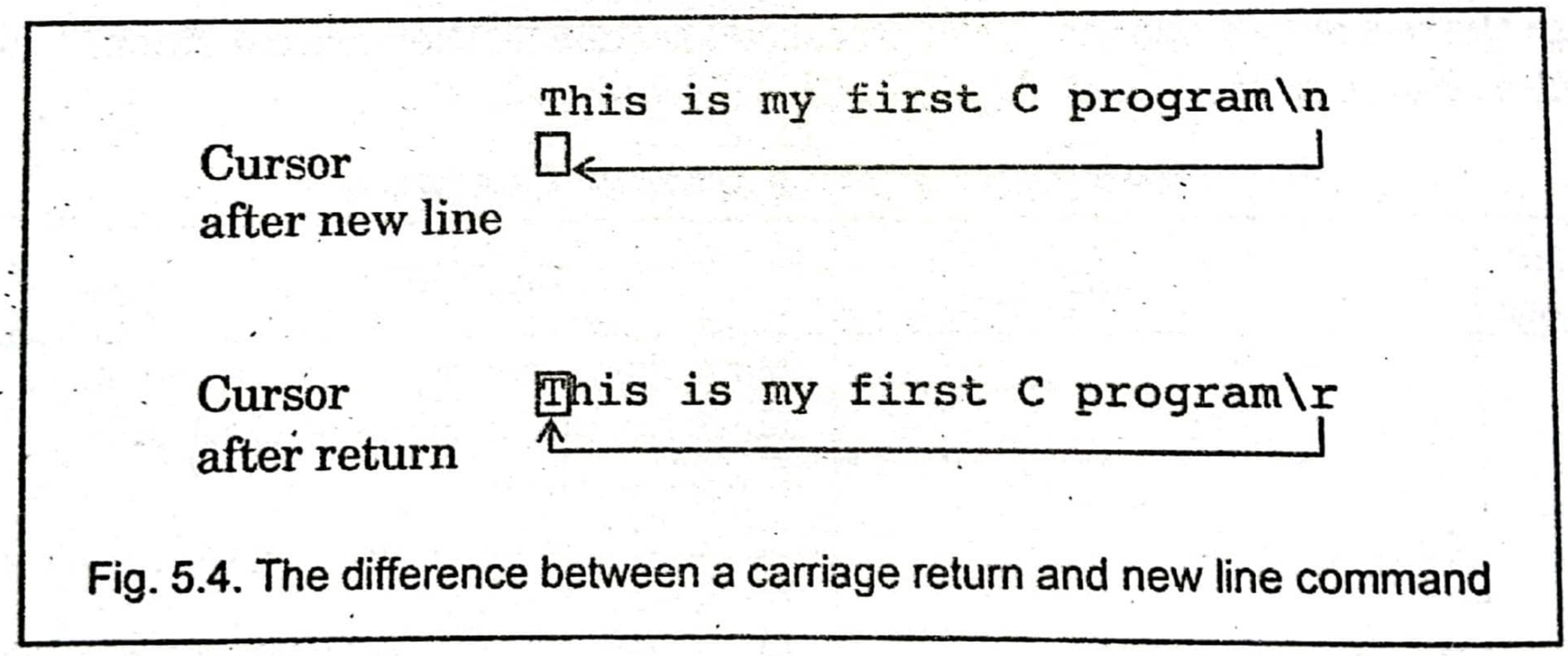


Fig. 5.4. The difference between a carriage return and new line command

If your display something, move the cursor back to the start of the line, and then display something else, the first text will be overtyped. The effect of using the \r code can be seen in this instruction:

```
cout << "Left\rRight";
```

The screen displays only the word Right. Here is why: After displaying the *Left*, the \r code moves the cursor back to the start of the line. The word *Right* is displayed, replacing the characters of the word *Left*.

Unlike the \r code, which backs the cursor to the start of the line, the \b code moves the cursor just one position to the left. It, too, is non-destructive in that it does not erase a character it encounters.

CamScanner

The \0 code represents the *null* character, is used to indicate the end of a string. The compiler automatically places a null character at the end of every string, as the last character within the string. This character is not visible when the string is displayed.

Example 5.1

Name the escape sequences used in the following string constant.

```
"\tTo continue, press the \"ENTER\" key\n";
```

Solution

The escape sequence \t for horizontal tab
\" for double quotation marks, which appears twice
and
\n for new line.

Good Programming Practice

1. The compiler will be able to perform the program either way. But to make program more readable, put `int main()` function on a line by itself and openir and closing braces on their own lines.
2. Every program or function should be preceded by a comment describing tl purpose of the program.
3. Indent the entire body of function one level of indentation within the braces th defines the body of function. This displays the functional structure of prograi and helps make programs easier to read.
4. The last character printed by a function that does any printing should be a n line (\n). This ensures that the function will leave the screen cursor positioned the beginning of a new line.
5. C++ language is case sensitive. Commands and function names are always writ in lowercase characters, so we use `cout` rather than `COUT` or `Cout`. You may get an error message if you use uppercase, but the program just would not look a C++ program. Uppercase is reserved for things called *constants* and *symb names*.

CamScanner

Exercise

5.1. Briefly explain the history of C++ languages.

5.2. What is C++? What is the relationship between C and C++?

5.3. What are the advantages of C++?

5.4. What is the purpose of the statement `#include<iostream>` in a C++ program?

5.5. What is a statement terminator?

5.6. What is the purpose of comments in a C++ program?

5.7. Describe the composition of a function in C++. Explain the `int main()` function.

5.8. What is the general structure of a C++ program?

5.9. Why are some of the statements within a C++ program indented?

5.10. Write a program that displays the following on the screen:

(a) Welcome to Computer Lab.

(b) Board of Intermediate Education, Karachi.

5.11. Write the `cout` instructions for displaying your name, address, and telephone number, centered on the screen.

5.12. Explain what is wrong with the following program:

```
#include <iostream>
using namespace std;
int main()
(
 cout << "I am a College Student";
 return 0;
}
```

5.13. What is wrong with this program?

```
#include <iostream>
int main()
{ // prints "Hello,W orld!":
cout << "Hello,Worl d!\n"
}
```

5.14. What is wrong with the following C-style comment?

```
cout << "Hello,/* change? */ World.\n";
```

5.15. What's wrong with this program:

```
#include <iostream>;
int main
{ // prints "n = 22":
n = 22;
cout << "n = << n << endl;
}
```

5.16. What does a declaration do?

5.17. How do the following two statements differ:

```
char ch = 'A';
char ch = 65;
```

CamScanner

5.18. What are escape sequences?

5.19. What is the difference between \n and \r escape sequence?

5.20. Are C++ programs required to be typed in lowercase? Is uppercase ever used in a C++ program? Explain.

5.21. State some rules of a good programming practice.

Objectives and MCQs

Choose the answer that best completes the statement.

5.1. In 1983, C++ was developed by Bjarne Stroustrup at the
- (a) Bell Laboratories.
- (b) Microsoft
- (c) IBM
- (d) U.S. Defence Department

5.2. Embarcadero Technologies Inc., USA developing its own version of C++ compiler since 2020 and it is called as
- (a) Dev-C++
- (b) R&K C++
- (c) SuperSoft C++
- (d) Turbo C++

5.3. All C++ programs consist of one or more
- (a) functions
- (b) `cout` statements
- (c) `cin` statements
- (d) pointers

5.4. At a minimum, all C++ programs must specify one function called
- (a) basic function
- (b) main function
- (c) primary function
- (d) elementary function

5.5. You begin a comment with the characters
- (a) */
- (b) /*
- (c) *
- (d) *\

CamScanner

Objectives

5.1. C++ is basically a superset of C.
5.2. All C++ programs consist of one or more functions.
5.3. All C++ statements must end with a semicolon.
5.4. The statements within a C++ function must always be enclosed within a pair of {}.
5.5. Every C++ program must contain a `main()` function.
5.6. It is possible to write programs in C++ that are portable to most computers.
5.7. A compiler translates a C++ program into machine language code.
5.8. Comments can appear anywhere in a program.
5.9. Comments are ignored by the compiler.
5.10. C++ provides capabilities to do object-oriented programming.

CamScanner

CHAPTER 6

By: Sir Ahad

C++ FUNDAMENTALS

We refer to input given to the computer as *data*. You feed data into the computer; the computer processes it following your instructions and gives you information as output. But before you input data into your program, you have to tell it what type of data it is dealing with.

In this chapter we will discuss the basic elements used to construct simple C++ programs. These basic elements include the C++ character set, identifier and keywords, data types, constants or literals, variables and arrays and their declarations.

The C++ Character Set

A character denotes any alphabet, digit or special symbol used to represent information. The valid alphabets, numbers and special characters allowed in C++ are listed below:

Letters:	A-Z, a-z
Digits:	0-9
Special symbols:	+ − * / ^ \ ( ) [ ] { } = != < > ' " $; : . , % ! & _ # <= >= @ ~ ¦
White spaces:	blank space, horizontal tab, carriage return, newline, form feed

Above are the list of character set, which are used most of the time, specially the first two which are 'letters' and 'digits.' However, C++ can process any of the 256 ASCII characters as data or as literals. Note that the term 'literals' refers to the character or group of characters (word) that is used to describe the value that is used in the program.

Identifiers and Keywords

Identifiers are names given to various program elements, such as variables, objects, classes, functions, arrays, etc. Following rules must be observed while constructing identifiers:

CamScanner

1. An identifier must start with a letter or (rarely) an underscore (_).
2. An identifier can contain letters, digits, or underscores (_).
3. An identifier cannot contain blanks, commas, or any other special characters.
4. Reserved words (like C++ keywords, such as `int`) cannot be used as names.
5. An identifier must conform to case sensitivity (C++ differentiates between uppercase and lowercase alphabetical characters). For example, the identifier `MAX` is not the same as the identifier `max` or `Max`.

There are several conventions for selecting variable names. Variable names are usually written in lowercase letters, with uppercase names reserved for symbolic constants. The underscore may be used to improve the readability of a variable name. The following names are valid identifiers.

```
x          y1         y2         sum_1      sum_2
TABLE      table      tax_rate   _auto      _temperature
```

The following names are *not* valid identifiers for the reasons stated.

`triangle area`	illegal character (blank space).
`tax-rate`	illegal character (-).
`"x"`	illegal character (").
`est#west`	special character # not allowed.
`5th`	the first character must be a letter.
`–temperature`	do not mistake – for _.

Convention suggests using the underscore as the first letter for variable and function names in the standard library (e.g., `_auto`). The underscore minimizes the chances of a "collision" between function and variable names in your library routines and those in a program.

Good programming style requires the programmer to choose names that are meaningful. If you were to write a program to figure out various taxes, you might have identifiers such as tax_rate, price, and tax, so that the statement

```
tax = price * tax_rate
```

would have an obvious meaning. The underscore is used to create a single identifier from what would normally be a string of words separated by spaces. Meaningfulness, avoiding confusion and readability constitute the main guidelines for a good programming style.

There are certain reserved words, called *keywords*, that have standard, predefined meanings in C++. These keywords can be used only for their intended purpose; they cannot be used as variable names. The standard keyword for C++ are:

alignas	const	for	private	throw
alignof	const_cast	friend	protected	true
and	constexpr	goto	public	try
and_eq	continue	if	register	typedef
asm	decltype	inline	reinterpret_cast	typeid
auto	default	int	return	typename
bitand	delete	long	short	union

CamScanner

bitor	do	mutable	signed	unsigned
bool	double	namespace	sizeof	using
break	dynamic_cast	new	static	virtual
case	else	noexcept	static_assert	void
catch	enum	not	static_cast	volatile
char	explicit	not_eq	struct	wchar_t
char16_t	export	nullptr	switch	while
char32_t	extern	operator	template	xor
class	false	or	this	xor_eq
compl	float	or_eq	thread_local	

Compared to other major languages, C++ has only a small number of keywords. It is a characteristic of C++ that it does a lot with relatively few special symbols and keywords. Although C++ does not permit the use of keywords as variable names, a variable named `auto_type`, for example, is permissible because C++ treats it as one word, and therefore, does not consider it to be a keyword.

C++ Data Types

The data are the values used in the program. Data can be of many types, for example, character, integer, real, string, etc. Anything enclosed in single quotes represents character data in C++. Numbers without fractions represent integer data. Numbers with fractions represent real data, and anything enclosed in double quotes represents a string.

C++, like any other language, provides ways and facilities to handle different types of data by providing data types. C++ data types are of the three types: (1) Fundamental data type (2) Derived data types, (3) User-defined data types. Here, we will discuss only Fundamental data types.

Fundamental data types are those that are not composed of other data types. Five fundamental data types, their description and the memory requirements for each data type is listed in Table 6.1.

Table 6.1. The Fundamental Data Types

Data Type	Description	Typical Memory Requirements
`int`	integer quantity	4 bytes
`char`	single character	1 byte
`float`	floating-point number (*i.e.*, a number containing a decimal point and/or an exponent)	4 bytes
`double`	double-precision floating-point number (*i.e.*, more significant figures, and an exponent which may be larger in magnitude)	8 bytes
`bool`	stores values with two states: true or false	1 byte

Integer, floating-point and double data types represent numbers. They are often referred to collectively as *numeric data type.*

These four basic types can be modified using C++ *type modifiers* to more precisely fit your specific need. The type modifiers are

CamScanner

1. short
2. long
3. singed
4. unsigned

These type modifier precedes the type name. For example, this declares a `long` integer.

```
long int i;
```

There is also a data type called 'void', which is used to specify an empty set of values. The use of such a data type is important when we study *functions* and *pointers*, which is out of the scope of this book.

Now we will discuss each of these four data types in detail in the next subsection.

`int` Data Type

The `int` data type represents *integer* values, meaning any positive or negative number, or zero, without a decimal point. When such values appear in a program, they are called integer. Example of valid integers are

4 −87 +9345 −18239

An integer consists of an optional plus or minus sign followed by a series of digits and cannot contain any other character. Commas, decimal points, or special symbols such as $ or `e` for exponential notation are not allowed.

Declaration of an integer can be preceded by type modifiers short, long, signed or unsigned. So, examples are:

```
short int num_1, num_2;        define short integer variables
long int avg;                  define long integer variable
unsigned int total;            define unsigned integer variable
long unsigned int c;           define unsigned long integer variable
int a, b;                      define integer variables
int a=5;                       initialize integer variable
```

In fact, a `short int` is nothing but ordinary `int`. C++ allows the abbreviation of `short int` to `int` and of `long int` to `long`. So, the first two declarations made above can be written as,

```
int num_1, num_2;              define short integer variables
long avg;                      define long integer variable
```

Also, the `unsigned int` is nothing but a `short unsigned int`. Thus, all the following declarations are same:

```
short unsigned int total;
unsigned int total;
unsigned total;
```

The way there exists a `short unsigned int`, there also exists a `long unsigned int`. Naturally, most C++ programmers prefer this short-cut. The range reserved by the compiler for the above integer type modifiers are as given below.

CamScanner

Data Type	Range
short singed int	−32767 to +32767
short unsigned int	0 to 65535
long singed int	−2147483647 to +2147483647
long unsigned int	0 to 4294967295

Example 6.1

Which of the following are not valid integers?

15 +983.8709 $13,789 78,975 875

Solution

+983.8709	decimal point
$13,789	$ and comma
78,975	comma

`float` and `double` Data Types

The `float` data type represents *floating-point* values, meaning any singed or unsigned number having a decimal point. Examples of valid `float` variables are

2. −78.294 395.20587 +95.0 −439.

A `float` consists of an optional plus or minus sign followed by a series of digits and must include a decimal point. It cannot contain commas or special characters such as the dollar sign but can include the letter `e`.

In addition, the handling of floating-point numbers can be done with the data type *double*. However, it is still considered to be a separate data type due to the fact that it requires twice as much memory as the float type and can store floating-point numbers with a significantly greater range and precision (significant numbers after the decimal point). It is used in situations where the precision offered by type float is insufficient.

`float` and `double` can be written in exponential or scientific notation if needed. The number is written as a value (the mantissa), which is multiplied by 10 (represented by letter `e` (or `E`) which stands for 'exponent') to the stated power (the characteristic). The exponent specifies the number of places the decimal point is to be moved to the left or right. If the exponent is positive, the decimal point is moved to the right; if the exponent is negative, the decimal point is moved to the left. The mantissa always has one non-zero digit before the decimal point. Fig. 6.1 shows the parts of a number represented in exponential notation. The number in Fig. 6.1 would be 2730.0 in

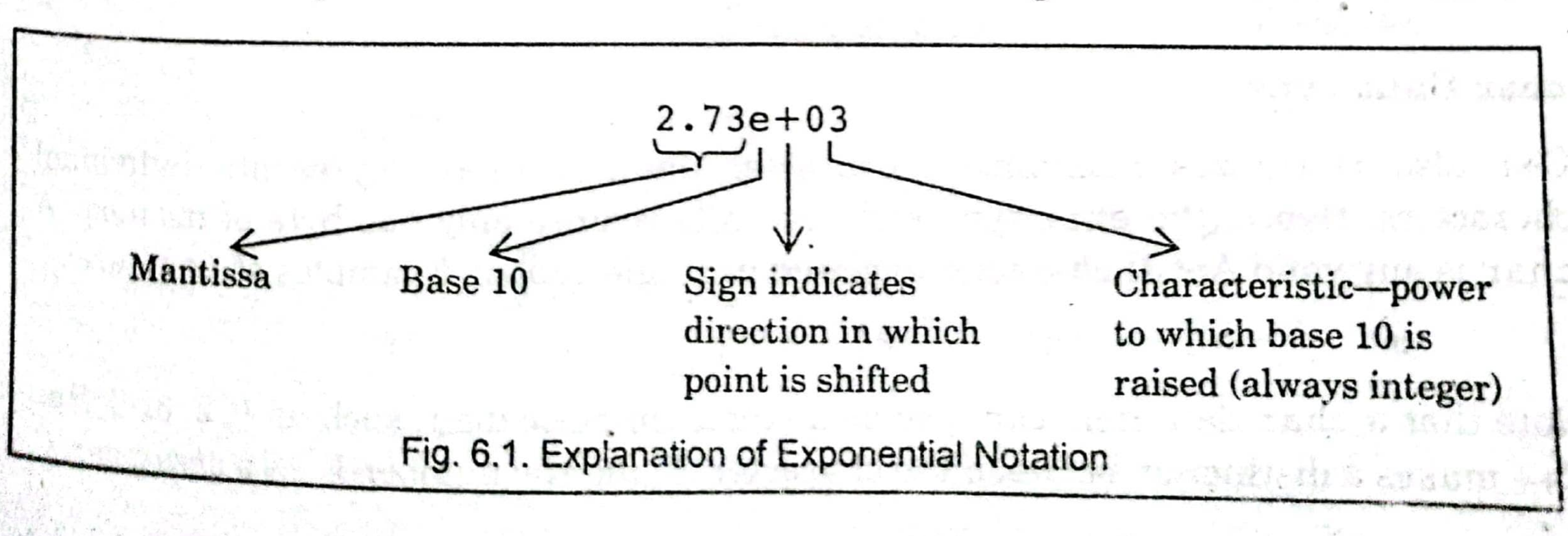


Fig. 6.1. Explanation of Exponential Notation

CamScanner

decimal notation. Below are some numbers in exponential notation along with their equivalents in floating-point notation:

Exponential Notation	*Float Notation*
7.262e+4	72620.0
−2.1613e+2	−216.13
3.14E−3	0.00314
−729.0e−2	−7.29

The `long double` may refer to a separate "extra-large" double-precision data type. It stores an 80-bit floating point number.

Some examples of the declaration of a float are shown below:

```
float flnum;                define floating point number
float fna=93.9;             initializing floating point number
double verybig;             define double precision floating point number
double b=98.74563;          initializing double precision number
long double verybig;        define long double
long double b=9.74563;      initializing long double
```

The range reserved by the compiler for the above float type variables are as given below.

Data Type	Range
float	3.4e−38 to 3.4e+38
double	1.7e−308 to 1.7e+308
long double	3.4e−4932 to 1.1e+4932

Example 6.2

Which of the following are not valid doubles?

15 983.8709 $13,789 −7897I +4.1 487.OO 673.0e03 8475e−3

Solution

15	no decimal point
$13,789	$ and comma and no decimal point
−7897I	no decimal point, letter "I" instead of number "1"
487.OO	"OO" are two characters not the two number "00"
8475e−3	no decimal point

char Data Type

C++ also recognizes nonnumeric character data. A `char` represents individual characters. Hence, the `char` type will generally require only one byte of memory. A `char` is any valid ASCII character enclosed in single quotes. Examples of `char` are:

```
'm'    '>'    'H'    '2'    '['    '!'    'F'
```

Note that a `char` data item can also be a single numeric digit, such as 1, 2, or 3. But C++ makes a distinction between the character '1' and the number 1. As a *character* '1'

CamScanner

cannot be used in a mathematical operation because it does not have a numeric value. As a *number*, 1 can be used in math.

Declaration of a character can be preceded by type modifiers *signed* or *unsigned*. The `singed char` is same as ordinary `char`. Some examples of the declaration of a character are shown below:

```
char ch;                define character variable
unsigned char p;        define unsigned character variable
char c1='a';            initialize character variable
char c2=432;            initialize using decimal value
char c3=0x61;           initialize using hexadecimal value
```

The range reserved by the compiler for the above character type variables are as given below.

Data Type	Range
signed char	−128 to 127
unsigned char	0 to 255

Example 6.3

Which of the following are not valid `char`?

'z' 'zz' "z" '0' z ' '

Solution

```
'zz'        2 characters
"z"         double quotation marks
z           no quotation marks
```

`bool` Data Type

The "bool" type in C++ is used when we need to store the following two values: (1) true, (2) false. Don't use any type of quote while initializing a value (true or false) to a bool type variable. Correct statement is:

```
bool x = true;     // Boolean (true or false)
```

The term "true" refers to "1," whereas "false" refers to "0."

Program 6.1

```
// program 6-1.cpp
#include<iostream>
using namespace std;
int main()
{
  bool x = true, y = false;
  cout << x << endl;
  cout << y;
  return 0;
}
```

CamScanner

The output of the above program for the "bool" type should be:

```
1
0
--------------------------------
Process exited after 0.2644 seconds with return value 0
Press any key to continue . . .
```

Choosing the Data Type

When choosing the data type to represent something in a program, think about the likely data range of the variable. For example, if a variable is to hold student examination marks out of 100, then an `int` type will be more than sufficient. However, if the variable is to hold, for example, the population of Pakistan, then the integer type will be too small, and a `long unsigned int` type will be required.

Unsigned variables are useful if we know in advance that there will never be a negative value for that variable. For example, if we are considering number of students in a class, then the value can never be less than zero, so an unsigned variable may be used.

Any variable which is likely to contain decimal fractions will have to be at least a `float`. If we are considering a variable which is required to handle very large numbers, *i.e.*, for programs dealing with astronomy for example, the `long` data type would be necessary to meet the accuracy requirements.

The `char` data type can hold numbers, but is used to represent characters. A `char` can be assigned either a numeric value or the character itself, surrounded by apostrophes. For example, using ASCII code, the number 65 and the character 'A' are equivalent values for a `char` data type, since 65 is the ASCII code for a capital 'A'.

Example 6.4

What would be the appropriate data types to store the following?

1. Some one's height in meters
1. An exclamation mark (!)
2. The number of students in a university

Solution

The selected data types should be as follows:

1. A *float,* because the height may contain a decimal fraction.
2. A *char,* because it is an ASCII character.
3. An *int,* or more appropriately, an *unsigned int,* because there will never be a negative number of students.

Constants

Constants are quantities whose values do not change during program execution. In C++, however, some authors refer them as *literals* or *literal constants*. In any case, constants can be of any of the basic data types and can be divided into *integer, floating-point, characters, strings* and *Boolean* values. Integer and floating-point constants

CamScanner

represent numbers. They are often referred to collectively as *numeric-type constants.* Let us see the details of each of these constants.

Integer Constants

An *integer constant* is an integer-valued number. Thus, it consists of a sequence of digits.

Following rules must be observed while constructing integer constants:

1. An integer constant must have at least one digit.
2. It must not have a decimal point.
3. It could be either positive or negative.
4. If no sign precedes an integer constant it is assumed to be positive.
5. No commas or blanks are allowed within an integer constant.
6. The value of a constant cannot exceed specified minimum and maximum bonds. The allowable range for integer constants is −32767 to +32767.

Integer constants include decimal, octal and hexadecimal constants. A *decimal* integer constant can consist of any combination of digits from 0 to 9. If the decimal integer constant contains two or more digits, the first digit must not be a 0.

Some examples of valid decimal integer constants are:

```
273      −597      0      3      666
```

On the other hand, the following decimal integer constants are written incorrectly for the reason stated.

`08`	the first digit cannot be a zero
`574-26-28`	illegal character (-)
`−36.2`	contain fractional point
`67 29`	blank space is not allowed
`17,269`	illegal character (,)

An *octal* integer constant can consist of any combination of digits from 0 to 7. However, the first digit must be 0, in order to identify the constants as an octal number.

Some examples of valid octal integer constants are:

```
0      01      −0573      0666
```

On the other hand, the following are not valid octal integer constants.

`05927`	illegal digit (9)
`056.23`	contain fractional point
`264`	does not begin with 0

A *hexadecimal* integer constant can consist of any combination of digits from 0 to 9 and letters a to f (either upper- or lowercase). Hexadecimal integer constants must begin with `0x` or `0X` in order to identify it.

Some examples of valid hexadecimal integer constants are:

```
0x      0x2      −0X27FA      0xabcd
```

CamScanner

On the other hand, the following are not valid hexadecimal integer constants.

`0XDFG`	illegal character (G)
`0x14.9A`	contain fractional point
`9AC5`	does not begin with 0x or 0X

Floating-Point Constants

Integers are inadequate for representing very large or very small numbers and fractions. For this you need floating-point types.

Floating-point constants are also called *real constants*. These are base-10 numbers and could be written in two forms, factional form or exponential form (or both).

Some examples of valid floating-point constants are:

```
0.      1.      −0.5      6070      0.0001      3E−7      3e−7
0.002e+3        .2191129e19
```

On the other hand, the following floating-point constants are written incorrectly for the reason stated.

`3`	no decimal point or exponent
`7,00.0`	illegal character (,)
`2e+5.6`	the exponent value must be an integer
`5e 2`	illegal character (blank space in the exponent)
`2.7e`	the exponent sign must be followed by a number

Character Constants

A *character constant* is a single character, enclosed in apostrophes (*i.e.*, single quotation marks). They include the letters of alphabet (lowercase and uppercase), the numerals 0 to 9, punctuation marks such as :, ; and ?, and special characters such as &, +, −, # and $.

Some examples of valid character constants are:

```
'B'      'm'      '5'      '*'      '?'      ' '
```

Notice that the last constant consists of a blank space, enclosed in apostrophizes.

On the other hand, the following character constants are written incorrectly for the reason stated.

`'3`	missing right quotation mark
`'73'`	more than one character inside quotation marks
`#`	quotation marks missing

Notice that it is possible for the computer to store a number as a character value. However, because the computer stores character values differently than numeric values, the computer can no longer perform arithmetic operations on these numbers.

Virtually all the personal computers make use of the ASCII character set to encode a character. The standard ASCII code uses 7-digit binary numbers; *i.e.*, number consisting of various sequences of 0s and 1s. The code can represent 128 (*i.e.*, $2^7 = 128$) different characters, since there are 128 different possible combinations of seven 0s and 1s. For example, the ASCII code for 'A' is 65. Therefore, after making declaration

CamScanner

```
char ch;
```

you can make either of the flowing assignment statements;

```
ch = 'A';
ch = 65;
```

Note that character constants are enclosed in single quotes. The quotes tell the compiler to get the numeric code (ASCII code value) of the character.

Program 6.2

This Program prints ASCII code value of the inputted character.

```
// program 6-2.cpp
#include <iostream>
using namespace std;
int main()
{
  char key;
  cout << "\nPress any Key: ";
  cin >> key;
  cout << "\nASCII code of " << key << " is " << int(key);
  return 0;
}
```

String Constants

A sequence of zero or more characters enclosed by double quotes is called a *string constant*. This can also be defined as an array of character constant. Some examples of valid character constants are:

"The quick brown fox jumps over the lazy dog."
"514-86-112109"
"12-Qasim Rd."

Below are two incorrect examples:

"The city	(missing right quotation mark)
'A brown cat.'	(must use double quotation marks)

A blank string constant can be represented by " ". In this case, a blank space is placed between two quotes. Also, a character constant (*e.g.*, 'A') and the corresponding single-character string constant ("A") are not equivalent. Also remember that a character constant has an equivalent integer value, whereas a single-character string constant does not have an equivalent integer value and, in fact, consists of two characters—the specified character followed by the escape sequence (\0).

Boolean Constants

There are two Boolean constants and they are part of standard C++ keywords:

- A value of *true* representing true.
- A value of *false* representing false.

You should not consider the value of true equal to 1 and value of false equal to 0.

CamScanner

Symbolic Constants

A *symbolic constant* is a programmer-defined identifier that is replaced with a sequence of characters called *text*. The replacement *text* may represent a numeric constant, a character constant or a string constant. Thus, a symbolic constant allows an identifier to appear in place of a numeric constant, a character constant or a string constant. When a program is compiled, each occurrence of a symbolic constant is replaced by its corresponding replacement text.

Symbolic constants are usually defined at the beginning of a program, often before `main`, although they are also allowed to appear after the opening brace { and `main`. The symbolic constants may then appear later in the program in place of the numeric constants, character constants, etc. that the symbolic constant represent.

Consider a program that makes extensive use of π. We could use the constant 3.14159 throughout, but there are several compelling reasons not to. First, there is a possibility of a typographical error on one of the occurrences of the number. Second, if it later becomes important to change the degree of precision, we have to find all occurrence of 3.14159 and replace them with, say, 3.14159265358979 3. Finally, it is usually more informative to use a name rather than a "mystery" number whose value may have no meaning to the reader and may get confused with other unrelated numbers.

C++ provides two techniques for creating symbolic constants. The first uses `const`. We specify the keyword const, then the data type of the constant, then its symbolic name, and finally an assignment of its given value. Examples using `const` are as follows:

```
const float PI = 3.14159;
const int YEAR_LENGTH = 12;
const char ANSWER = 'Y';
```

For stylistic reasons, we use the convention of writing constants in uppercase. This distinguishes them from variables, which are usually written in lowercase.

The second technique for creating constants is through the `#define` pre-processor

```
#define NUMBER 5.37 <-------------- Numeric values are not in quotes

#define NUMBER 8 <---------------- Int values must not have a decimal point

#define NUMBER 7.0 <-------------- Float values must have at least one digit on the left and right of the decimal point
#define NUMBER 0.4 <-------------- Float values must have at least one digit on the left and right of the decimal point

#define CHARACTER 'A' <----------- Char values are in single quotes

#define STRING "ABC" <------------ String values are in double quotes
```

Fig. 6.2. Syntax for declaring symbolic constants

CamScanner

directive. The `#define` directive has an effect similar to `const`, although the syntax is slightly different. The general syntax for defining symbolic constant is:

```
# define NAME TEXT
```

where NAME represents a symbolic name, typically written in uppercase letters, and TEXT represents the sequence of characters that is associated with symbolic name. Note that the directive does not end with a semicolon or contain = sign. The exact syntax is based on the type of data you are defining—numeric, character, or string (see Fig. 6.2).

You declare a numeric constant by specifying the constant's name and value. For example, this directive:

```
# define PI 3.14
```

creates a constant called PI and gives it the value 3.14 (see Fig. 6.3).

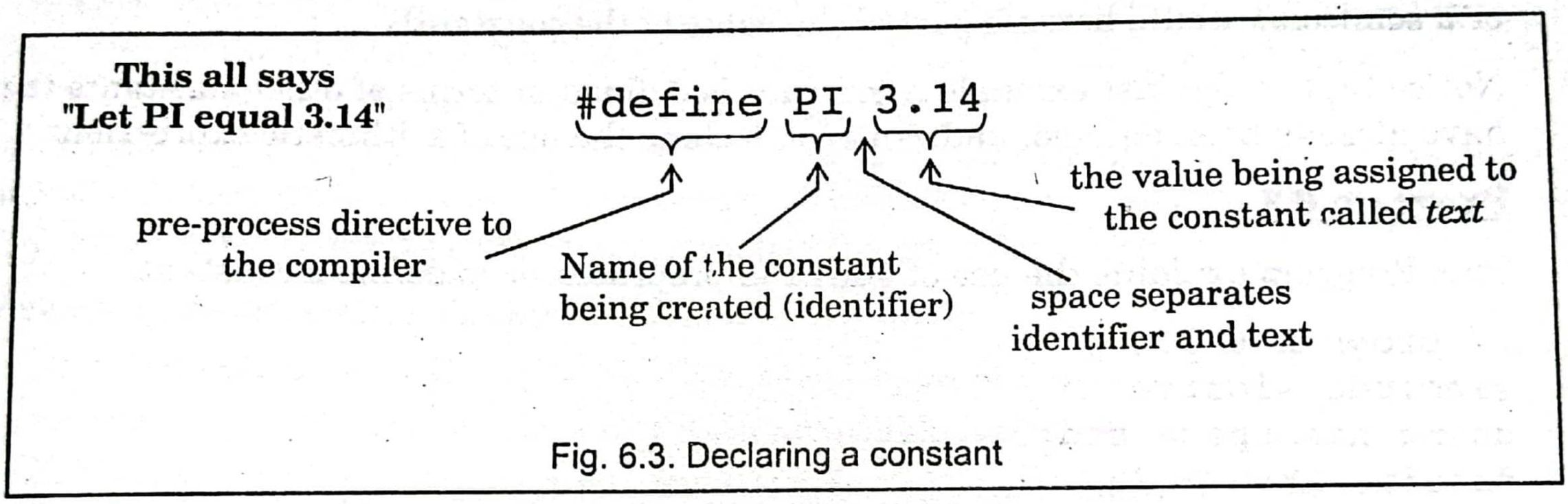


Fig. 6.3. Declaring a constant

Whenever the compiler sees PI in the source code (for example, in a formula), it substitutes the value 3.14 (see Fig. 6.4).

Some other examples of pre-processed defined constants are as follows:

Before the object code is created each occurrence of PI is replace with the value of 3.14

```
#define PI 3.14
int main()
{
   ..... PI.....
   ..... PI.....
   ... PI.....
}
```

This is what is compiled:

```
#define PI 3.14
int main()
{
   ..... 3.14 .....
   ..... 3.14 .....
   ... 3.14 .....
}
```

Fig. 6.4. The compiler substitutes the value for the identifier

CamScanner

```
#define ANSWER              'Y'
#define PI                  3.14159
#define RADIUS              27
#define DAYS_OF_WEEK        "Mo Tu We Th Fr Sa Su"
#define FIRST_PROGRAMMER    "Ada Lovelace"
#define TRUE                1
#define FALSE               0
#define POLLUTION           FALSE
#define CIRCUMFERENCE       2*PI*RADIUS
```

Unlike `const`, which is a true C++ language element, `#define` is a pre-processor directive and is handled differently. Before compilation, the pre-processor removes all references to the symbols created by `#define` and replaces them with their designated values. That is why no equal sign or semicolon is associated with them: An equal sign or a semicolon would become part of the value of the constant!

Notice that in the last example a constant is defined in terms of other constants that have already been defined, and can even include the use of arithmetic expressions.

Program 6.3

This Program explains the use of `#define` preprocessor to define a constant.

```
// program 6-3.cpp
#include <iostream>
using namespace std;
#define LENGTH 20
#define WIDTH  5
#define NEWLINE '\n'
int main()
{
   int area;
   area = LENGTH * WIDTH;
   cout << area;
   cout << NEWLINE;
   return 0;
}
```

When the above code is compiled and executed, it produces output: `100`

Program 6.4

This Program explains the use of `const` prefix to declare constants with a specific type.

```
// program 6-4.cpp
#include <iostream>
using namespace std;
int main()
{
   const int   LENGTH = 20;
   const int   WIDTH  = 5;
   const char NEWLINE = '\n';
   int area;
```

CamScanner

```
    area = LENGTH * WIDTH;
    cout << area;
    cout << NEWLINE;
    return 0;
}
```

When the above code is compiled and executed, it produces output: 100

Variables

A *variable* is a programmer-defined identifier whose value can change during the execution of the program. All variables must have a name and a data type. C++ supports most of the common data types for variables—integer, float (real), and character. The data type should be chosen carefully because it determines how the computer internally represents the associated data. If the situation calls for integer data, such as the number of pages in a book or the number of characters in a word, use the `int` data type. If the situation calls for decimal precision, such as the volume of a sphere, use the `float` data type. For extra precision in floating-point numbers (at the expense of using more memory and increased computation time), use the `double` data type. Finally, for processing character data, use the *char* data type.

Rules for constructing Variable Names

1. The first character in the variable name must be an alphabet.
2. No commas or blanks are allowed within a variable name.
3. No special symbol other than an underscore (as in `tax_rate`) can be used in a variable name.

Declaration of Variables

A *declaration* associates a group of variables with a specific data type. A declaration consists of a data type, followed by one or more variable names, ending with a semicolon.

Example of variable names and their data types as they might be declared at the beginning of the function look like this program statement:

```
int main()
{
  int num;
  float tax_rate;
```

This creates an `int` variables named num, and a `float` variable called `tax_rate` (see Fig. 6.5).

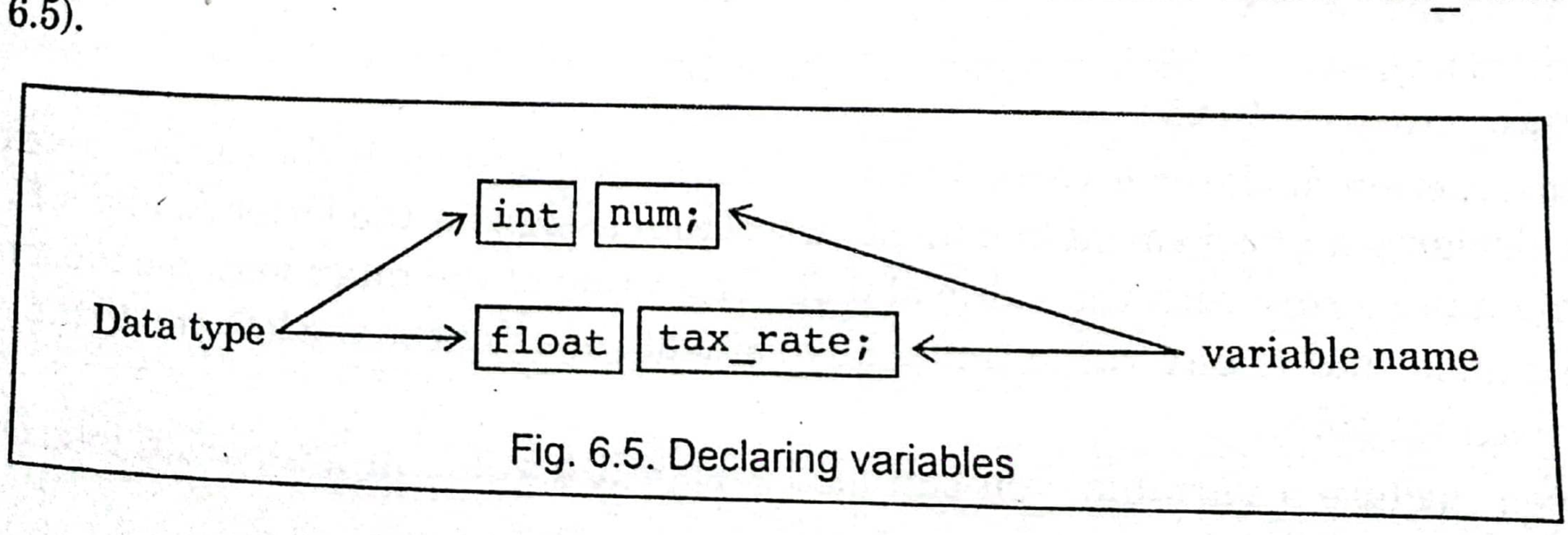


Fig. 6.5. Declaring variables

CamScanner

If you have several variables of the same type, you can declare them in one instruction. Separate each variable name with a comma and end the declaration with a semicolon, as in

```
int main()
{
  int count, students, year;
  float tax_rate, discount;
  … …
  return 0;
}
```

This creates five variables—three `int` values and two `float` values. Declare variables in a group within the `main()` function, after its opening brace and before any other instructions. These declarations could also have been written as follows.

```
int main()
{
  int count;
  int students;
  int year;
  float tax_rate;
  float discount;
  … …
  return 0;
}
```

This form may be useful if each variable is to be accompanied by a comment explaining its purpose. In small programs, however, items of the same type are usually combined in a single declaration.

Some variables have an *initial* value—the value you want the variable to have when you start the program. To do so, the declaration must consist of a data type, followed by a variable name, an equal sign (=) and a constant of the appropriate type. A semicolon must appear at the end, as usual. Unlike a constant, this value can change as the program runs. You assign an initial value either in the declaration or as a separate instruction.

With numeric and `char` data, you can assign the value as part of the declaration:

```
int main()
{
  int count = 5;
  char initial = 'A';
  float rate = 0.55;
```

This instruction declares a variable named *count* and assigns it the initial value of 5. It also declares a `char` variable named *initial* and assigns it the letter A, and a `float` variable named *rate* with the value of 0.55. The value of the `char` variable must be in single quotes, and `float` values must have at least one digit to the left and to the right of the decimal point.

Once you declare a variable, you can also assign it a value in a separate instruction

CamScanner

using the = symbol, like this:

```
count = 5;
initial = 'A';
rate = 0.55;
```

The assigned value is only the initial value. As variables, the initial values can be changed as the program runs.

Arrays

An *array* is an identifier that refers to a collection of data items that all have the same name. The data items must all be of the same type (*e.g.*, all integers, all characters, etc.).

To declare an array, define the variable type, specify the name of the array followed by square brackets and specify the number of elements it should store:

```
string cars[4];
```

Here, `string` is the type of element to be stored in the array, `cars` is the name of the array having size of 4.

We have now declared a variable that holds an array of four strings. To insert values to it, we can use an array literal—place the values in a comma-separated list, inside curly braces:

```
string cars[4] = {"Toyota", "BMW", "Ford", "Mazda"};
```

To create an array of six integers, you could write:

```
int Num[6] = {10, 20, 30, 40, 50, 60};
```

You can access an array element by referring to the index number inside square brackets [ ].

In C++, if an array has a size n, we can store up to n number of elements in the array. However, what will happen if we store less than n number of elements. For example,

```
// store only 3 elements in the array
int Num[6] = {10, 20, 30};
```

Here, the array `Num` has a size of 6. However, we have initialized it with only 3 elements. In such cases, the compiler assigns random values to the remaining places. Oftentimes, this random value is simply 0.

Few things to remember:

1. The array indices start with 0. Meaning `Num[ 0 ]` is the first element stored at index 0.
2. If the size of an array is n, the last element is stored at index $(n-1)$. In the above example, `Num[ 5 ]` is the last element.
3. Elements of an array have consecutive addresses. For example, suppose the starting address of `Num[ 0 ]` is 2120. Then, the address of the next element `Num[ 1 ]` will be 2124, the address of `Num[ 2 ]` will be 2128, and so on.
4. Here, the size of each element is increased by 4. This is because the size of `int` is 4 bytes.

CamScanner

This statement, for example, accesses the value of the first element in cars:

```
string cars[4] = {"Toyota", "BMW", "Ford", "Mazda"};
cout << cars[0];
```

The output of this statement will be: Toyota

If you want to change the value of a specific element, refer to the index number, for example:

```
string cars[4] = {"Toyota", "BMW", "Ford", "Mazda"};
cars[0] = "Volvo";
cout << cars[0];
```

Now the output of above statements will be Volvo instead of Toyota.

CamScanner

Exercise

6.1. List five keywords and explain their use.

6.2. State the rules which must be observed while constructing identifiers.

6.3. Which of the following are not valid identifiers and why?

`2y` `yes` `ono_o_no` `00_go` `star*it` `me_to-2` `xYint` `int`

6.4. What are the keywords in C++? What restrictions apply to their use?

6.5. Which data type is each of the following?

(a) TRUE
(b) 3958
(c) "&"
(d) 9
(e) "9"
(f) "This is my home."

6.6. Are these integers valid or invalid?

(a) 485
(b) 000
(c) 3854&
(d) −935
(e) 96,109

6.7. Are these valid floating-point numbers? If not, change them so they will be valid?

(a) +953.0
(b) −0.3874
(c) 6,209.85
(d) %385.86
(e) 395e+3

6.8. Determine which of the following are valid character constants?

(a) 't'
(b) "U'
(c) '#'
(d) '\n'
(e) '\876'
(f) '\\'

6.9. Determine which of the following are valid string constants?

(a) "1.7E+23"
(b) '4:40 A.M.'
(c) "STUDENT:
(d) "Chap-6 (cont\'d)"

6.10. What is a char data type?

CamScanner

6.11. How is the data type `string` different from the data type `char`?

6.12. How does the character '3' differ from the number 3?

6.13. What data type would you use for a variable storing area of a triangle?

6.14. Why would a program need to use the `long int` data type?

6.15. Why would you use a `double float` type?

6.16. Name and describe the five fundamental data types in C++.

6.17. What is a variable?

6.18. What is the difference between a constant and a variable?

6.19. How do you declare a constant?

6.20. Describe two different ways that floating-point constants can be written. What special rules apply in each case?

6.21. Can a variable have the same value throughout the entire program?

6.22. How do you change the value of a constant?

6.23. Distinguish between the following
   (a) float and double data type
   (b) constant and variable
   (c) integer and character constant

6.24. Decide what types of data you need, and write their declarations, for a program that computes the sum and average of four numbers.

6.25. Write C++ statement to declare a variable to represent the answer to a true/false question. Assume that true has a value of 1 and false has a value of 0.

6.26. Explain what is wrong in the following program segment:

```
char student[3] = "Rizwan"
float tax_rate;
char name (10);
int num [10]
tax_due = "Rs.1,284.00";
```

6.27. Write appropriate declaration for following variables and arrays.
   (a) Integer variables: `a, b`
   (b) Unsigned integer variable: `students`
   (c) Floating-point variables: `x, y, z`
   (d) Character variables: `l, m, n`
   (e) 60-element character array: `message`

6.28. Write appropriate declaration and assign the given initial values to following variables and arrays.
   (a) Integer variables: `a = 485, b = −278`
   (b) Unsigned integer variable: `students = 96854`
   (c) Floating-point variables: `x = 0.4, y = 0.0005, z = 45.7`
   (d) Double-precision variables: `d1 = 1.68×10^-6`

CamScanner

Objectives and MCQs

Choose the answer that best completes the statement.

6.1. Char constant values
- (a) must be enclosed in double quotes
- (b) must be enclosed in single quotes
- (c) can be enclosed in either single or double quotes
- (d) does not need to be enclosed in quotes

6.2. int constant values
- (a) can include an optional comma to improve readability
- (b) can include an optional dollar sign when representing money
- (c) can include an optional plus or minus sign
- (d) can consist only of digits

6.3. Symbolic constants
- (a) must appear in capital letters
- (b) are disliked by serious programmers
- (c) enhance the readability of a program
- (d) can include keywords

6.4. Variables
- (a) are created using the #define directive
- (b) are floating-point by default
- (c) should have simple names
- (d) can be defined as having any C++ data type

6.5. A variable definition differs from a declaration in that
- (a) the declaration sets aside storage space.
- (b) the declaration specifies the name and type of the variable.
- (c) the definition sets aside storage space.
- (d) the definition specifies the name and type of the variable.

Objectives

6.1. On a personal computer an integer is usually represented with four bytes.

6.2. Integer and floating-point data types represent numbers. They are often referred to collectively as numeric data type.

6.3. Constants are quantities whose values do not change during program execution.

6.4. In C++ some authors refer constants as literals or literal constants.

6.5. C++ allows us to work with string-type data through the use of arrays.

6.6. The characters of a string are stored in consecutive memory storage locations.

6.7. Names cannot contain special characters.

6.8. #define is called a pre-processor directive.

CamScanner

CHAPTER 7

OPERATORS AND EXPRESSIONS

By: Sir Ahad

The processing phase of a program converts input *data* into *information* that is output. This process of converting data into information takes many forms. When working with numbers, the process often involves some form of mathematical operation. *Operators* are needed to perform such type of calculations that transform data into information. An operator defines what is done with data. Therefore, we can say that 'operators' are used to perform some 'operation' on the 'operands'. For example:

$$a = b + c$$

Here, *b* and *c* are operands, and the operation is being performed is 'addition' by using '+' operator.

Expressions

An expression is a collection of operands (one or more) and operators (zero or more) that can be evaluated to a single value. An operand is a constant, variable, or another expression. The most general form is

```
operand-1 operation operand-2
```

The result is a new value. For example:

```
2 + 6 evaluates to 8
4 * 5 evaluates to 20
48/8 evaluates to 6
10 - 3 evaluates to 7
18.4 - 4 evaluates to 14.4
```

CamScanner

Note that the use of whitespace (that is, blanks) has no effect on the evaluation of the expression.

```
6 + (7 + 6)(i) / (3*j/-k) - 84.27
```

The use of parentheses also improves the readability and guarantees the order of evolution according to precedence rules.

Classifications of Operators

Operators fall into three classes:

1. *Unary Operator:* Those that requires one operand.
2. *Binary Operator:* Those that requires two operands.
3. *Ternary Operator:* This one is unique. It requires three operands.

C++ is very rich in operators and is sometimes called "the language of operators". It has as many as 45 different operators. Some commonly used operators are:

1. Arithmetic operators
2. Assignment operators
3. Relational operators
4. Logical/Boolean operators
5. Increment and Decrement operators

Arithmetic Operators

Arithmetic operators are used to perform mathematical operations. An arithmetic expression is made up of constants, variable, a combination of both or a function call, connected by arithmetic operators. Following list provide the detail about arithmetic operators.

Operator	Operation	Example
+	addition	a + b
−	subtraction	a − b
*	multiplication	a * b
/	division	a / b
%	remainder after integer division	a % b

The operands acted upon by arithmetic operators must represent numeric values. Thus, the operands can be integer quantities, floating-point quantities or characters (remember that character constants represent integer values, as determined by the computer's character set). The remainder operator (%) requires that both operands be integers and the second operand be nonzero. Similarly, the division operator (/) requires that the second operand be nonzero.

Division of one integer quantity by another is referred to as *integer division*. This operation always results in a truncated quotient (*i.e.*, the decimal portion of the quotient will be dropped). For example:

```
7/2     evaluates to    3
11/5    evaluates to    2
1/3     evaluates to    0
```

CamScanner

On the other hand, if a division operation is carried out with two floating-point numbers, or with one floating-point number and one integer, the result will be a floating-point quotient.

Suppose that `i` and `j` are integer variables whose values are 7 and 2, respectively. Several arithmetic expressions involving these variables are shown below, together with their resulting values.

Expression	*Value*
`i + j`	`= 9`
`i - j`	`= 5`
`i * j`	`= 14`
`i / j`	`= 3`
`i % j`	`= 1`

Notice the truncated quotient resulting from the division operation, since both operands represent integer quantities. Also, notice the integer remainder resulting from the use of the remainder operator in the second expression.

Now suppose that `f` and `g` are floating-point variables whose values are `18.5` and `2.0`, respectively. Several arithmetic expressions involving these variables are shown below, together with their resulting values.

Expression	*Value*
`f + g`	`= 20.5`
`f - g`	`= 16.5`
`f * g`	`= 37.0`
`f / g`	`= 9.25`

Now again suppose that `c` and `d` are character-type variables that represent the characters `F` and `S`, respectively. Several arithmetic expressions involving these variables are shown below, together with their resulting values (based upon the ASCII character set).

Expression	*Value*
`c`	`= 70`
`c + d`	`= 153`
`c + d + 9`	`= 162`
`c + d + '9'`	`= 210`

Note that `F` is encoded as (decimal) `70`, `S` is encoded as `83`, and `9` is encoded as `57` in the ASCII character set.

If one or both operands represent negative values, then the addition, subtraction, multiplication and division operations will result in values whose signs are determined by the usual rules of algebra. Integer division will result in truncation toward zero; *i.e.*, the resultant will always be smaller in magnitude than the true quotient.

Unlike the other arithmetic operators, which accept both integer and floating-point operands, the remainder operator (sometime called the modulus operator) accepts only integer operands. The resulting value is the

CamScanner

the second operands. For example:

```
9 % 5 = 4
```

because 5 goes into 9 once with a remainder of 4. The expression:

```
10 % 5 = 0
```

because 5 goes into 10 two times with no remainder.

Order of Precedence

Order of precedence means the order in which the multiple arithmetic operations take place. The order of operation of arithmetic operators is as follows:

Operator	Operation	Order
*, /, %	multiplication/division/remainder	higher
+, −	addition/subtraction	lower

Another important consideration is the *order* in which consecutive operations within the same precedence group are carried out. This is known as *associativity*. Within each of the precedence groups described above, the associativity is left to right. In other words, consecutive addition and subtraction operations are carried out from left to right, as are consecutive multiplication, division and remainder operations.

Several arithmetic expressions are shown below whose results may have different values if the associativity is changed.

```
8 − 2 + 3   = 9
8*3 / 2     = 12
9/3 + 2 * 4 = 11
```

The natural precedence of operations can be altered through the use of parentheses. Whatever operations occur in parentheses is performed first following the order of precedence given in table above. In the event of nested parentheses (*i.e.* parentheses inside the parentheses), the expression enclosed by the innermost parentheses is performed first. For example, the expression 1 + ((3 + 1)/(8 − 4) − 5) is evaluated as shown in Fig. 7.1:

1 + ((3 + 1)/(8 − 4) − 5)	The innermost parentheses are evaluated first. The expressions (3+1) and (8−4) are at the same depth, so they can be evaluated in either order.
1 + (4/(8 − 4) − 5)	
1 + (4/4 − 5)	
1 + (1 − 5)	Division has a higher precedence than subtraction.
1 + −4	
−3	Final result

Fig. 7.1. Evaluation of an expression enclosed by parentheses

CamScanner

Several arithmetic expressions are shown below where parentheses override the order of precedence.

```
3 * (4 + 2)                    = 18
(6 + 3)/(1 + 2)                = 3
2 * (1 + 6/2) / (3 + 1)        = 2
```

Sometimes it is a good idea to use parentheses to clarify an expression, even though the parentheses may not be required. On the other hand, the use of overlay complex expressions, such as

```
2 * ((5 % i) * (4 + (j - 3)/(k+2)))
```

should be avoided if at all possible. Such expressions are difficult to read, and they are often written incorrectly because of unbalanced parentheses.

Example 7.1

Given the following declarations:

Integer variables `i = 3, j = 4`

Floating-point variables `f = 2.5, g = 1.0`

Evaluate the values for the following expressions:

1. $i + j + f + g$
2. $i + f * j + g$
3. $f / g + i / j$
4. $f - g * i + g / j$
5. $f / 0$

Solution

1. The expression $i + j + f + g$ is equal to 3 + 4 + 2.5 + 1.0 = 10.5
2. The expression $i + f * j + g$ is equivalent to $(i + (f * j) + g)$ because the precedence of * is higher than that of +. Substituting the values of the variables, we get: (3 + (2.5 * 4) + 1.0) = 14.0
3. The expression $f/g + i/j$ is equivalent to $(f/g) + (i/j)$. Thus, substituting the values of f, g, i and j, we get:

   (2.5/1.0) + (3/4) = 2.5

   Note that the value of 3/4 is calculated to be 0. Because the division of two integer values 3/4 does not give any integer quotient. Hence it is taken to be zero.
4. The expression $f - g * i + g/j$ is equivalent to $(f - (g * i)) + (g/j)$. Thus, substituting the values of f, g, i and j, we get:

   (2.5 − (1.0*3)) + (1.0/4) = 2.5 − 3.0 + 0.25 = −0.25
5. $f/0$ is undefined. Any number divided by zero is infinite and hence the value is not defined.

Program 7.1

```
// program 7-1.cpp Arithmetic Operators
#include <iostream>
using namespace std;
```

CamScanner

```
int main()
{
  int a, b;
  a = 11;
  b = 2;

  // printing the sum of a and b
  cout << "a + b = " << (a + b) << endl;

  // printing the difference of a and b
  cout << "a - b = " << (a - b) << endl;

  // printing the product of a and b
  cout << "a * b = " << (a * b) << endl;

  // printing the division of a by b
  cout << "a / b = " << (a / b) << endl;

  // printing the modulo of a by b
  cout << "a % b = " << (a % b) << endl;
  return 0;
}
```

Output

```
a + b = 13
a - b = 9
a * b = 22
a / b = 5
a % b = 1
```

Assignment Operators

The basic assignment operator is =. This is used to assign value of an expression to an identifier. It has the general form

```
identifier = expression
```

where `identifier` generally represents a variable, and `expression` represents a constant, a variable or a more complex expression.

Some examples of expressions that make use of the = operator are:

Expression	*Meaning*
`a = 5`	assign integer value 5 to variable a
`x = y`	assign the value of y to x
`sum = 25.23`	assign floating point value 25.23 to sum
`a = b + c`	assign the sum value of b and c to a
`area = length * breath`	assign the multiplicative value of length and breadth to area

The assignment operator has right-to-left associativity, so it causes the value of the

CamScanner

right-hand operand to be written into the memory location of the left-hand operand. Hence the expression

```
a = b = c = d = 1;
```

is interpreted as

```
(a = (b = (c = (d = 1))));
```

First the value 1 is assigned to d, then d is assigned to c, then c is assigned to b, and finally b is assigned to a. The value of the entire expression is 1. This is a convenient syntax for assigning the same value to more than one variable. Note, however, that each assignment may cause quiet conversions, so

```
int i;
double f;
f = i = 3.5;
```

assigns the truncated value 3 to both f and i. On the other hand,

```
i = f = 3.5;
```

assigns 3.5 to f and 3 to i.

In addition to =, there are a number of assignment operators in C++. Some of them are

Operator	Operation	Example	Explanation
+=	add-assign	a += b	same as a = a + b.
-=	subtract-assign	a -= b	same as a = a - b
*=	multiply-assign	a * =b	same as a = a * b
/=	divide-assign	a/ =b	same as a = a / b
%=	remainder-assign	a % =b	same as a = a % b

For example, the expression

```
i = i * 5;
```

can be written as

```
i *= 5;
```

Now suppose that i and j are integer variables whose values are 5 and 7, and f and g are floating-point variables whose values are 5.5 and -3.25 respectively. Several arithmetic expressions involving these variables are shown below, together with their resulting values. Each expression utilizes the original values of i, j, f and g.

Expression	*Equivalent Expression*	*Final Value*
`i += 5`	`i = i + 5`	`= 10`
`f -= g`	`f = f - g`	`= 8.75`
`j *= (i - 3)`	`j = j * (i - 3)`	`= 14`
`f /= 3`	`f = f / 3`	`= 1.833333`
`i %= (j - 2)`	`i = i % (j - 2)`	`= 0`

The assign operators have relatively low precedence. This leads to interesting consequences. For example, the following two expression are not the same:

```
i = i * 3 + 4;
i *= 3 + 4;
```

CamScanner

The addition operator has higher precedence than the assign operator, and the multiplication operator has higher precedence than the addition operator, so the two expressions are interpreted as follows:

Expression 1

```
i = i * 3 + 4;
or i = ((i * 3) + 4);
```

Expression 2

```
i *= 3 + 4;
or i *= (3 + 4);
or i = (i * (3 + 4));
```

Example 7.2

Given the following declarations:

Integer variables `i = 3, j = 4`

Floating-point variables `f = 2.5, g = 1.0`

Evaluate the values for the following expressions:

1. $i \mathrel{+}= j + f - g$
2. $i \mathrel{/}= f * j + g$
3. $j \mathrel{\%}= g + i$
4. $f \mathrel{+}= g \mathrel{-}= i$

Solution

1. The expression $i \mathrel{+}= j + f - g$ is equivalent to $i = i+((j + f) - g)$. Substituting the values of the variables, we get the result to be 8.
2. The expression $i \mathrel{/}= f * j + g$ is equivalent to $i = i/((f * j) + g)$. Substituting the values of the variables we get the result to be 0.
3. The expression $j \mathrel{\%}= g + i$ is equivalent to $j = j \% (g + i)$. Substituting the values of the variables, we get the result to be 0.
4. The expression $f \mathrel{+}= g \mathrel{-}= i$ is equivalent to $f = f + (g = (g - i))$. Substituting the values of the variables, we get the result to be 0.5.

Program 7.2

```
// program 7-2.cpp Assignment Operators
#include <iostream>
using namespace std;
int main()
{
  int a, b;

  // 3 is assigned to a
  a = 3;

  // 11 is assigned to b
  b = 11;

  cout << "a = " << a << endl;
  cout << "b = " << b << endl;
  cout << "\nAfter a += b;" << endl;
```

CamScanner

```
    // assigning the sum of a and b to a
    a += b;     // a = a + b
    cout << "a = " << a << endl;
    return 0;
}
```

Output

```
a = 3
b = 11

After a += b;
a = 14
```

Relational Operators

Relational operators are used to compare two numeric operands. The operands can be variables, constants or expressions that ultimately get evaluated to a numerical value. Since characters can be represented as integers using ASCII code, therefore, operands to be compared can be characters as well. All C++ relational operators are binary operators and hence require *two* operands. The relational operators are summarized as:

Operator	Operation	Example
<	less than	a < b
>	greater than	a > b
<=	less than or equal to	a <= b
>=	greater than or equal to	a >= b
==	equal to	a == b
!=	not equal to	a != b

A relational expression is made up of two arithmetic expressions connected by a relational operator. It returns 0 when the relation is *false* and 1 when it is *true*.

These relational operators all fall within the same precedence group, which is lower than the arithmetic operators. For example, the expression:

```
a + b * c < d / f
```

is evaluated as if it had been written as follows:

```
(a + (b * c)) < (d / f)
```

The associativity of these operators is left to right.

Suppose the `i`, `j` and `k` are integer variables whose values are `1`, `2` and 3, respectively. Several expressions using relational operators involving these variables are shown below:

Expression	*Interpretation*	*Value*
`i < j`	true	1
`(i + j) >= k`	true	1
`(j + k) > (i + 5)`	false	0
`k != 3`	false	0
`j == 2`	true	1

Now suppose that `i` is an integer variable whose value is `7`, `f` is a floating-point variable whose value is `5.5`, and `c` is a character variable that represents the character

CamScanner

'w'. Several logical expressions that make use of these variables are shown below. Each expression involves two different types of operands. (Assume that the ASCII character set applies.)

Expression	*Interpretation*	*Value*
`f > 5`	`true`	`1`
`(i + f) <= 10`	`false`	`0`
`c == 119`	`true`	`1`
`c != 'p'`	`true`	`1`
`c >= 10 * (i + f)`	`false`	`0`

Example 7.3

Given the following declarations:

Integer variables `i = 0, j = 1, k = -1`

Floating-point variables `f = 2.5, g = 0.0`

Evaluate the values for the following expressions:

1. $i > j$
2. $j/k < f$
3. $i <= j >= k$
4. $i <= f == j$
5. $-f + i == g > k >= j$
6. $f += (g >= k)$
7. $i + 1 == j != g * 2$

Solution

1. $i > j$ is not true because the value of $i = 0$ and value of $j = 1$. Hence the result is 0 (false).
2. The expression $j/k < f$ is equivalent to $(j/k) < f$ because the precedence of / is higher than that of <. Substituting the values of the variables, we get the result to be 1 (true).
3. The expression $i <= j >= k$ is equivalent to $(i <= j) >= k$. Substituting the values of the variables we get the result to be 1 (true).
4. The expression $i <= f == j$ is equivalent to $(i <= f) == j$. Substituting the values of the variables, we get the result to be 1 (true).
5. The expression $-f + i == g > k >= j$ is equivalent to $((-f) + i) == ((g > k) >= j)$. Substituting the values of the variables we get the result to be 0 (false).
6. The expression $f += (g >= k)$ is equivalent to $f = (f + (g >= k))$. Substituting the values of the variables we get the result $2.5 = 3.5$. Thus, the value of the expression is 0 (false).
7. The expression $i + 1 == j != g * 2$ is equivalent to $((i + 1) == j) != (g * 2)$. Substituting the values of the variables we get the result to be true (1). The term $(i + 1 == j)$ is evaluated to be 1 because the value of $i + 1$ is 1, and the value of $j = 1$. Therefore, $(i + 1 == j)$ is true or the value of the term is 1. The value of the term $(g * 2)$ is $0 * 2$ is zero. Hence the resultant expression amounts to $(1 != 0)$ and this is true. Thus, the value of the expression is 1.

CamScanner

Program 7.3

```
// program 7-3.cpp Relational Operators
#include <iostream>
using namespace std;
int main()
{
  int a, b;
  a = 3;
  b = 5;
  bool result;

  result = (a == b);      // false
  cout << "3 == 5 is " << result << endl;

  result = (a != b);      // true
  cout << "3 != 5 is " << result << endl;

  result = a > b;         // false
  cout << "3 > 5 is " << result << endl;

  result = a < b;         // true
  cout << "3 < 5 is " << result << endl;

  result = a >= b;                    // false
  cout << "3 >= 5 is " << result << endl;

    result = a <= b;                  // true
    cout << "3 <= 5 is " << result << endl;
    return 0;
}
```

Output

```
3 == 5 is 0
3 != 5 is 1
3 > 5 is 0
3 < 5 is 1
3 >= 5 is 0
3 <= 5 is 1
```

Confusing Equality (==) and Assignment (=) Operators

There is one very common type of error that C++ programmers make. The error is the swapping of operators == (equality) and = (assignment). The more damaging thing is that with this error program may run correctly but probably generate incorrect result.

There are two aspects of this problem. One is that any expression in C++ that produces a value can be used for decision-making. If the value is 0, it is treated as false, and if the value is nonzero, it is treated as true. The second is that assignments in C++ produce a value, namely the value that is assigned to the variable on the left side of

CamScanner

the assignment operator. For example, suppose we intend to write

```
if (pay_code == 4)
        cout << "You get a bonus!";
```

but we accidentally write

```
if (pay_code = 4)
        cout << "You get a bonus!";
```

The first `if` statement properly awards a bonus to the person whose pay code is equal to 4. The second `if` statement—the one with the error—evaluates the assignment expression in the `if` condition. This expression is a simple assignment whose value is the constant 4. Because any nonzero value is interpreted as "true", the condition in this `if` statement is always true, and the person always receives a bonus regardless of what the actual pay code is!

Logical/Boolean Operators

A logical operator also called Boolean operator to honor the Irish mathematician George Boole (1815-1864) combines the results of one or more expressions and the resultant expression is called the *logical expression*. After testing the conditions, they return either true (non-zero value) or false (0) as net result.

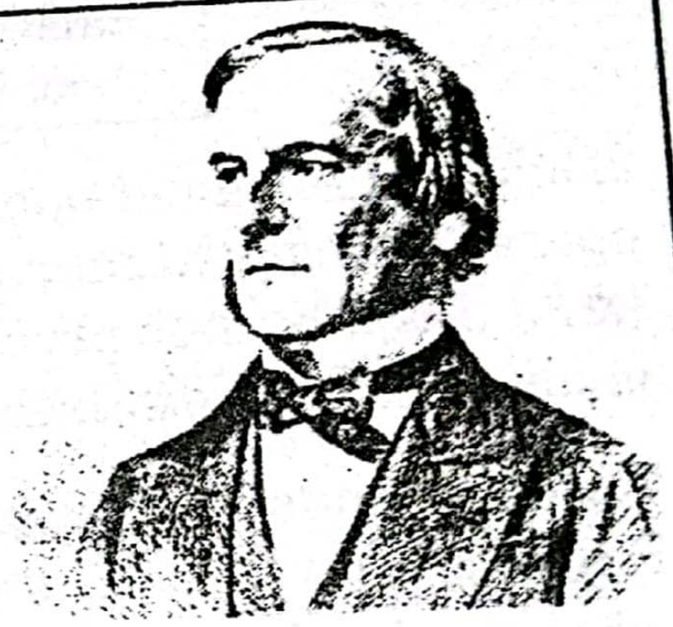


Fig. 7.1 George Boole

The logical operators are unary or binary operators. The operands may be constants, variables or expressions. The basic logical operators are:

Operator	Operation	Example
&&	Logical AND	1 if a = b && c = d; else 0
||	Logical OR	1 if a = b || c > d; else 0
!	Logical NOT	1 if !a; else 0

Logical AND operator (&&) takes two operands. The result of operation will be *true* (nonzero) only if both operands are *true* (nonzero) otherwise *false* (zero). The table 7.1 shows all the four possible combinations of *false* (zero) and *true* (nonzero) values of operands. Such tables are often called *truth tables*.

Table 7.1. Truth Table for logical AND (&&) operator

First Operand	Operator	Second Operand	Result
false (0)	&&	false (0)	0
true (nonzero)	&&	false (0)	0
false (0)	&&	true (nonzero)	0
true (nonzero)	&&	true (nonzero)	1(nonzero)

Logical OR operator (||) also takes two operands. The result of operation will be *tru* (nonzero) if either operand is *true* or if both operands are *true* (nonzero) otherwise *fals* (zero). In other words, the result of a logical OR operation will be *false* only if bot operands are *false* (zero). The table 7.2 shows all the four possible combinations of *fal* (zero) and *true* (nonzero) values of operands.

CamScanner

Table 7.2. Truth Table for logical OR (||) operator

First Operand	Operator	Second Operand	Result
false (0)	||	false (0)	0
true (nonzero)	||	false (0)	1(nonzero)
false (0)	||	true (nonzero)	1(nonzero)
true (nonzero)	||	true (nonzero)	1(nonzero)

Logical NOT operator (!) takes only one operand. If the operand is *true* (nonzero), the result is *false* (zero); if the operand is *false*, the result is *true*. The table 7.3 shows these two possibilities.

Table 7.3. Truth Table for logical NOT (!) operator

Operand	Operator	Result
false (0)	!	1(nonzero)
true (nonzero)	!	0

Each of the logical operators falls into its own precedence group. The logical NOT (!) operator has a higher precedence than the other two logical operators. Logical AND (&&) has a higher precedence than logical OR (||). The precedence of logical operators is lower than the relational and arithmetic operators. The associativity is left to right.

The operands to the logical operators may be integers or floating-point variables. Suppose that `i` is an integer variable whose value is `7`, `f` is a floating-point variable whose value is `5.5`, and `c` is a character variable that represents the character `'w'`. Several complex logical expressions that make use of these variables are shown below.

Expression	*Interpretation*	*Value*
`(i >= 6) && (c == 'w')`	`true`	`1`
`(i >=6) || (c == 119)`	`true`	`1`
`(f < 11) && (i > 100)`	`false`	`0`
`(c != 'p') || ((i + f) <= 10)`	`true`	`1`

The first expression is true because both operands are true. In the second expression, both operands are again true; hence the overall expression is true. The third expression is false because the second operand is false. And finally, the fourth expression is true because the first operand is true.

Now suppose that `i` is an integer variable whose value is `7`, and `f` is a floating-point variable whose value is `5.5`. Several logical expressions which make use of these variables and the logical NOT (!) operator are shown below:

Expression	*Interpretation*	*Value*
`f > 5`	`true`	`1`
`!(f > 5)`	`false`	`0`
`i <= 3`	`false`	`0`
`!(i <= 3)`	`true`	`1`
`i > (f +1)`	`true`	`1`
`!(i > (f +1))`	`false`	`0`

CamScanner

Example 7.4

Given the following declarations:

Integer variables `i = 0, j = 1, k = -1`

Floating-point variables `f = 2.5, g = 0.0`

Evaluate the values for the following expressions:

1. $i \&\& j$
2. $i < j \&\& k < j$
3. $j + k = 1 \mid\mid !i$
4. $f*5 \&\& 5 \mid\mid j/k$
5. $i <= 10 \&\& f >= 1 \&\& j$
6. $!f \mid\mid !k \mid\mid j + k$
7. $f*g < i + j \mid\mid k$

Solution

1. $i \&\& j$ is not true because the value of $i = 0$ and the value of $j = 1$. Hence the result is 0 (false).
2. The expression $i < j \&\& k < j$ is equivalent to $(i < j) \&\& (k < j)$ because the precedence of relational operators is higher than that of logical operators. Substituting the values of the variables we get the result to be 1 (true).
3. The expression $j + k \mid\mid !i$ is equivalent to $(j+k) \mid\mid (!i)$. Substituting the values of the variables we get the result to be 1 (true).
4. The expression $f*5 \&\& 5 \mid\mid j/k$ is equivalent to $((f*5) \&\& 5) \mid\mid (j/k)$. Substituting the values of the variables, we get the result to be 1 (true).
5. The expression $i <= 10 \&\& f >= 1 \&\& j$ is equivalent to $((i<=10) \&\& (f >= 1)) \&\& j$. Substituting the values of variables, we get the result to be 1 (true).
6. The expression $!f \mid\mid !k \mid\mid j + k$ is equivalent to $((!f) \mid\mid (!k)) \mid\mid (j + k)$. Substituting the values of the variables, we get the result to be 0 (false).
7. The expression $f*g < i + j \mid\mid k$ is equivalent to $((f*g) < (i + j)) \mid\mid k$. Substituting the values of variables, we get the result to be 1 (true).

Program 7.4

```
// program 7-4.cpp Logical Operators
#include <iostream>
using namespace std;
int main()
{
 bool result;

 result = (3 != 5) && (3 < 5);                    // true
 cout << "(3 != 5) && (3 < 5) is " << result << endl;

 result = (3 == 5) && (3 < 5);                    // false
 cout << "(3 == 5) && (3 < 5) is " << result << endl;
```

CamScanner

```
result = (3 == 5) && (3 > 5);                    // false
cout << "(3 == 5) && (3 > 5) is " << result << endl;

result = (3 != 5) || (3 < 5);                    // true
cout << "(3 != 5) || (3 < 5) is " << result << endl;

result = (3 != 5) || (3 > 5);                    // true
cout << "(3 != 5) || (3 > 5) is " << result << endl;

result = (3 == 5) || (3 > 5);                    // false
cout << "(3 == 5) || (3 > 5) is " << result << endl;

result = !(5 == 2);                              // true
cout << "!(5 == 2) is " << result << endl;

result = !(5 == 5);                              // false
cout << "!(5 == 5) is " << result << endl;

return 0;
}
```

Output

```
(3 != 5) && (3 < 5) is 1
(3 == 5) && (3 < 5) is 0
(3 == 5) && (3 > 5) is 0
(3 != 5) || (3 < 5) is 1
(3 != 5) || (3 > 5) is 1
(3 == 5) || (3 > 5) is 0
!(5 == 2) is 1
!(5 == 5) is 0
```

Increment and Decrement Operators

The increment operator ++ and decrement operator -- are very useful operators and are not found in some other languages. If a variable a is incremented by 1, the increment operator ++ can be used rather than the expression `a = a + 1` or `a += 1`. The increment operator causes its operand to be increased by 1, whereas the decrement operator causes its operand to be decreased by 1.

These are 'unary' operators since they operate on only one operand. The operand has to be a variable and not a constant. Thus, the expression a++ is valid whereas 6++ is invalid.

The increment and decrement operators can be of two types. These are:

1. First increment or decrement the value of the variable and then take this new value for processing (*prefix*).
2. First take the value of the variable for processing and then only increment or decrement the variable (*postfix*).

The increment and decrement operators (postfix and prefix) are summarized as:

CamScanner

Operator	Example	Operation	Explanation
++	++a	prefix increment	Increment a by 1 then use the new value of a in the expression in which a resides.
++	a++	postfix increment	Use the current value of a in the expression in which a resides, then increment a by 1.
--	--a	prefix decrement	Decrement a by 1 then use the new value of a in the expression in which a resides.
--	a--	postfix decrement	Use the current value of a in the expression in which a resides, then decrement a by 1.

To illustrate these operations, suppose that `i` and `j` are integer variables whose values are 3 and 5 respectively. Several expressions that make use of these variables are shown below.

```
int i, j;          /* declare */
i = 3, j = 5;      /* initialize */

j = i++;           /* j now = 3 and i = 4 postfix increment */
j = ++i;           /* j now = 5 and i = 5 prefix increment */
j = i--;           /* j now = 5 and i = 4 postfix decrement */
j = --i;           /* j now = 3 and i = 3 prefix decrement */
```

If you just want to increment or decrement without any assignment, the postfix and prefix methods are effectively equivalent.

```
i++;          /* set i to i+1 */
++i;          /* set i to i+1 */
i--;          /* set i to i-1 */
--i;          /* set i to i-1 */
```

Example 7.5

Subtract 1 from a variable x in four different ways.

Solution

```
assignment:                 x = x - 1;
compound assignment:        x -= 1;
decrement (postfix):        x--;
decrement (prefix)          --x;
```

Example 7.6

Given the following declarations:

Integer variables	`i = 0, j = 1, k = -1`
Floating-point variables	`f = 2.5, g = 0.0`

Evaluate the values for the following expressions:

1. $++i == j\ !=\ g*2$
2. $(f > g) + !i \mid\mid k++$
3. $(i \mid\mid j) + (f \mid\mid ++k)$

CamScanner

Solution

1. The expression `++i == j != g*2` is equivalent to `((++i) == j) != (g*2)`. Substituting the values of the variables we get the result to be true (1). The term `(++i == j)` is evaluated to be 1 because the value of `++i` is 1, and the value of `j = 1`. Therefore, `(++i == j)` is true or the value of the term is 1. The value of the term `(g*2)` is zero because `0*2` is zero. Hence the resultant expression amounts to `(1 != 0)` and this is true.
2. The expression `(f > g) + !i || k++` is equivalent to `((f > g) + (!i)) || (k++)`. Substituting the values of the variables, we get the result to be true (1).
3. The expression `(i || j) + (f || ++k)` is equivalent to `(i || j) + (f || (++k))`. Substituting the values of the variables, we get the result to be 2. Note here that `++k` represents the prefix increment value of `k`.

Program 7.5

```
// program 7-5.cpp Increment and Decrement Operators
#include <iostream>
using namespace std;
int main()
{
  int a = 10, b = 100, result_a, result_b;
  // incrementing a by 1 and storing the result in result_a
  result_a = ++a;
  cout << "result_a = " << result_a << endl;

  // decrementing b by 1 and storing the result in result_b
  result_b = --b;
  cout << "result_b = " << result_b << endl;
  return 0;
}
```

Output

```
result_a = 11
result_b = 99
```

Conversions of Data Type

Suppose that an arithmetic expression such as `i + j` has both a value and a data type. For example, if both `i` and `j` have type `int`, then the expression `i + j` also has type `int`. But if `i` has type `int` and `j` has type `float`, the `int` would be automatically converted to a `float` before carrying out the operation. Some other conversion rules are shown in following tables:

First Operand	Second Operand	Result
char	char	char
char	int	int
char	long int	long int
char	float	float
char	double	double

First Operand	Second Operand	Result
int	char	int
int	int	int
int	long int	long int
int	float	float
int	double	double

CamScanner

First Operand	Second Operand	Result
long int	char	long int
long int	int	long int
long int	long int	long int
long int	float	float
long int	double	double

First Operand	Second Operand	Result
float	char	float
float	int	float
float	long int	float
float	float	float
float	double	double

First Operand	Second Operand	Result
double	char	double
double	int	double
double	long int	double
double	float	double
double	double	double

To illustrate the idea of automatic conversion, we take the following examples:

Operation	Result	Operation	Result
5/2	2	2/5	0
5.0/2	2.5	2.0/5	0.4
5/2.0	2.5	2/5.0	0.4
5.0/2.0	2.5	2.0/5.0	0.4

Casts

Sometimes we are required to convert the value of an expression to a particular data type. To do so, the expression must be preceded by the name of the desired data type, enclosed in parentheses *i.e.*

```
(data type) expression
```

This type of construction is known as *cast*. This would be clear from the following segment of program:

```
float f;
int i = 6, j = 4;
f = i/j;
```

The output of this segment of program is `f = 1.000000` and not 1.5. This is because, 6 and 4 are both integers and hence 6/4 yields an integer, 1. This 1 when stored in `f` is converted to 1.000000. But what if we do not want the quotient to be truncated. One solution is to make either `i` and `j` as `float`. Let us say that other requirements of the program does not permit us to do this. In such a case what do we do? Use cast. The following segment of program illustrate this:

```
float f;
int i = 6, j = 4;
f = (float) i/j;
```

and we will get the output `f = 1.500000`. In this program `(float)` causes the variable to be converted from type `int` to type `float` before being used in the division operation

CamScanner

Some other examples of casts are:

```
(long)('A' + 1.0)
i = (float) ((int) j + 1)
(double) (x = 45)
```

but not

```
(double) x = 45    // equivalent to ((double) x) = 77, error
```

The cast operator is a unary operator having the same precedence and right-to-left associativity as other unary operators. Thus, the expression

`(float) i + 3` is equivalent to `((float) i) + 3`

because the cast operator has higher precedence than +.

Table 7.4. gives the summary of the precedence, for highest to lowest, for various operators including their associativity.

Table 7.4. Precedence of Various Operators

Operators	Type	Associativity
++ -- ! casts	unary	right to left
* / %	multiplicative	left to right
+ -	additive	left to right
< > <= >=	relational	left to right
== !=	equality	left to right
&&	logical AND	left to right
||	logical OR	left to right
= *= /= %= += -=	assignment	right to left

Library Functions

Library functions which are also called as "built-in" functions are the functions that are already available and implemented in C++.

We can directly call these functions in our program as per our requirements. Library functions in C++ are declared and defined in special files called "Header Files" which we can reference in our C++ programs using the "include" directive.

The standard library contains library functions to perform standard input/output operations (*e.g.*, read and write characters, read and write numbers, open and close files, test for end of file, etc.), functions that perform operations on characters (*e.g.*, convert from lower- to uppercase, test to see if a character is uppercase, etc.), functions that perform operations on strings (*e.g.*, copy a string, compare strings, concatenate strings, etc.), functions that carry out various mathematical calculations (*e.g.*, evaluate trigonometric, logarithmic and exponential functions, compute absolute values, square roots, etc.), and much more.

For Example, to include all the built-in functions related to math, we should include <cmath> header as follows:

```
#include <cmath>
```

Most of the mathematical functions that you find on a pocket calculator are declared

CamScanner

in the <cmath> header file, including all those shown in the table 7.5.

Table 7.5. Some Functions Defined in the <cmath> Header

Function	Description	Example
`acos(x)`	inverse cosine of x (in radians)	`acos(0.2)` returns 1.36944
`asin(x)`	inverse sine of x (in radians)	`asin(0.2)` returns 0.201358
`atan(x)`	inverse tangent of x (in radians)	`atan(0.2)` returns 0.197396
`ceil(x)`	ceiling of x (rounds up)	`ceil(3.141593)` returns 4.0
`cos(x)`	cosine of x (in radians)	`cos(2)` returns -0.416147
`exp(x)`	exponential of x (base e)	`exp(2)` returns 7.38906
`fabs(x)`	absolute value of x	`fabs(-2)` returns 2.0
`floor(x)`	floor of x (rounds down)	`floor(3.141593)` returns 3.0
`log(x)`	natural logarithm of x (base e)	`log(2)` returns 0.693147
`log10(x)`	common logarithm of x (base 10)	`log10(2)` returns 0.30103
`pow(x,p)`	x to the power p	`pow(2,3)` returns 8.0
`sin(x)`	sine of x (in radians)	`sin(2)` returns 0.909297
`sqrt(x)`	square root of x	`sqrt(2)` returns 1.41421
`tan(x)`	tangent of x (in radians)	`tan(2)` returns -2.18504

Notice that every mathematical function returns a double type. If an integer is passed to the function, it is promoted to a double before the function processes it.

The table below lists some of the more useful header files in the Standard C++ Library.

Table 7.6. Some of the Header Files in the Standard C++ Library

Header File	Description
`<cassert>`	Defines the `assert()` function
`<ctype>`	Defines functions to test characters
`<cfloat>`	Defines constants relevant to floats
`<climits>`	Defines the integer limits on your local system
`<cmath>`	Defines mathematical functions
`<cstdio>`	Defines functions for standard input and output
`<cstdlib>`	Defines utility functions
`<cstring>`	Defines functions for processing strings
`<ctime>`	Defines time and date functions

These are derived from the Standard C Library. They are used the same way that Standard C++ header files such as `<iostream>` are used.

CamScanner

Exercise

7.1. What is an operator? Describe several different types of operators.

7.2. What is an operand? What is the relationship between operators and operands?

7.3. Describe the five arithmetic operators. Summaries the rules associated with their use.

7.4. What is the difference between the operators / and %?

7.5. What is meant by operator precedence? What is the relative precedence of the arithmetic operators?

7.6. What is meant by associativity? What is the associativity of the arithmetic operators?

7.7. When should parentheses be included within an expression? When should the use of parentheses be avoided?

7.8. In what order are the operations carried out within an expression that contains nested parentheses?

7.9. Suppose a, b and c are integer variables that have been assigned the values $a = 8$, $b = 3$ and $c = -5$. Determine the value of each of the following arithmetic expressions.

1. $a + b + c$ (Ans. 6)
2. $2 * b + 3 * (a - c)$ (Ans. 45)
3. a / b (Ans. 2)
4. a / c (Ans. −1)
5. $a \% b$ (Ans. 2)
6. $a * b / c$ (Ans. −4)
7. $a * (b / c)$ (Ans. 0, because b / c is zero)

7.10. Suppose a, b and c are floating-point variables that have been assigned the values $a = 8.8$, $b = 3.5$ and $c = -5.2$. Determine the value of each of the following arithmetic expressions.

1. $a + b + c$ (Ans. 7.1)
2. $2 * b + 3 * (a - c)$ (Ans. 49)
3. a / b (Ans. 2.51429)
4. a / c (Ans. −1.69231)
5. $a \% b$ (Ans. The remainder operator is not defined for floating-point operands.)
6. $a / (b + c)$ (Ans. −5.17647)
7. $(a / b) + c$ (Ans. −2.68571)
8. $2 * a / 3 * b$ (Ans. 20.53333)
9. $2 * a / (3 * b)$ (Ans. 1.67619)

7.11. Describe the six relational operators. With what type of operands can they be used? What type of expression is obtained?

CamScanner

7.12. Describe the three logical operators. What is the purpose of these operators? With what type of operands can they be used? What type of expression is obtained?

7.13. Write the logical statements for the following:

(a) basic_pay is more than 2200 for all employees whose category is P.

(b) stock of all items whose item_no is 100 and in the shop A.

7.14. What is the relative precedence of the relational and logical operators with respect to one another and with respect to the arithmetic operators?

7.15. Describe the logical NOT operator. What is its purpose? Within which precedence group is it included? How many operands does it require?

7.16. Explain the difference between prefix and postfix increment and how does it affect the result of an expression?

7.17. Explain the difference between `count=count+1` and `count++`.

7.18. What is the difference between `--count` and `count--`.

7.19. Can you mix different types of data in the same operation? If so, how will it affect the results of the operations.

7.20. What is the result after the execution of the following program segment.

```
a = 7;
b = ++a + 5;
c = b-- + 10;
```

7.21. What is an expression?

7.22. Evaluate following expressions independent of each other. Assume that `i`, `j`, are integers with values 3, 4, and 2 respectively.

(a) `i++ - j--`

(b) `i-- % j++`

(c) `j++ / i--`

(d) `k++ * --i`

(e) `i - 1 % j + 1`

7.23. Write C++ expressions corresponding to the following. Assume all quantities a of the type integer.

(a) $\frac{(ax+b)}{(ax-b)}$

(b) $\frac{(2x+3y)}{(x^3-6)}$

(c) $x^5+10x^4+8x^3+4x+2$

(d) $\frac{(4a+3)}{(2y^2+2z+2)}$

7.24. What is the final value of `f` in the following program segment:

```
float f;
int i;
f = 2.5;
f = (f + 0.04) * 10;
```

7.25. What does the cast operator do? Why is it useful?

CamScanner

Objectives and MCQs

Choose the answer that best completes the statement.

7.1. Which of the following is an example of initializing a variable?
- (a) num = 2;
- (b) int num;
- (c) num < 2;
- (d) int num = 2;

7.2. Floating point variables are used instead of integers to
- (a) avoid being too specific about what value a number has
- (b) permit the use of decimal points in numbers
- (c) make possible the use of even numbers
- (d) conceal the true value of the numbers

7.3. Which of the following is an arithmetic operator?
- (a) +
- (b) &
- (c) &&
- (d) <

7.4. Are the following expressions true or false?
- (a) `1 > 2`
- (b) `'a' < 'b'`
- (c) `1 == 2`
- (d) `'2' == '2'`

7.5. Precedence determines which operator
- (a) is most important
- (b) is used first
- (c) is fastest
- (d) operates on the largest number

7.6. A relational operator is used to
- (a) combine values
- (b) compare values
- (c) distinguish different types of variables
- (d) change variables to logical values

7.7. The standard library contains many useful functions called
- (a) initial functions
- (b) basic functions
- (c) library commands
- (d) library functions

CamScanner

Objectives

7.1. An expression is a collection of operands and operators.
7.2. Operators are used to perform some operation on the operands.
7.3. Unary operator requires one operand.
7.4. Arithmetic operators are used to perform mathematical operations.
7.5. Order of precedence means the order in which the multiple operations take place.
7.6. After testing the conditions, logical operators return true or false as net result.
7.7. Logical NOT operator takes only one operand.
7.8. The precedence of logical operators is lower than the relational operators.
7.9. C++ is very rich in operators and is sometimes called the language of operators.

By: Sir Ahad

CamScanner

CHAPTER 8

INPUT AND OUTPUT STATEMENTS

Input and output statements of computer languages are used to provide communications between the user and the program. In C++, I/O is a sequence of bytes, called a *stream*, from the source to the destination. The bytes are usually characters, unless the program requires other types of information, such as a graphic image or digital speech. Therefore, a *stream* is a sequence of characters from the source to the destination. There are two types of streams:

Input stream: A sequence of characters from an input device to the computer.

Output stream: A sequence of characters from the computer to an output device.

The standard input device is usually the keyboard, and the standard output device is usually the screen. To receive data from the keyboard and send output to the screen, every C++ program must use the header file `iostream`. This header file contains, among other things, the definitions of two data types, *istream* (input stream) and *ostream* (output stream).

1. `istream` is a general-purpose input stream. `cin` (pronounced "see-in"), which stands for *common input*, is an example of an `istream`.
2. `ostream` is a general-purpose output stream. `cout` (pronounced "see-out"), which stands for *common output*, is an example of `ostream`.

To use `cin` and `cout`, every C++ program must use the preprocessor directive:

```
#include <iostream>
```

CamScanner

Because `cin` and `cout` are already defined and have specific meanings, to avoid confusion, you should never redefine them in programs.

In C++ after the header files, we often use `using namespace std;`. The reason behind it is that all of the standard library definitions are inside the `namespace std`. As the library functions are not defined at global scope, so in order to use them we use `namespace std`. So, that we don't need to write `STD::` at every line (*e.g.*, `STD::cout` etc.).

Standard Output (cout)

In C++, `cout` sends formatted output to standard output devices, such as the screen. We use the `cout` object along with the *insertion operator* << (*i.e.*, two 'less than' signs) for displaying output.

```
cout << "Hello World";          // prints Hello World on screen
cout << 140;                    // prints number 120 on screen
cout << x;                      // prints the value of x on screen
```

The insertion operator << inserts the data that follows it into the stream. In the examples above, it inserted the string `Hello World`, the number 140, and the value of variable `x` into the standard output stream `cout`. Notice that the sentence in the first statement is enclosed in double quotes (") because it is a string, while in the last one, `x` is not. The double quoting is what makes the difference; when the text is enclosed between them, the text is printed as is; when they are not, the text is interpreted as the identifier of a variable, and its value is printed instead. For example, these two sentences have very different results:

```
cout << "Hello";                // prints Hello
cout << Hello;                  // prints the content of variable Hello
```

Multiple insertion operations (<<) may be chained in a single statement:

```
cout << "This " << " is a " << "single C++ statement";
```

This last statement would print the text `This is a single C++ statement`. Chaining insertions is especially useful to mix constants and variables in a single statement:

```
cout << "I am " << age << " years old and my height is " << height;
```

Assuming the age variable contains the value 21 and the height variable contains 5.8, the output of the above statement would be:

```
I am 21 years old and my height is 5.8
```

What `cout` does not do automatically is add line breaks at the end, unless instructed to do so. For example, take the following two statements inserting into `cout`:

```
cout << "This is a sentence. ";
cout << "This is another sentence.";
```

The output would be in a single line, without any line breaks in between. Something like:

CamScanner

```
This is a sentence. This is another sentence.
```

To insert a line break, a new-line character shall be inserted at the exact position the line should be broken. In C++, a new-line character can be specified as \n (*i.e.*, a backslash character followed by a lowercase n). For example:

```
cout << "First sentence.\n";
cout << "Second sentence.\nThird sentence.";
```

This produces the following output:

```
First sentence.
Second sentence.
Third sentence.
```

Alternatively, the endl manipulator can also be used to break lines. For example:

```
cout << "First sentence." << endl;
cout << "Second sentence.";
```

This would print:

```
First sentence.
Second sentence.
```

The endl manipulator produces a newline character, exactly as the insertion of \n does; but it also has an additional behavior: the stream's buffer (if any) is flushed, which means that the output is requested to be physically written to the device, if it wasn't already. This affects mainly *fully buffered* streams, and cout is (generally) not a *fully buffered* stream. Still, it is generally a good idea to use endl only when flushing the stream would be a feature and \n when it would not.

Program 8.1a

```
// program 8-1a.cpp
#include <iostream>
using namespace std;
int main()
{
  // prints the string enclosed in double quotes
  cout << "This is C++ Programming";
  return 0;
}
```

Output

```
This is C++ Programming
```

Note: If we don't include the `using namespace std;` statement, we need to use `std::cout` instead of `cout`.

This is the preferred method as using the `std` namespace can create potential problems.

CamScanner

Program 8.1b

```
// program 8-1b.cpp
#include <iostream>
int main()
{
  // prints the string enclosed in double quotes
  std::cout << "This is C++ Programming";
  return 0;
}
```

Program 8.2

To print the numbers and character variables, we use the same cout object but without using quotation marks.

```
// program 8-2.cpp
#include <iostream>
using namespace std;
int main()
{
  int num1 = 65;
  double num2 = 143.485;
  char ch = 'P';

  cout << "integer: " << num1 << endl;      // print integer
  cout << "double: " << num2 << endl;       // print double
  cout << "character: " << ch;              // print char
  return 0;
}
```

Output

```
integer: 65
double: 143.485
character: P
```

Note that the `endl` manipulator is used to insert a new line. That's why each output is displayed in a new line. The << operator can be used more than once if we want to print different variables, strings and so on in a single statement. For example:

```
cout << "integer: " << num1 << endl;
```

Standard Input (`cin`)

In C++, `cin` takes formatted input from standard input devices, such as the keyboard. We use the `cin` object along with the *extraction operator* >> (*i.e.*, two 'greater than' signs) for taking input. This operator is then followed by the variable where the extracted data is stored. For example:

```
int age;
cin >> age;
```

The first statement declares a variable of type `int` called `age`, and the second extracts

CamScanner

from cin a value to be stored in it. This operation makes the program wait for input from cin; generally, this means that the program will wait for the user to enter some sequence with the keyboard. In this case, note that the characters introduced using the keyboard are only transmitted to the program when the ENTER key is pressed.

The extraction operation on cin uses the type of the variable after the >> to determine how it interprets the characters read from the input; if it is an integer, the format expected is a series of digits, if a string a sequence of characters, etc.

Program 8.3

```
// program 8-3.cpp i/o example
#include <iostream>
using namespace std;
int main()
{
  int i;
  cout << "Please enter an integer value: ";
  cin >> i;          // Taking input
  cout << "The value you entered is " << i;
  cout << " and its double is " << i*2 << ".\n";
  return 0;
}
```

Output

```
Please enter an integer value: 13
The value you entered is 13 and its double is 26.
```

As you can see, extracting from cin seems to make the task of getting input from the standard input pretty simple and straightforward. But this method also has a big drawback. What happens in the example above if the user enters something else that cannot be interpreted as an integer? Well, in this case, the extraction operation fails. This, by default, lets the program continue without setting a value for variable i, producing undetermined results if the value of i is used later.

This is very poor program behavior. Most programs are expected to behave in an expected manner no matter what the user types, handling invalid values appropriately. Only very simple programs should rely on values extracted directly from cin without further checking. In the following section, we will see how stringstreams can be used to have better control over user input.

Extractions on cin can also be chained to request more than one data piece in a single statement:

```
cin >> a >> b;
```

This is equivalent to:

```
cin >> a;
cin >> b;
```

In both cases, the user is expected to introduce two values, one for variable a, and

CamScanner

another for variable b. Any kind of space is used to separate two consecutive input operations; this may either be a space, a tab, or a new-line character.

Program 8.4

```
// program 8-4.cpp
#include <iostream>
using namespace std;
int main()
{
  char a;
  int num;

  cout << "Enter a character and an integer: ";
  cin >> a >> num;

  cout << "Character: " << a << endl;
  cout << "Number: " << num;
  return 0;
}
```

A sample run of this program gives the output:

```
Enter a character and an integer: P
57
Character: P
Number: 57
```

Single Character Input

There are some functions in C++ that handle characters only. These functions are useful in some applications when the user is prompted to answer yes/no, or ask you to select from a menu of items, such as the number of the choice or the first letter of the choice name. The most common input function is `getchar()` function. The function is used without arguments, but you must keep the empty parentheses.

The `getchar()` Function

The function `getchar()` stands for "get character" and inputs a single character from the keyboard. When called, the function waits for a key to be pressed. You can input the character as either a `char` or `int` type because `getchar()` function treat characters as integers and accept the ASCII value of character as a variable of type `int`.

Therefore, to input a character, use either of these formats:

```
char letter;
letter = getchar();
```

or

```
int letter;
letter = getchar();
```

CamScanner

When the user presses a key, `getchar()` echoes (displays) the character that you typed to the screen and further it requires Enter key ↵ to be typed following the input character.

The following program demonstrates the use of the function `getchar()`. It begins with declaring a character variable "ch", then the function itself is assigned to this variable. This means that the character received by the function will be contained in the variable "ch". The first `cout` function displays the contents of the variable "ch" in the character format preceded by the string "You typed:". The second `cout` function displays the same variable in the decimal format preceded by the string "which corresponds to the ASCII:".

```
// program 8-5.cpp
#include <iostream>
using namespace std;
int main()
{
  char ch;
  ch = getchar();          // read a character

  cout << "You typed: " << ch << endl;
  cout << "which corresponds to the ASCII: " << int(ch) << endl;
  return 0;
}
```

A sample run of this program gives the output:

```
w
You typed: w
which corresponds to the ASCII: 119
```

The function `getchar()` can also be assigned to a variable of the type `int`. Just replace the word `char` by the word `int` in the above program and the program still gives the same results.

```
// program 8-6.cpp
#include <iostream>
using namespace std;
int main()
{
  int ch;
  ch = getchar();          // read a character

  cout << "You typed: " << char(ch) << endl;
  cout << "which corresponds to the ASCII: " << ch << endl;
  return 0;
}
```

Notice that if you enter more than one character, the function `getchar()` still reads the first one only. Also, if you press Enter key ↵ only, you get the number "10", which is the ASCII code for the new-line character.

CamScanner

Standard Input (cin) and Strings

The extraction operator can be used on cin to get strings of characters in the same way as with fundamental data types:

```
string mystring;
cin >> mystring;
```

However, cin extraction always considers spaces (whitespaces, tabs, new-line etc.) as terminating the value being extracted, and thus extracting a string means to always extract a single word, not a phrase or an entire sentence.

To get an entire line from cin, there exists a standard library function, called getline. It is a part of the <string> header. The getline() function extracts characters from the input stream and appends it to the string object until the delimiting character is encountered. If no delimiter is specified, the default is the newline character at the end of the line. The input value does NOT contain the delimiter. While doing so the previously stored value in the string object str will be replaced by the input string if any. Here is an example of using getlin() function.

Program 8.7

```
// program 8-7.cpp cin with getline() function
#include <iostream>
#include <string>
using namespace std;
int main()
{
  string str;
  cout << "What's your name? ";
  getline (cin, str);
  cout << "Hello " << str << ".\n";
  cout << "Who is your favorite cricketer? ";
  getline (cin, str);
  cout << "I like " << str << " too!\n";
  return 0;
}
```

In this code, we simply perform a getline() using cin as our input stream and a variable named str to hold the input obtained by getline(), then we pass this variable to the cout routing.

When the above code is compiled and executed, it produces the following result:

```
What's your name? Qasim Bhatti
Hello Qasim Bhatti.
Who is your favorite cricketer? Babar Azam
I like Babar Azam too!
```

Notice how in both calls to getline, we used the same string identifier (str). What the program does in the second call is simply replace the previous content with the new one that is introduced.

CamScanner

The standard behavior that most users expect from a program is that each time the program queries the user for input, the user introduces the field, and then presses ENTER. That is to say, input is generally expected to happen in terms of lines on computer screen, and this can be achieved by using `getline` to obtain input from the user. Therefore, unless you have a strong reason not to, you should always use `getline` to get input in your programs instead of extracting from `cin`.

stringstream

The standard header `<sstream>` defines a type called `stringstream` that allows a string to be treated as a stream, and thus allowing extraction or insertion operations from/to strings in the same way as they are performed on `cin` and `cout`. This feature is most useful to convert strings to numerical values and vice versa. For example, in order to extract an integer from a string we can write:

```
string mystr ("1204");
int myint;
stringstream(mystr) >> myint;
```

This declares a string with initialized to a value of "1204", and a variable of type `int`. Then, the third line uses this variable to extract from a `stringstream` constructed from the string. This piece of code stores the numerical value 1204 in the variable called `myint`.

Program 8.8

```
// program 8-8.cpp stringstream example
#include <iostream>
#include <string>
#include <sstream>
using namespace std;
int main ()
{
  string mystr;
  float price=0;
  int quantity=0;
  cout << "Enter price: ";
  getline (cin, mystr);
  stringstream(mystr) >> price;
  cout << "Enter quantity: ";
  getline (cin, mystr);
  stringstream(mystr) >> quantity;
  cout << "Total price: " << price*quantity << endl;
  return 0;
}
```

Output

```
Enter price: 22.25
Enter quantity: 7
Total price: 155.75
```

CamScanner

In this example, we acquire numeric values from the *standard input* indirectly: Instead of extracting numeric values directly from `cin`, we get lines from it into a string object (`mystr`), and then we extract the values from this string into the variables price and quantity. Once these are numerical values, arithmetic operations can be performed on them, such as multiplying them to obtain a total price.

With this approach of getting entire lines and extracting their contents, we separate the process of getting user input from its interpretation as data, allowing the input process to be what the user expects, and at the same time gaining more control over the transformation of its content into useful data by the program.

Practical 1

Object

Writing a program to input a 4 (four) digit number and display each digit in separate line.

Source Code

```
/* program 8-9.cpp: Practical 1 - Writing a program to input a 4
(four) digit number and display each digit in separate line. */
#include <iostream>
using namespace std;
int main()
{
  int num, d1, d2, d3, d4;

  // Prompt the user to enter a 4-digit number
  cout << "Enter a 4-digit number: ";
  cin >> num;

  // Extract each digit from the number
  d1 = num / 1000;
  d2 = (num / 100) % 10;
  d3 = (num / 10) % 10;
  d4 = num % 10;

  // Display each digit on a separate line
  cout << d1 << endl;
  cout << d2 << endl;
  cout << d3 << endl;
  cout << d4;
  return 0;
}
```

Output

```
Enter a 4-digit number: 6947
6
9
4
7
```

CamScanner

This program uses the modulo operator % and integer division / to extract each digit from the input number. In this program, the user is prompted to enter a 4-digit number, which is stored in the variable num. Then, each digit is extracted using mathematical operations. The thousands digit is obtained by dividing the number by 1000, the hundreds digit is obtained by dividing the number by 100 and taking the remainder after dividing by 10, the tens digit is obtained by dividing the number by 10 and taking the remainder after dividing by 10, and the units digit is obtained by taking the remainder of the number divided by 10. Finally, each digit is displayed on a separate line using cout.

C++ Formatting Output

Formatting output in C++ is important in the development of the output screen, which can be easily read and understood. C++ offers the programmer several input/output manipulators. Two of these widely used input/output manipulators are:

```
setw()
setprecision()
```

In order to use these manipulators, you must include the header file named iomanip, *i.e.*, `include #include <iomanip>`

The setw() manipulator

In C++, the `setw()` manipulator sets the width of the field assigned for the output. The field width determines the minimum number of characters to be written in output. The syntax is:

```
setw(number of characters)
```

Here is an example of this code line:

```
cout << setw(6) << "A";
```

generates the following output on the screen (each underscore represents a blank space).

_ _ _ _ _ A

The `setw()` manipulator does not stick from one cout statement to the next. For example, if you want to right-justify three numbers within an 8-space field, you will need to repeat `setw()` for each value, as shown below:

```
cout << setw(8) << 22 << "\n";
cout << setw(8) << 4444 << "\n";
cout << setw(8) << 666666 << endl;
```

The output will be (each underscore represents a blank space):

```
_ _ _ _ _ _ 2 2
_ _ _ _ 4 4 4 4
_ _ _ _ 6 6 6 6 6 6
```

Note: The endl and "\n" are both used to break the line.

CamScanner

Here is an example of using setw( ) manipulator to calculate net salary.

Program 8.10

```
// program 8-10.cpp using setw() manipulator
#include <iostream>
#include <iomanip>
using namespace std;
int main()
{
  float basic, ca, hra, gpf, tax, gross, np;
  cout << "Enter Basic Pay: ";
  cin >> basic;
  cout << "Enter Conveyance Allowance: ";
  cin >> ca;
  cout << "Enter House Rent Allowance: ";
  cin >> hra;
  gross = basic+ca+hra; // sum of basic, ca and hra
  gpf = (basic+ca) * 0.10; // 10% of basic and ca
  tax = gross * 0.10; //10% of gross pay
  np = gross - (gpf+tax); //netpay = earnings – deductions

  cout<<setw(25)<<endl<<"Basic Pay: "<<setw(10)<< basic<<endl;
  cout<<setw(25)<<"Conveyance Allowance: "<<setw(10)<<ca<<endl;
  cout<<setw(25)<<"House Rent Allowance: "<<setw(10)<<hra<<endl;
  cout<<setw(25)<<"Gross Pay: "<<setw(10)<<gross<<endl;
  cout<<setw(25)<<"General Provident Fund: "<<setw(10)<<gpf<<endl;
  cout<<setw(25)<<"Income Tax: "<<setw(10)<<tax<<endl;
  cout<<setw(25)<<"Net Pay: "<<setw(10)<<np;

  return 0;
}
```

A sample run of this program gives the output:

```
Enter Basic Pay: 1000
Enter Conveyance Allowance: 200
Enter House Rent Allowance: 300

                Basic Pay:       1000
  Conveyance Allowance:           200
  House Rent Allowance:           300
                Gross Pay:       1500
  General Provident Fund:         120
              Income Tax:         150
                  Net Pay:       1230
```

In the above program, every output statement has two setw( ) manipulators; first setw creates a filed with 25 spaces and second setw( ) creates another field with 10 spaces. When you represent a value to these fields, it will show the value within the field from right to left.

CamScanner

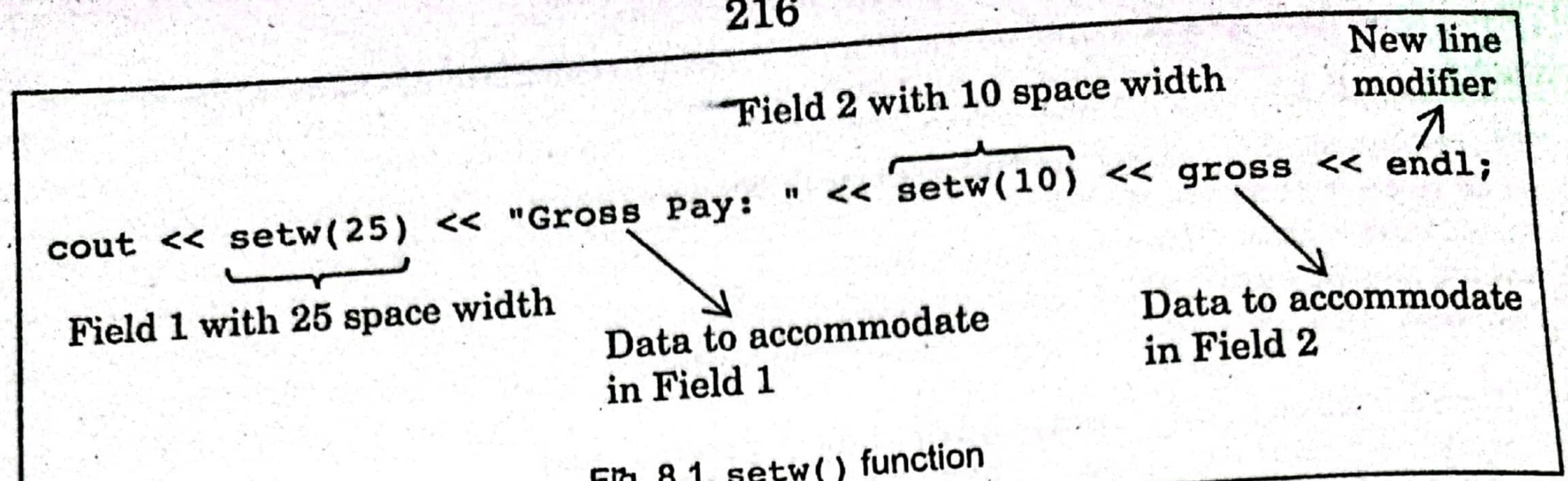


Fig. 8.1. setw() function

In field 1 and field 2, the string "Gross Pay: " and the value of gross pay are shown as given in Fig. 8.2.

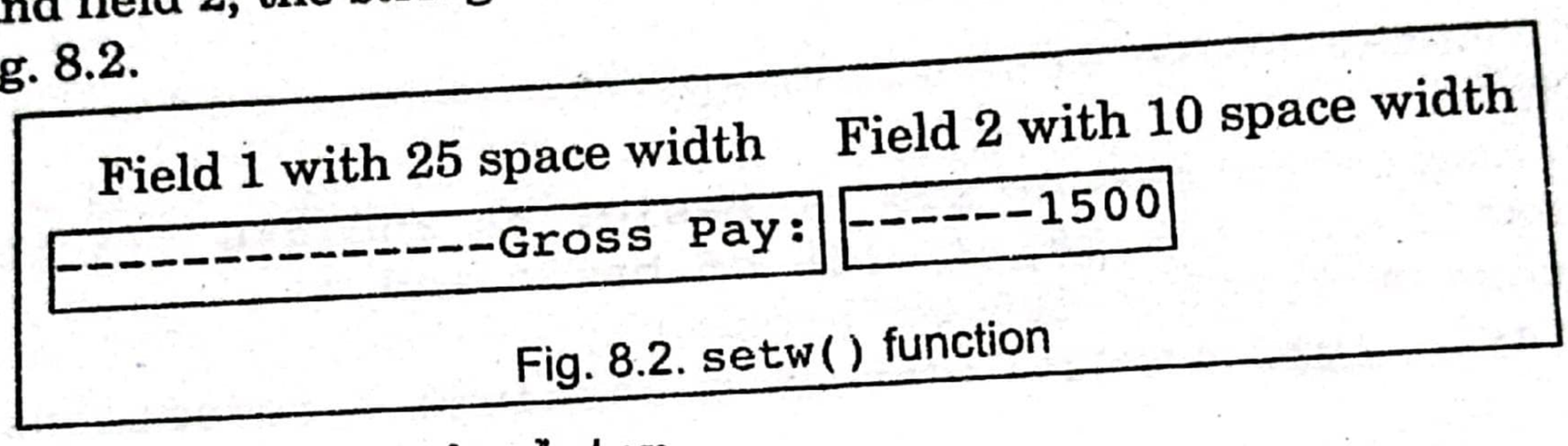


Fig. 8.2. setw() function

The setprecision() manipulator

In C++, the setprecision() manipulator sets the total number of digits to be displayed when floating-point numbers are printed. Here is an example:

```
cout << setprecision(5) << 836.746;
```

will print the following output to the screen (notice the rounding):

```
836.75
```

The following program is a simple implementation of set precision in C++ which will display the number of significant digits after decimal in a double type number.

Program 8.11

```
// program 8-11.cpp to demonstrate the working of setprecision()
function
#include <iostream>                    // for cout and fixed
#include <iomanip>                     // for setprecision
using namespace std;
int main ()
{
  double num = 3.14159;

  cout << setprecision(5) << num << '\n';
  cout << setprecision(9) << num << '\n';
  cout << fixed;
  cout << setprecision(5) << num << '\n';
  cout << setprecision(9) << num << '\n';
  return 0;
}
```

CamScanner

Output

```
3.1416
3.14159
3.14159
3.141590000
```

In the statement `cout << fixed;` of above program, `"fixed"` is referred to as a format option. Other format alternatives include the following:

Format Value	Meaning
left	left-justify the output
right	right-justify the output
showpoint	displays decimal point and trailing zeros for all floating-point numbers, even if the decimal places are not needed
uppercase	display the 'e' in E-notation as 'E' rather than 'e'
showpos	display a leading plus sign before positive values
scientific	display floating point numbers in scientific ('E') notation
fixed	display floating point numbers in normal notation—no trailing zeroes and no scientific notation

Note that all the subsequent `couts` retain the precision set with the last `setprecision()`. That means `setprecision()` is 'sticky.' Whatever precision you set sticks with the `cout` device until such time as you change it with an additional `setprecision()` later in the program.

CamScanner

Exercise

8.1. What is output?

8.2. What are the commonly used input/output functions in C++? How are they accessed?

8.3. What is the difference between `STD::` and `using namespace std`?

8.4. What happens if an input data item contains more characters than the maximum allowable field width? What if the data item contains fewer characters?

8.5. What is the purpose of the `getline` function? How is it used within a C++ program?

8.6. Identify and correct the error in each of the following:

(a) `cout >> count;`

(b) `int double=44;`

8.7. Write a program that sounds a bell.

8.8. Write a program that prints the block letter "B" in a 7 × 6 grid of stars like this:

```
*****
*    *
*    *
*****
*    *
*    *
*****
```

8.9. Write a program which prints two groups of five numbers each with right justification as,

```
    1
   12
  123
 1234
12345
   -1
  -12
 -123
-1234
-12345
```

Ans.

```
// program 8-12.cpp
#include <iostream>
#include <iomanip>
using namespace std;
int main()
{
  cout << setw(6) << 1 << endl;
```

CamScanner

```
cout << setw(6) << 12 << endl;
cout << setw(6) << 123 << endl;
cout << setw(6) << 1234 << endl;
cout << setw(6) << 12345 << endl;
cout << setw(6) << -1 << endl;
cout << setw(6) << -12 << endl;
cout << setw(6) << -123 << endl;
cout << setw(6) << -1234 << endl;
cout << setw(6) << -12345 << endl;
return 0;
}
```

8.10. How do you display the value of a numeric variable?

8.11. Write a program that asks the user to enter the radius of a circle and then computes and displays the circle's area. Use the formula

$$area = PI \times radius \times radius$$

where *PI* is the constant whose value is `3.14159`.

8.12. Write a program that converts Fahrenheit temperatures to Celsius temperatures with 3 digits of precision. Use the formula

```
Celsius = 5.0/9.0 * (Fahrenheit - 32);
```

to perform the calculation.

Objectives

8.1. In C++, I/O is a sequence of bytes from the source to the destination, called a stream.

8.2. This header file `iostream` contains the definitions of two data types, istream and ostream.

8.3. `istream` is a general-purpose input stream.

8.4. `ostream` is a general-purpose output stream.

8.5. We use the `cout` object along with the insertion operator <<.

8.6. We use the `cin` object along with the extraction operator >>.

8.7. `cin` extraction always considers spaces (whitespaces, tabs, new-line etc.) as terminating the value being extracted. (T/F)

8.8. The standard header `<sstream>` defines a type called stringstream that allows a string to be treated as a stream.

8.9. The `setw()` manipulator sets the width of the field assigned for the output.

8.10. The `setprecision()` manipulator sets the total number of digits to be displayed when floating-point numbers are printed.

CamScanner

CHAPTER 9

SELECTION CONTROL STRUCTURES

By: Sir Ahad

In the early days of programming, as programmers began writing longer and more sophisticated software, they quickly ran into difficulties. Their programs consisted of long lists of instructions, one after the other. It was hard to determine where one task stopped and another started. As a result, the logic of the programs was difficult to follow and making modifications at a later time was extremely difficult. In the mid-1960s, the idea of structure programming was conceived to solve these problems, and it is now the standard for programming. *Structured programming* is a standardized approach that creates a program as a series of interrelated sub-programs, or functions. It also makes programs easier to write, check, read, and modify. Here we will examine the basic concepts of structure in C++ language.

In a program, *control structures* control the flow of execution in a program or function. Structured programming uses a limited number of control structures to minimize the complexity of programs and thus cut down on errors. There are three basic control structures in structured programming:

1. Sequence, or sequential flow
2. Selection, or conditional flow
3. Iteration, or repetitive flow

The *sequence control structure* is the most straightforward: the statements are executed in the same order in which they appear within the program. The programs written in the previous chapters are all sequential programs. We say that such a program is a sequence of steps, meaning that

CamScanner

1. At a time only one step is to be carried out.
2. Every step is to be carried out once and only once; none is repeated and none is omitted.
3. The order of executing the steps is the same as that in which they are written.
4. Termination of the last step indicates the end of the program.

The Fig. 9.1 shows the general format of a sequence control structure using flowchart.

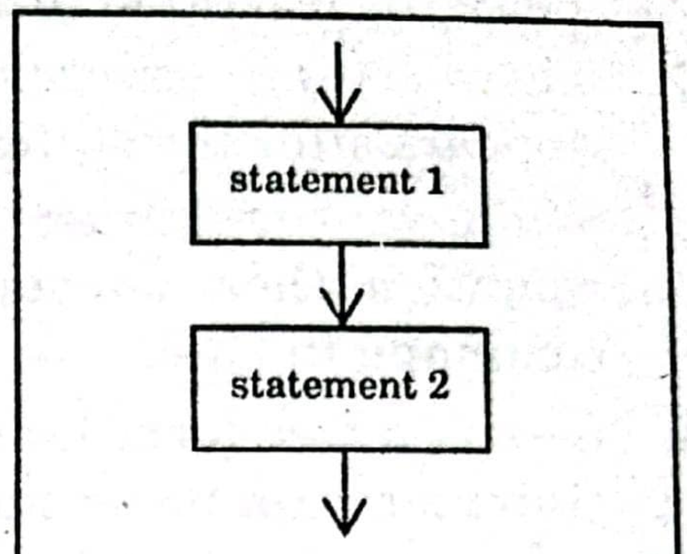


Fig. 9.1. Sequence control structure flowchart.

This chapter describes the C++ control structures for *selection* and chapter 10 covers the control structures for *iteration* or *repetition*.

The *selection control structure* employs a number of conditions which lead to a selection of one out of several alternatives. C++ provides three types of selection structures. The `if` selection structure either performs (selects) an action if a condition is true or skips the action if the condition is false. The `if/else` selection structure performs an action if a condition is true and performs a different action if the condition is false. The `switch` selection structure performs one of many different actions depending on the value of an expression.

C++ provides three types of repetition structures, namely the `while`, and the `do/while` and the `for` (discussed in chapter 10).

That is all there is. C++ has only seven control structures: Sequence, three types of selection and three types of repetition. Each C++ program is formed by combining as many of each type of control structure as is appropriate for the program.

The `if` Selection Structure

The `if` statement tests a particular condition. Whenever that condition evaluates as *true*, an action is performed—but if it is not true, then the action is skipped and control transfers to the next step of the program.

The general structure of the command is:

```
if(condition)
   statement;
```

The command says "if this condition is true, then perform the following statement" (see Fig. 9.2). The computer performs the statement, then continues normally, performing the statements that follow the `if` command.

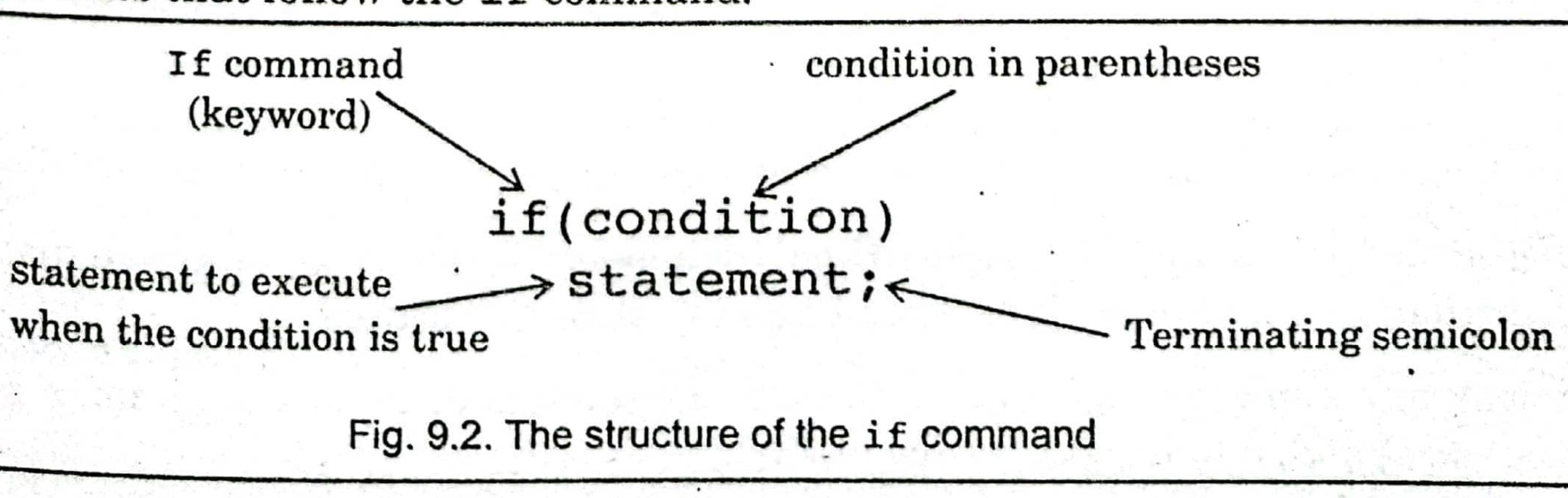


Fig. 9.2. The structure of the `if` command

CamScanner

If the condition is false, however, the computer skips the statement that is part of the `if` command, and moves on to the next statement in the program. Writing an `if` command is easy as long as you remember these basic points:

1. Place the condition in parentheses.
2. There is no semicolon after the condition—it appears after the statement.
3. While we paraphrase the command by saying "If condition then do statement," there is no *then* command in C++.
4. C++ is a free-form language, so the condition and statement can be on the same line. Separating and indenting the condition, however, make a program easier to read.

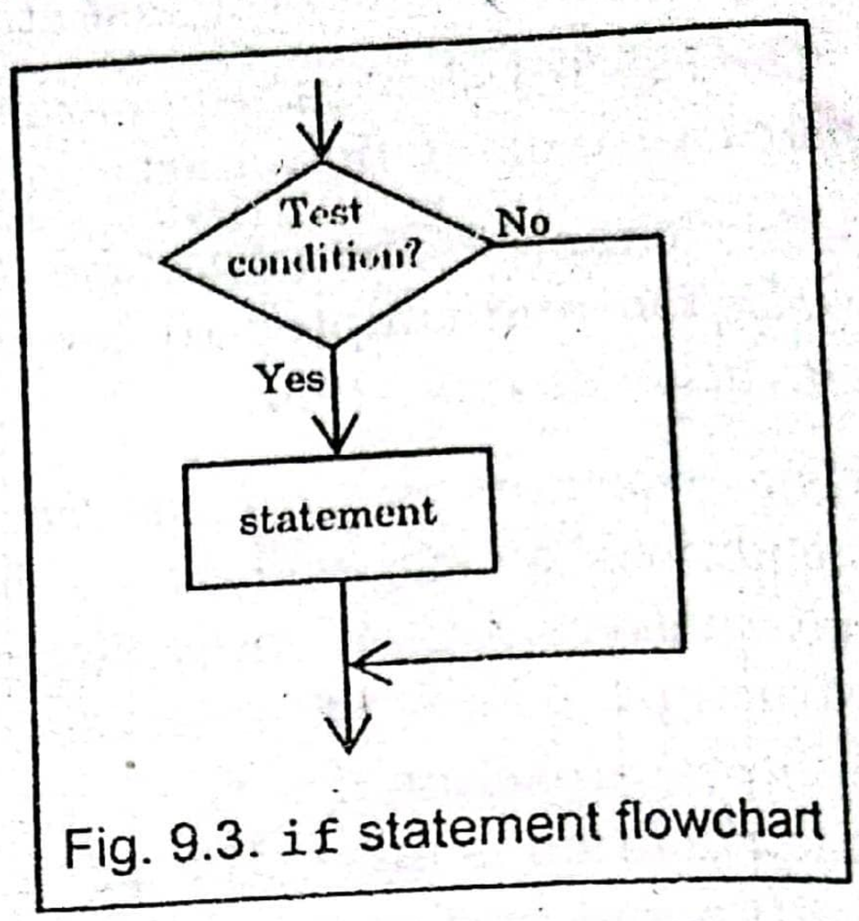


Fig. 9.3. `if` statement flowchart

The `if` statement can be visualized using the flowchart, as shown in Fig. 9.3.

Conditions in `if` commands compare values—a variable or constant is compared with another variable or constant. The comparison is done with one of these relational operators:

Operator	Meaning	Example
==	equal	`if(salary == 10000)`
<	less than	`if(hours < 40)`
>	greater than	`if(hours > 40)`
<=	less than or equal to	`if(cost <= limit)`
>=	greater than or equal to	`if(cost >= limit)`
!=	not equal to	`if(count != 1)`

Note that when testing whether two values are the same, you must use the == operator, two equal signs in a row. If you use a single equal sign, your compiler will generate an error. The single = symbol is reserved for assigning value to variables.

For example, to print the word `Overtime!` when hours are greater than 40, we would write

```
if(hours > 40)
   cout << "Overtime!\n";
```

If the variable `hours` were 40 or less, the word `Overtime!` would not be displayed.

Similarly, in the example

```
if(percentage_marks >= 80)
   cout << "Congratulations!\n";
cout << "Your grade is\n" << grade;
```

a congratulatory message is displayed only when the value of `percentage_marks` is greater than or equal to 80. The second `cout` is always executed.

Now look at complete program that uses an `if` command. In this program, a special luxury tax is added to items over Rs. 40,000.

CamScanner

```
// program 9-1.cpp
#include <iostream>
using namespace std;
int main()
{
  float cost, tax, luxury, total;
  luxury = 0.0;
  cout << "Enter the cost of the item: ";
  cin >> cost;
  tax = cost * 0.06;
  if (cost > 40000.00)
    luxury = cost * 0.005;
  total = cost + tax + luxury;
  cout << "The total cost is " << total;
  return 0;
}
```

The calculation is performed in the command:

```
if (cost > 40000.00)
   luxury = cost * 0.005;
```

The command says "If the value of `cost` is over Rs. 40,000, then assign the variable name `luxury` the value of `cost` times 0.5 percent". The last two instructions in the program are always performed, whether there is a luxury tax or not, because they come after the `if` command's terminating semicolon.

Notice that the variable `luxury` is assigned an initial value of 0.0 earlier in the program, but not the variable `tax`. The variable `tax` will always be assigned a value in the calculation `tax = cost * 0.06`, but `luxury` is only assigned a calculated value if the cost is over Rs. 40,000. If the condition is false, the calculation would not be performed and, if you did not assign an initial value, some unknown value would be added to the total. So as a general rule, it is wise to assign initial values to variables calculated in `if` statements, as well as to counters and accumulators.

Multiple Statements

The basic `if` command performs one statement. If you want to perform more than one statement when the condition is true, you must use a second level of braces. The braces mark the beginning and end of a block of statements performed when the condition is true:

```
if(condition)
  {
    statement 1;
    statement 2;
    statement 3;
  }
```

In the following example two `if` statements are used:

CamScanner

```
if(j < k)
   minutes = j;
if(j < k)
   cout << "j is smaller than k\n";
```

The more efficient and more understandable segment of the program can be written by using a single `if` statement with a compound statement for its body.

```
if(j < k)
  {
    minutes = j;
    cout << "j is smaller than k\n";
  }
```

Now look at the program 9-2, which is the modified form of program 9-1. In this program two statements are performed when the condition is true.

```
// program 9-2.cpp
#include <iostream>
using namespace std;
int main()
{
  float cost, tax, luxury, total;
  luxury = 0.0;
  cout << "Enter the cost of the item: ";
  cin >> cost;
  tax = cost * 0.06;
  if (cost > 40000.00)
  {
     luxury = cost * 0.005;
     cout << "The luxury tax is " << luxury << endl;
  }
  total = cost + tax + luxury;
  cout << "The total cost is " << total;
  return 0;
}
```

Nested `if` Statements

An `if` condition can be written as deeply as you need within the body of another `if` statement. This is called *nesting* of `if`'s. For example, an action is performed if two conditions are true. The condition 1 is tested and then condition 2 is tested.

```
if(condition 1)
   if(condition 2)
      statement;
```

The second `if` command in the following program segment is nested in the first:

```
if(income > 100000)
   if(status == 'S')
      tax_rate = 0.35;
```

The second `if` condition is tested only when the first condition is true, so the variable

CamScanner

`tax_rate` is assigned only when both conditions are true. The same logic could be written as:

```
if(income > 100000 && status == 'S')
   tax_rate = 0.35;
```

Both perform the same task. The nested `if` statement can be visualized using the flowchart, as shown in Fig. 9.4.

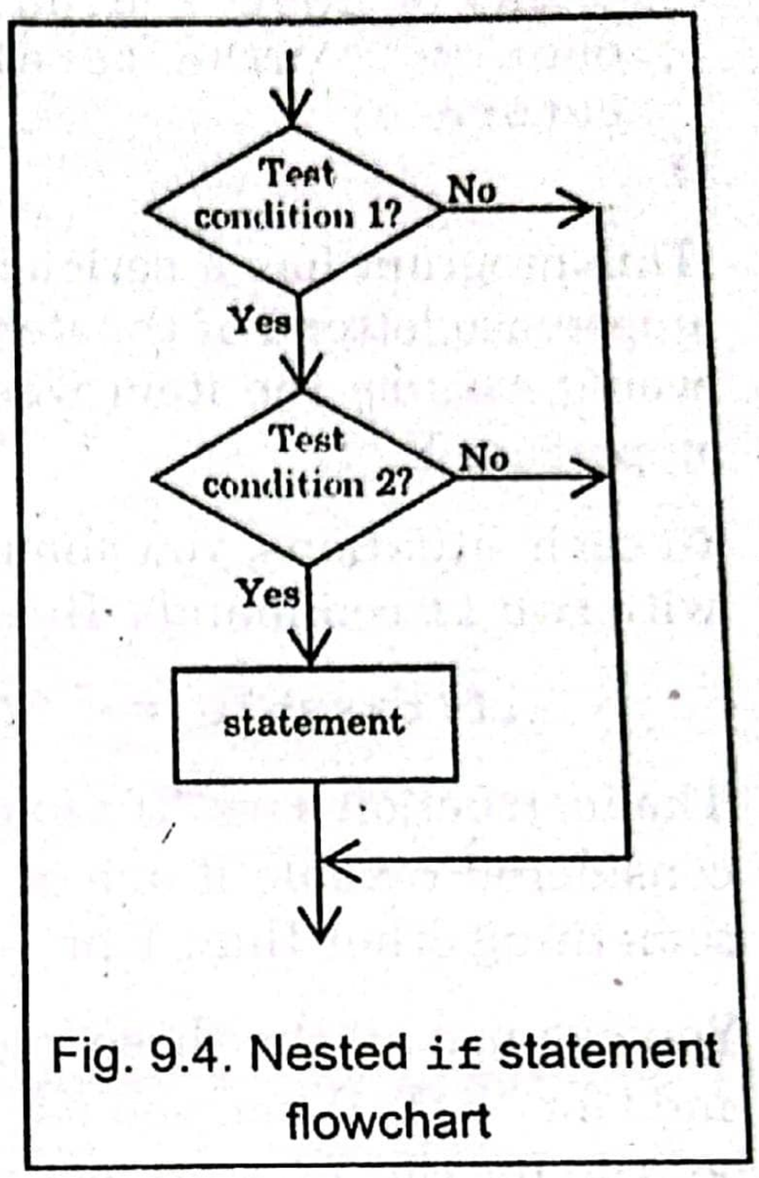


Fig. 9.4. Nested `if` statement flowchart

Consider another simple program:

```
// program 9-3.cpp
#include <iostream>
using namespace std;
int main()
{
  if (getchar() == 'm')
  {
    if (getchar() == 'e')
      cout << "\nYou typed me.";
  }
  return 0;
}
```

In the above program, the inner `if` statement will not be executed unless the outer one is true, and the `cout` statement will not be executed unless both `if` statements are true. Analysis of the program shows:

b	→	a non `'m'` as first letter terminates the program
mf	→	a non `'e'` as second letter terminates the program
me	→	only `'m'` followed by `'e'` executes `cout` statement

Logical Operators

The `if` statement test only a single variable and a single value. But sometimes programs need to test for more than one value.

Look at the program 9-4, which asks whether the item is taxable or not, then if the item is taxable, adds a 6 percent sales tax.

```
// program 9-4.cpp
#include <iostream>
using namespace std;
int main()
{
  float cost, tax;
  char taxable;
  tax = 0.0;

  cout << "Enter Y if the item is taxable, N if non-taxable: ";
  taxable = getchar();
  cout << "Enter the cost of the item: ";
```

CamScanner

```
  cin >> cost;
  if (taxable == 'Y')
    tax = cost * 0.06;
  cout << "\nThe total due is " << cost+tax;
  return 0;
}
```

This program has a serious flaw in it. The program assumes the user will enter an uppercase letter *Y* if the item is taxable. If the user enters a lowercase *y*, the program would assume the item was not taxable because the `if` command tests only for an uppercase *Y*.

In such situations, you should test for both possible inputs—*Y* or *y*. You could do this with two `if` commands. But instead, use the logical OR operator (||) like this:

```
if(taxable == 'Y' || taxable == 'y')
```

The instruction says "If `taxable` equals *Y* OR `taxable` equals *y*". So, the item will be considered taxable if one or the other condition is true. If neither is true—you enter something other than *Y* or *y*—the item would be non-taxable.

You can use all the three logical operators with `if` statement: || for OR, && for AND, and ! for NOT. When you use the OR operator, one or the other condition (or both) must be true for the `if` statement to be true. When you use the AND condition, both must be true. When you use the NOT operator the condition must be false.

For example, suppose you need to write a program that inputs the user's income and number of dependents, as in program 9-5:

```
// program 9-5.cpp
#include <iostream>
using namespace std;
int main()
{
  int dependents;
  float income;

  cout << "Please enter your income: ";
  cin >> income;
  cout << "Please enter the number of dependents: ";
  cin >> dependents;
  if (income < 5000 && dependents > 5)
     cout << "You owe no income tax";
  return 0;
}
```

The `if` condition tests two variables, `income` and `dependents`. The income must be less than Rs. 5000 AND the number of dependents must be greater than 5 for the message to be displayed. If either condition were false, such as one 2 dependents or Rs. 5001 of income, the `cout` statement would not be performed.

CamScanner

Program 9.6

This program tells whether a number entered from the keyboard is positive or negative.

```
// program 9-6.cpp
#include <iostream>
using namespace std;
int main()
{
  int num;
  cout << "Enter a number: ";
  cin >> num;
  if (num > 0)
    cout << "Number is positive";
  if (num < 0)
    cout << "Number is negative";
  return 0;
}
```

Program 9.7

This program displays two numbers and asks the user what the answer is. The program then tells the user if the answer is right or wrong.

```
// program 9-7.cpp
#include <iostream>
using namespace std;
int main()
{
  int answer;
  cout << "What is 10 + 15? ";
  cin >> answer;
  if (answer == 10+15)
     cout << "Correct!";
  return 0;
}
```

Program 9.8

This program either converts feet to meters or meters to feet, depending upon what the user requests.

```
// program 9-8.cpp
#include <iostream>
using namespace std;
int main()
{
  float num;
  int choice;
  cout << "Enter value: ";
  cin >> num;
```

CamScanner

```
  cout << "\n1: feet to metres\n2: metres to feet";
  cout << "\n\nEnter choice: ";
 cin >> choice;
  if (choice == 1)
    cout << num/3.281;
  if (choice == 2)
    cout << num*3.281;
  return 0;
}
```

Example 9.1

The current luxury tax on cars is 10 percent but the tax is applied only to the portion of the sales price over Rs. 100,000. Write a program to calculate the tax for a given sales price.

Algorithm

Step 1. Input cost.
Step 2. IF cost > Rs. 100,000
tax = (cost − 100,000)*tax_rate.
Output tax.
[End of IF structure.]
Step 3. Exit.

Source Code

```
// program 9-9.cpp
#include <iostream>
using namespace std;
int main()
{
  const float cut_off = 100000;
  const float tax_rate = 0.1;
  float cost, tax;

  cout << "\nLuxury tax computer ... Enter cost of car: ";
  cin >> cost;
  if (cost > cut_off)
  {
    tax = (cost - cut_off)*tax_rate;
    cout << "\nFor a car costing Rs. " << cost << ", a luxury tax
of Rs. " << tax << " is due.";
  }
  cout << "\n\nThank you for your cooperation.";
  return 0;
}
```

Output

```
Luxury tax computer ... Enter cost of car: 200000
For a car costing Rs. 200000.00, a luxury tax of Rs. 10000.00 is due.
Thanks you for cooperation.
```

CamScanner

The `if-else` Selection Structure

The `if` selection structure performs an indicated action only when the condition is true, otherwise the action is skipped. Whereas, `if-else` selection structure performs certain action when the condition is true and some different action when the condition is false.

The general structure of the command is:

```
if(condition)
   statement 1;
else
   statement 2;
```

The structure says "if the condition is true, perform the statement 1; else perform the statement 2". The statement 2 which follows `else` is performed only if the condition is false. If you need to perform more than one statement in either case, enclose the multiple statements in braces. You need a semicolon after each statement but not after the `else` keyword. The `if-else` statement can be visualized using the flowchart as shown in Fig. 9.5.

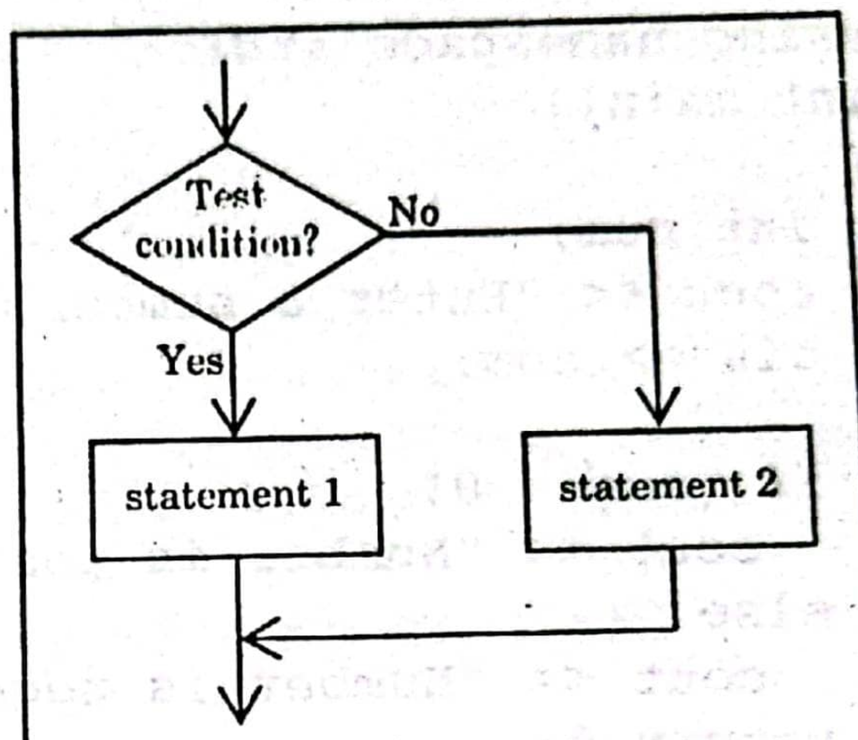


Fig. 9.5. if-else statement flowchart

For example, to find the larger of two variables you can use `if-else` statement:

```
if(x > y)
   max = x;          /*NOTE: semicolon required here! */
else
   max = y;
```

If `x > y` is *true*, then `max` will be assigned the value of `x`, and if it is *false*, then `max` will be assigned the value of `y`. As before, we can make statement 1 and statement 2 either single or compound by placing braces appropriately.

To print the message when no luxury tax is charged, you would modify the program 9-2 as illustrated below:

```
if (cost > 40000.00)
{
   luxury = cost * 0.005;
   cout << "The luxury tax is " << luxury << endl;
}
else
{
   cout << "There is no luxury tax for this item";
   luxury = 0.0;
}
```

Note that we no longer need to assign luxury an initial variable at the start of the program because the `if` command now accounts for all possible conditions.

CamScanner

Example 9.2

You can use the `else` to create more efficient code in some cases. For example, here the `else` is used in place of a second `if` in the program given in Example 9.1, which determines whether a number is positive or negative.

```
// program 9-10.cpp
#include <iostream>
using namespace std;
int main()
{
  int num;
  cout << "Enter a number: ";
  cin >> num;

  if (num > 0)
    cout << "Number is positive";
  else
    cout << "Number is negative";
  return 0;
}
```

Since there are only two possibilities, there is no reason to test num again to see if it is 0 or greater. The program in Example 9.1 tested for num to be less than 0 using a second `if` statement. Because of the way a C++ compiler generates code, the `else` requires far fewer machine instructions than an additional `if`.

Example 9.3

This program prompts the user for two numbers, divides the first by the second, and displays the result. However, division by 0 is undefined, so the program uses an `if` and an `else` statement to prevent division by 0 from occurring.

```
// program 9-11.cpp
#include <iostream>
using namespace std;
int main()
{
  int num1, num2;
  cout << "Enter first number: ";
  cin >> num1;
  cout << "Enter second number: ";
  cin >> num2;

  if (num2 == 0)
    cout << "Cannot divide by zero";
  else
    cout << "Answer is: " << num1/num2;
  return 0;
}
```

CamScanner

Program 9.12

```
/* program 9-12.cpp: Writing a Program which uses operator
(calculate the area of triangle, volume of sphere and arrange the
resultant values in ascending order).*/
#include <iostream>
#include <cmath>
#include <iomanip>                    // for setprecision
#define pi 3.1415927
using namespace std;
int main()
{
  float a, b, r, t, s;

  cout << "\n\t\tEnter the value for Altitude of a Triangle. ";
  cin >> a;
  cout << "\n\t\tEnter the value for Base of a Triangle. ";
  cin >> b;
  cout << "\n\t\tEnter the value for Radius of a Sphere. ";
  cin >> r;

  t=(1.0/2.0)*(a*b);
  s=(4.0/3.0)*pi*pow(r,3);

  if (t < s)
    {
     cout << "\n\t\tArea of Triangle = " << fixed << setprecision(2) << t;
     cout << "\n\t\tArea of Sphere = " << fixed << setprecision(2) << s;
    }
   else
    {
     cout << "\n\t\tArea of Sphere = " << fixed << setprecision(2) << s;
     cout << "\n\t\tArea of Triangle = " << fixed << setprecision(2) << t;
    }
  return 0;
}
```

Output

```
Enter the value for Altitude of a Triangle. 3.65
Enter the value for Base of a Triangle. 4.98
Enter the value for Radius of a Sphere. 9.29

Area of Triangle = 9.09
Area of Sphere = 3358.43
```

Nested `if-else` Statements

Nested `if-else` statements test for multiple conditions by placing `if-else` statements inside `if-else` statements. When a condition is evaluated as true, the corresponding statements are executed and the rest of the structure is skipped. This structure is referred to as the *`if-else-if` ladder*.

CamScanner

To demonstrate the use of the ladder, consider the following program which convert a month number to a month name:

```
// program 9-13.cpp
#include <iostream>
using namespace std;
int main()
{
  int month;
  cout << "Input month number: ";
  cin >> month;

  if (month == 1)
    cout << "January";
  else if (month == 2)
    cout << "February";
  else if (month == 3)
    cout << "March";
  else if (month == 4)
    cout << "April";
  else if (month == 5)
    cout << "May";
  else if (month == 6)
    cout << "June";
  else if (month == 7)
    cout << "July";
  else if (month == 8)
    cout << "August";
    else if (month == 9)
      cout << "September";
    else if (month == 10)
      cout << "October";
    else if (month == 11)
      cout << "November";
    else if (month == 12)
      cout << "December";
    else
      cout << "Error in month number!!!!";
    return 0;
}
```

As another example of ladder, the following program will print A-1 for exam percentage greater than or equal to 80, A for percentage greater than or equal to 70, B for percentage greater than or equal to 60, C for percentage greater than or equal to 50, D for percentage greater than or equal to 40, E for percentage greater than or equal to 33, and F for Fails. The program may be written in C as:

```
// program 9-14.cpp
#include <iostream>
using namespace std;
```

CamScanner

```
int main()
{
  float percent;
  cout << "Input percentage of marks: ";
  cin >> percent;
  if (percent >= 80)
    cout << "Your grade is A-1\n";
  else if (percent >= 70)
    cout << "Your grade is A\n";
  else if (percent >= 60)
    cout << "Your grade is B\n";
  else if (percent >= 50)
    cout << "Your grade is C\n";
  else if (percent >= 40)
    cout << "Your grade is D\n";
  else if (percent >= 33)
    cout << "Your grade is E\n";
  else
    cout << "Your grade is F\n";
  return 0;
}
```

If the variable percent is greater than or equal to 80, the first six conditions will be true, but only the cout statement after the first test will be executed. After that cout is executed, the else-part of the "outer" if-else statement is skipped.

The if selection structure expects only one statement in its body. To include several statements in the body of an if, enclose the set of statements in braces. A set of statements contained within a pair of braces is called a *compound statement*.

The following example includes a compound statement in the else part of an if-else structure.

```
if (percent >= 33)
   cout << "Passed.\n";
else
   {
   cout << "Failed.\n";
   cout << "You must work hard.";
   }
```

In this case, if percent is less than 33, the program executes both cout statements in the body of the else and displays

```
Failed.
You must work hard.
```

The ladder is diagrammed in Fig. 9.6. Notice

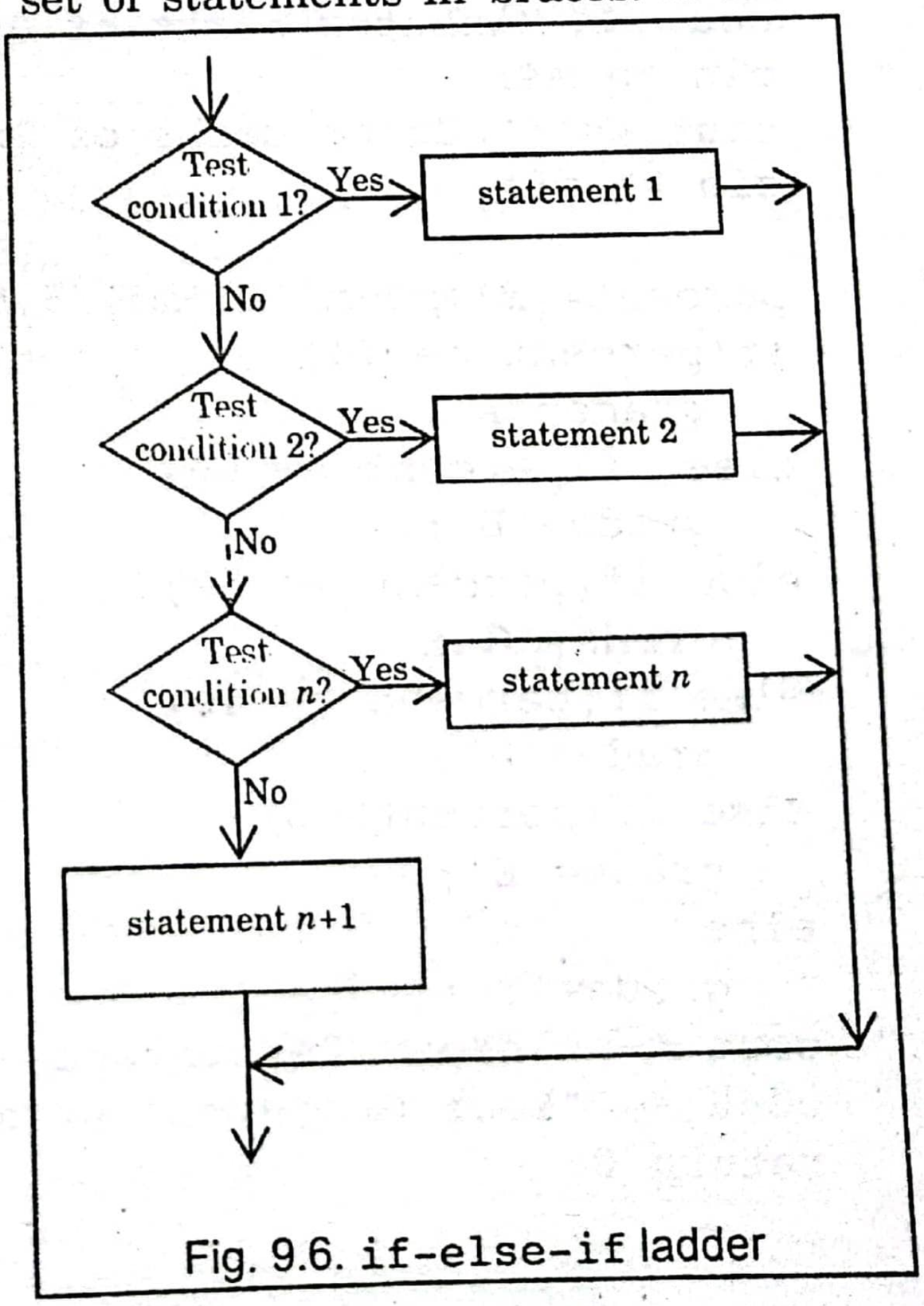


Fig. 9.6. if-else-if ladder

CamScanner

that in this structure when a true path is found, the corresponding statement or compound statement executes and the program skips over the rest of the structure. The final `else` although not necessary, is often included as a default case and executes only if none of the expressions is true.

Practical 2

Object

Writing a program of simple marksheet by using selection structures.

Source Code

```
/* program 9-15.cpp: Practical 2 - Writing a program of simple
marksheet by using selection structures. */
#include <iostream>
using namespace std;
int main()
{
  int m1, m2, m3, m4, m5;
  float percent;
  char grade;

 cout << "\nEnter marks of Physics: ";
 cin >> m1;
 cout << "\nEnter marks of Mathematics: ";
 cin >> m2;
 cout << "\nEnter marks of Computer Science: ";
 cin >> m3;
 cout << "\nEnter marks of Urdu: ";
 cin >> m4;
 cout << "\nEnter marks of English: ";
 cin >> m5;

percent=(m1+m2+m3+m4+m5)/5.0;
if(percent >= 70)
   grade='A';
else if(percent >= 60)
   grade='B';
else if(percent >= 50)
   grade='C';
   else if(percent >= 40)
      grade='D';
   else if(percent>=33)
      grade='E';
   else
      grade='F';
   cout << "\nYour Percentage = " << percent << endl;
   cout << "Your Grade = " << grade << endl;
   return 0;
}
```

CamScanner

Output

```
Enter marks of Physics: 87
Enter marks of Mathematics: 92
Enter marks of Computer Science: 63
Enter marks of Urdu: 56
Enter marks of English: 47
Your Percentage = 69
Your Grade = B
```

Program 9.16

```
// program 9-16.cpp
#include <iostream>
using namespace std;
int main()
{
    float a, b;
    char ch;
    cout << "Do you want to:" << endl;
    cout << "add, subtract, multiply, or divide?" << endl << endl;
    cout << "Enter first letter: ";
    cin >> ch;
    cout << endl;
    cout << "Enter first number: ";
    cin >> a;
    cout << "Enter second number: ";
    cin >> b;

    if (ch == 'a')
        cout << a+b;
    else if (ch == 's')
        cout << a-b;
    else if (ch == 'm')
        cout << a*b;
    else if (ch == 'd' && b != 0)
        cout << a/b;
    return 0;
}
```

In this program once a match is found, any remaining `if` statements are skipped. This means that the program is not wasting time on needless operations.

Practical 3

A prime number is a natural number greater than 1 that has exactly two distinct divisors, 1 and itself. In other words, prime numbers can't be divided by other numbers than itself or 1. For example, 2, 3, 5, 7, 11 are prime numbers. A composite number is a natural number greater than 1 that is not prime. In other words, it has more than two factors. For example, 4, 6, 8, 9 are composite numbers.

CamScanner

Object

Writing a program to input a number and find out whether it is a prime or composite.

Source Code

```
/* program 9-17.cpp: Practical 3 - Writing a program to input a
number and find out whether it is a prime or composite. */
#include <iostream>
using namespace std;
int main()
{
  int i, n;
  bool is_prime = true;
  cout << "Enter a positive number to check Prime or composite: ";
  cin >> n;

  // 0 and 1 are not prime numbers
  if (n == 0 || n == 1)
  {
    is_prime = false;
  }

  // loop to check if n is prime
  for (i = 2; i <= n/2; ++i)
  {
    if (n%i == 0)
    {
      is_prime = false;
      break;
    }
  }
  if (is_prime)
    cout << n << " is a Prime number.";
  else
    cout << n << " is a Composite number.";'
  return 0;
}
```

Output

```
Enter a positive number to check Prime or composite: 37
37 is a Prime number.
```

This program takes a positive integer from the user and stores it in the variable `n`. Notice that the Boolean variable `is_prime` is initialized to `true` at the beginning of the program.

Since 0 and 1 are not prime numbers, we first check if the input number is one of those numbers or not. If the input number is either 0 or 1, then the value of `is_prime` is set to `false`. Else, the initial value of `is_prime` is left unchanged. Then, the `for` loop is executed, which checks whether the number entered by the user is perfectly divisible by `i` or not.

CamScanner

```
for (i = 2; i <= n/2; ++i)
 {
  if (n%i == 0)
  {
     is_prime = false;
     break;
  }
}
```

The `for` loop runs from `i == 2` to `i <= n/2` and increases the value of `i` by 1 with each iteration. The loop terminates at `i == n/2` because we cannot find any factor for `n` beyond the number `n/2`. So, any iteration beyond `n/2` is redundant.

If the number entered by the user is perfectly divisible by `i`, then `is_prime` is set to `false` and the number will not be a prime number (*i.e.*, the number is composite). But if the input number is not perfectly divisible by `i` throughout the entirety of the loop, then it means that the input number is only divisible by 1 and that number itself. So, the given number is a prime number.

The `switch` Statement

The `if-else` statement is good for choosing among few (two to three) alternatives. It quickly becomes confusing when several alternatives are needed. An alternative is the `switch` statement. The `switch` statement is used to select one of the several alternatives, such as menu options. The `switch` statement is especially useful when the selection is based on the value of a single variable (called *controlling variable*) or of a simple expression (called the *controlling expression*). The value of this variable may be of type `int` or `char`, but not of type `float`. The `switch` statement also differs from `if`, in that `switch` can only test for equality, whereas the `if` conditional expression can be of any type. The general form of the `switch` statement is

```
switch(variable)                    /* may use switch(expression) */
{
case constant 1:
   statement(s);
   break;
case constant 2:
   statement(s);
   break;
case constant 3:
   statement(s);
   break;
.
.
.
default:
   statement(s);
}
```

CamScanner

A *variable* is tested against a list of integer or character constants. When the value matches a `case` condition, the statement(s) following the case is executed. The `break` statement at the end of each `case` sends control to the end of the `switch`, so once a `case` is performed, the remaining `case` statements are ignored and the `switch` ends. The `default` statement(s) is performed if no matches are found. The `default` is optional. If all matches fail and `default` is absent, no action takes place.

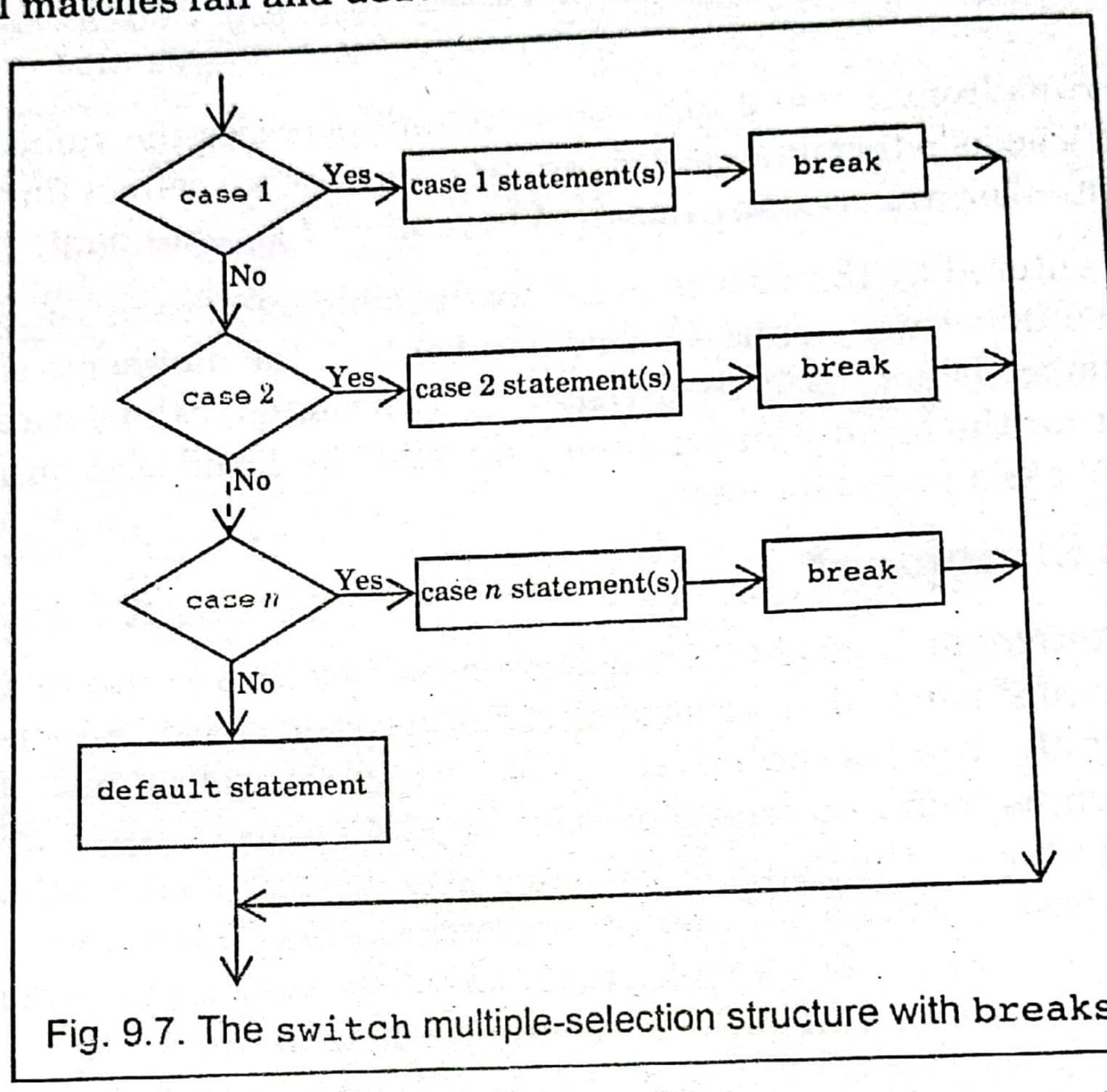


Fig. 9.7. The `switch` multiple-selection structure with `breaks`

The general `switch` multiple-selection structure (using a `break` in each `case`) is flowcharted in Fig. 9.7. The flowchart makes it clear that each `break` statement at the end of a `case` causes control to immediately exit the `switch` structure.

As a very simple example, this program recognizes the numbers 1, 2, 3, and 4 and prints the name of the one you enter. That is, if you enter 2, the program displays two.

```
// program 9-18.cpp
#include <iostream>
using namespace std;
int main()
{
  int i;
  cout << "Enter a number between 1 and 4: ";
  cin >> i;
  switch (i)
    {
    case 1:
        cout << "one";
        break;
```

CamScanner

```
    case 2:
        cout << "two";
        break;
    case 3:
        cout << "three";
        break;
    case 4:
        cout << "four";
        break;
    default:
        cout << "unrecognized number";
  }
return 0;
}
```

Now consider the following example:

```
// program 9-19.cpp
#include <iostream>
using namespace std;
int main()
{
  int num;
  cout << "Enter a Number from -1 to 4: ";
  cin >> num;
  switch (num)
    {
    case -1:
        cout << "\nInput value = -1";
        break;
   case 0:
        cout << "\nInput value = 0";
        break;
   case 1:
   case 2:
   case 3:
   case 4:
       cout << "\nInput value between 1-4";
       break;
  default:
       cout << "Input value < -1 or > 4";
    }
  return 0;
}
```

In this example there are three expected case values: −1, 0, and 1 through 4. The absence of a `break` statement for `case` values 1, 2, and 3 means that if any of those numbers happens to be the value of num, the program falls through to `case` value 4 and executes the statements there, ending with `break`. Any other input value is caught by the `default` label, which does not need a `break` statement if it is listed last.

CamScanner

In principle, a switch may occur within another, but in practice it is rarely done. Such statements would be called nested switch statements.

Example 9.4

The statement sequence associated with a case may be empty. This allows two or more cases to share a common statement sequence without duplication of code. For example, here is a program that categorizes letters into vowels and consonants.

```
// program 9-20.cpp
#include <iostream>
using namespace std;
int main()
{
  char ch;
  cout << "Enter the letter: ";
  ch = getchar();
  switch (ch)
    {
    case 'a':
    case 'e':
    case 'i':
    case 'o':
    case 'u':
       cout << ch << " is a vowel\n";
       break;
    default:
       cout << ch << " is a consonant";
    }
  return 0;
}
```

Practical 4

It is always possible to replace an if statement with an equivalent switch statement (and vice versa). So, we can rewrite program 9-16 by using switch statement.

Object

Writing a program of simple calculator using arithmetic operators.

Source Code

```
/* program 9-21.cpp: Practical 4 - Writing a program of simple
calculator using arithmetic operators. */
#include <iostream>
using namespace std;
int main()
{
  int c;
  float a, b;
  cout << "\t\t\tMENU";
  cout << "\n\t\t\t____\n\n";
```

CamScanner

```
    cout << "1. Add\n";
    cout << "2. Subtraction\n";
    cout << "3. Multiply\n";
    cout << "4. Divide\n";

    cout << "Enter the Menu option: ";
    cin >> c;
    cout << "Enter first number: ";
    cin >> a;
    cout << "Enter second number: ";
    cin >> b;

    switch (c)
      {
    case 1:
        cout << a << " + " << b << " = " << a+b;
        break;
    case 2:
        cout << a << " - " << b << " = " << a-b;
        break;
    case 3:
        cout << a << " * " << b << " = " << a*b;
        break;
    case 4:
        if (b != 0)
            cout << a << " / " << b << " = " << a/b;
        break;
    default:
        cout << "Invalid Menu option\n\n";
      }
    return 0;
}
```

Output

```
                    MENU

1. Add
2. Subtraction
3. Multiply
4. Divide
Enter the Menu option: 3
Enter first number: 12
Enter second number: 5
12 * 5 = 60
```

Example 9.5

The program displays one of five messages based on the value of `grade` (type `char`). If grade is `'D'`, `'d'`, or `'F'`, `'f'`, the student is put on probation. If `grade` is not listed in the case labels, displays an error message.

CamScanner

```
// program 9-22.cpp
#include <iostream>
using namespace std;
int main()
{
  char grade;
  cout << "Enter Grade of the Student: ";
  cin >> grade;
  switch (grade)
    {
    case 'A':
    case 'a':
        cout << "Excellent";
        break;
    case 'B':
    case 'b':
        cout << "Good";
        break;
    case 'C':
    case 'c':
        cout << "O.K.";
        break;
    case 'D':
    case 'd':
    case 'F':
    case 'f':
        cout << "Poor, student is on probation";
        break;
    default:
        cout << "Invalid letter grade";
    }
    return 0;
}
```

Example 9.6

```
/* program 9-23.cpp Writing a Program which uses a switch statement
and breaks the program if certain condition is observed.
Writing a Program which uses a switch, case and break statements
and interconvert the temperature scales using Menu options. */
#include <iostream>
using namespace std;
int main()
{
    int i;
    float c, f, k;

    cout << "\n\n\n";
    cout << "\t\t\tMENU FOR TEMPERATURE SCALE CONVERSION";
    cout << "\n\t\t\t_____\n\n";
```

CamScanner

```
cout << "1. From Celsius to Fahrenheit scale\n";
cout << "2. From Fahrenheit to Celsius scale\n";
cout << "3. From Celsius to Kelvin scale\n";
cout << "4. From Fahrenheit to Kelvin scale\n";
cout << "\nEnter the MENU option: ";
cin >> i;

switch (i)
{
case 1:
    cout << "Enter temperature in Celsius: ";
    cin >> c;
    f = 32.0+(9.0/5.0)*c;
    cout << "The temperature in Fahrenheit is = " << f;
    break;
case 2:
    cout << "Enter temperature in Fahrenheit: ";
    cin >> f;
    c = 5.0/9.0*(f-32.0);
    cout << "The temperature in Celsius is = " << c;
    break;
case 3:
    cout << "Enter temperature in Celsius: ";
    cin >> c;
    k = c + 273.15;
    cout << "The temperature in Kelvin is = " << k;
    break;
case 4:
    cout << "Enter temperature in Fahrenheit: ";
    cin >> f;
    k = (5.0/9.0*(f-32.0)) + 273.15;
    cout << "The temperature in Kelvin is = " << k;
    break;
default:
    cout << "Invalid MENU option\n\n";
}
return 0;
}
```

Output

MENU FOR TEMPERATURE SCALE CONVERSION

```
1. From Celsius to Fahrenheit scale
2. From Fahrenheit to Celsius scale
3. From Celsius to Kelvin scale
4. From Fahrenheit to Kelvin scale

Enter the MEMU option: 2
Enter temperature in Fahrenheit: 145.3
The temperature in Celsius is = 62.94
```

CamScanner

Exercise

9.1. Write a program that asks the user for an integer and then tells the user if that number is even or odd. (Hint: Use C++'s modulus operator %).

9.2. Write a program that inputs a number, then reports whether the number is in the range from 1 to 100.

9.3. How do you perform more than one statement when a condition is true?

9.4. The following is supposed to cause an action whenever x is greater than y.

```
if(x > y) then
cout << "x is greater than y";
```

Why is this wrong?

Ans. Unlike other languages, in C++ the `if` statement does not use the then keyword.

9.5. The following is supposed to cause an action whenever x is equal to y.

```
if(x = y)
cout << "x equals y";
```

Is there anything wrong with this?

Ans. This is a common mistake in C++. The expression mistakenly uses the assignment operator = instead of a relational operator ==. The effect is to assign the value of y to x. The expression then takes on the value of the assignment. If y happens to be 0, the assignment makes x equal 0. Because 0 is interpreted as *false*, the `cout` statement cannot execute even though x does in fact equal y!

9.6. Write a program that demonstrates multiple statements following `if`.

Ans.
```
// program 9-24.cpp
#include <iostream>
using namespace std;
int main()
{
  char ch;
  ch = getchar();
  if (ch == 'z')
    {
      cout << "\n\nYou typed z.";
      cout << "\nNot some other letter.";
    }
  return 0;
}
```

9.7. What is the propose of the `else` command?

9.8. Write a program that requests two numbers and then displays either their sum or product, depending upon what the user selects.

9.9. Explain how an `if` command can be used to test for valid input.

9.10. Explain the use of nested `if` statements.

9.11. Write a program that reads temperature and prints a message as given below:

CamScanner

Temperature	*Message*
> 35	Its hot day!
≥25 and ≤ 35	Its nice day!
< 25	Its cold day!

9.12. Write a program that takes two numbers and prints the greater number.

Ans.
```
// program 9-25.cpp
#include <iostream>
using namespace std;
int main()
{
  int num1, num2;

  cout << "Enter the first number: ";
  cin >> num1;

  cout << "Enter the second number: ";
  cin >> num2;

  if (num1 > num2)
  {
    cout << "The greater number is: " << num1 << endl;
  }
  else if (num2 > num1)
  {
    cout << "The greater number is: " << num2 << endl;
  }
  else
  {
    cout << "Both numbers are equal." << endl;
  }
  return 0;
}
```

9.13. Write a program that reads three numbers and prints the largest.

9.14. What is the difference between the `if-else-if` ladder and the series of one-way `if` statements?

Ans. The one-way `if` series is not as efficient as the ladder and does not show mutual exclusion because more than one statement can execute. In fact, all the expressions could be true, so each `if` statement has to be checked to determine its logical value.

9.15. Correct the syntax errors.

```
if x > 25.0
{
    y = x
else
    y = z;
}
```

CamScanner

9.16. What value is assigned to `fee` by the `if` statement when `speed` is 75?

```
if(speed > 35)
   fee = 20.0;
else if(speed > 50)
   fee = 40.0;
if(speed > 75)
   fee = 60.0;
```

9.17. Explain the difference between the statements on the left and the statements on the right. For each group of statements, give the final value of `x` if the initial value of `x` is 1.

```
if(x >= 0)
   x = x + 1;
else if(x >= 1)
   x = x + 2;
```

```
if(x >= 0)
   x = x + 1;
if(x >= 1)
   x = x + 2;
```

9.18. The ASCII codes for the lowercase letters are separated from the uppercase letters by a difference of 32. Therefore, to convert a lowercase letter to an uppercase one, simply subtract 32 from it. Write a program that reads characters from the keyboard and displays lowercase letters as uppercase ones. Stop when Enter is pressed.

9.19. Using a nested `if` statement, write a program that prompts the user for a number and then reports if the number is positive, zero, or negative.

9.20. How does the `switch` command differ from the `if` command?

9.21. What is wrong with this segment of program?

```
float f;
cin >> f;
switch(f)  {
case 12.05:
.
.
```

9.22. Write a `switch` statement that outputs messages indicating what day has been numerically input (for example, 1 → Monday, 2 → Tuesday ...).

9.23. What output line(s) are displayed by the statements that follow when `grade` is `'I'`? When `grade` is `'B'`? When `grade` is `'b'`?

```
switch(grade)
   {
   case 'A':
      points = 4;
      break;
   case 'B':
      points = 3;
      break;
   case 'C':
      points = 2;
      break;
   case 'D':
      points = 1;
```

CamScanner

```
        break;
    case 'E':
    case 'I':
    case 'Y':
        points = 0;
    }
    if(points > 0)
        cout << "Passed, points earned = " << points;
    else
        cout << "Failed, no points earned\n";
```

9.24. What will be printed by this carelessly constructed `switch` statement if the value of `color` is `'R'`?

```
switch(color)          /* break statements missing */
    {
    case 'R':
        cout << "red\n";
    case 'B':
        cout << "blue\n";
    case 'Y':
        cout << "yellow\n";
    }
```

9.25. Write a program that computes the area of either a circle, rectangle, or triangle. Use an `if-else-if` ladder. Rewrite this program using `switch` statement.

9.26. Improve the program 9-21 by using an `toupper()` or `tolower()` function.

Objectives and MCQs

Choose the answer that best completes the statement.

9.1. Structured programming
- (a) is hard to follow but worthwhile
- (b) calls for *one way in, one way out* structures
- (c) makes debugging difficult
- (d) relies heavily on `if-else` statements

9.2. Which of these expressions are true
- (a) 0
- (b) 1
- (c) 5*3 < 15
- (d) 1 == 1
- (e) −1

CamScanner

9.3. The statement `if(x = 0) cout << "X = 0";`

(a) is incorrect syntax
(b) is correct syntax, but `X = 0` will never print
(c) causes a run-time error
(d) has no effect on the program

9.4. In a simple `if` statement with no `else`, what happens if the condition following the `if` is false?

(a) the program searches for the last `else` in the program
(b) nothing, or control falls through to the statements following the `if`
(c) the body of the `if` statement is executed
(d) the program as a whole is executed

9.5. The statements following a particular `if-else` in an `if-else-if` ladder are executed when

(a) the conditional expression following the `if-else` is true and all previous conditions are true
(b) the conditional expression following the `if-else` is true and all previous conditions are false
(c) the conditional expression following the `if-else` is false and all previous conditions are true
(d) the conditional expression following the `if-else` is false and all previous conditions are false

9.6. Which `if` in a program does an `else` pair up with?

(a) the last `if` with the same indentation as the `else`
(b) the last `if` not matched with its own `else`
(c) the last `if` not enclosed in braces
(d) the last `if` not enclosed in braces and not matched with its own `else`

9.7. The advantage of a `switch` statement over an `if-else-if` construction is

(a) a default condition can be used in the `switch`
(b) the `switch` is easier to understand
(c) several different statements can be executed for each case in a `switch`
(d) several different conditions can cause one set of statements to be executed in a `switch`

9.8. The `switch` statement

(a) can always be used as a substitute for the ladder structure
(b) can be used only with an `if-else` statement
(c) is most useful when testing a variable having a limited range of possibilities
(d) all of the above

CamScanner

Objectives

9.1. Structured programming is a standardized approach that creates a program as a series of interrelated sub-programs, or functions.

9.2. An `if` condition can be written as deeply as you need within the body of another `if` statement. This is called nesting of `if`'s.

9.3. The `if` selection structure performs an indicated action only when the condition is true, otherwise the action is skipped.

9.4. The `if-else` selection structure performs certain action when the condition is true and some different action when the condition is false.

9.5. The `if-else` statement behaves like the logical operator XOR.

9.6. Nested `if-else` statements test for multiple conditions by placing `if-else` statements inside `if-else` statements.

9.7. An expression containing the OR operator is true if either or both of its operands is true.

9.8. When evaluating a ladder structure, the program stops testing after the first true expression is found.

9.9. The ladder structure facilitates testing by mutual exclusion.

9.10. The switch statement can sometimes be used as an alternative to the ladder structure.

9.11. The `switch` statement also differs from `if`, in that `switch` can only test for equality, whereas the `if` conditional expression can be of any type.

9.12. The `switch` selection structure is used to test a particular variable or expression for each of the constant integral values it may assume.

9.13. A `switch` may occur within another, such statements are called `nested switch` statements.

9.14. The default case is optional in the switch selection structure.

9.15. The `break` statement is used to exit the `switch` structure.

CamScanner

CHAPTER 10

ITERATION CONTROL STRUCTURES

By: Sir Ahad

Iteration is often referred to as a loop, because the program will keep repeating the activity until the condition becomes false. Loops simplify programming because they enable the programmer to specify certain instructions only once to have the computer execute them many times. C++ provides following three types of iteration structures:

1. `for` loop
2. `while` loop
3. `do-while` loop

Any of these can repeat either an instruction, a series of instructions, or an entire program. The `while` and `do-while` loops are used for *indefinite repetitions*. That is, they do not define the number of repetitions that will occur when the loop is executed. They merely give conditions for looping to stop. The `for` loop is used for *definite repetition*. It defines the number of repetitions that will occur during execution.

The `for` Loop

The `for` loop is used to repeat a statement or a block of statements for a specified number of times. The `for` loop statement includes the initialization of the counter, the condition, and the increment. The general form of the `for` statement is illustrated in Fig. 10.1. Notice that you put the terminating semicolon after the statement, not after the `for` argument. The statement can be simple or compound enclosed in braces. The

CamScanner

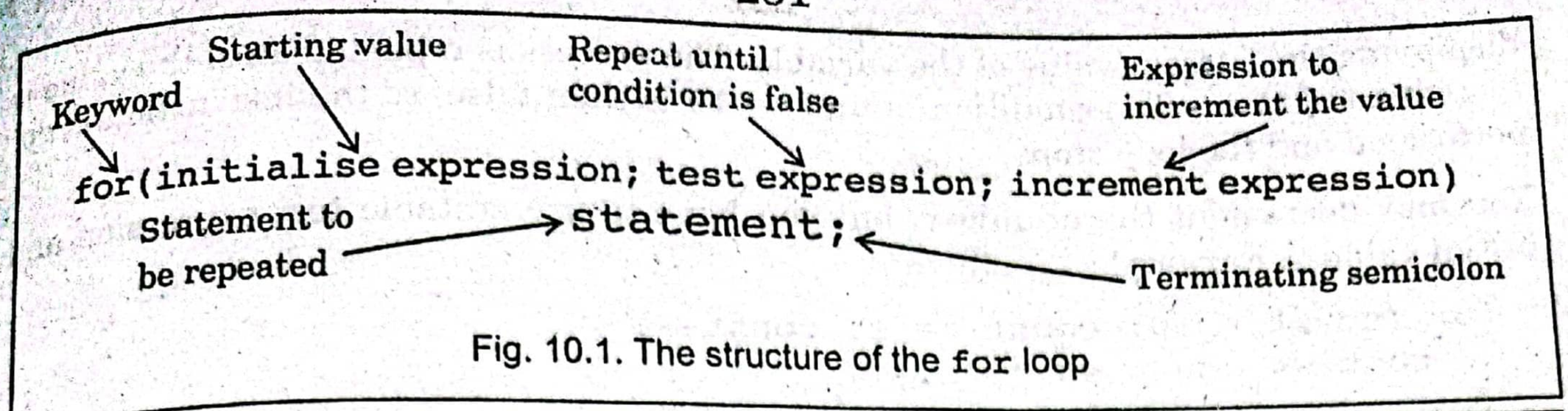


Fig. 10.1. The structure of the for loop

three expressions within the for loop statement, however, must be separated by semicolons. The for loop statement can be visualized using the flowchart as shown in Fig. 10.2.

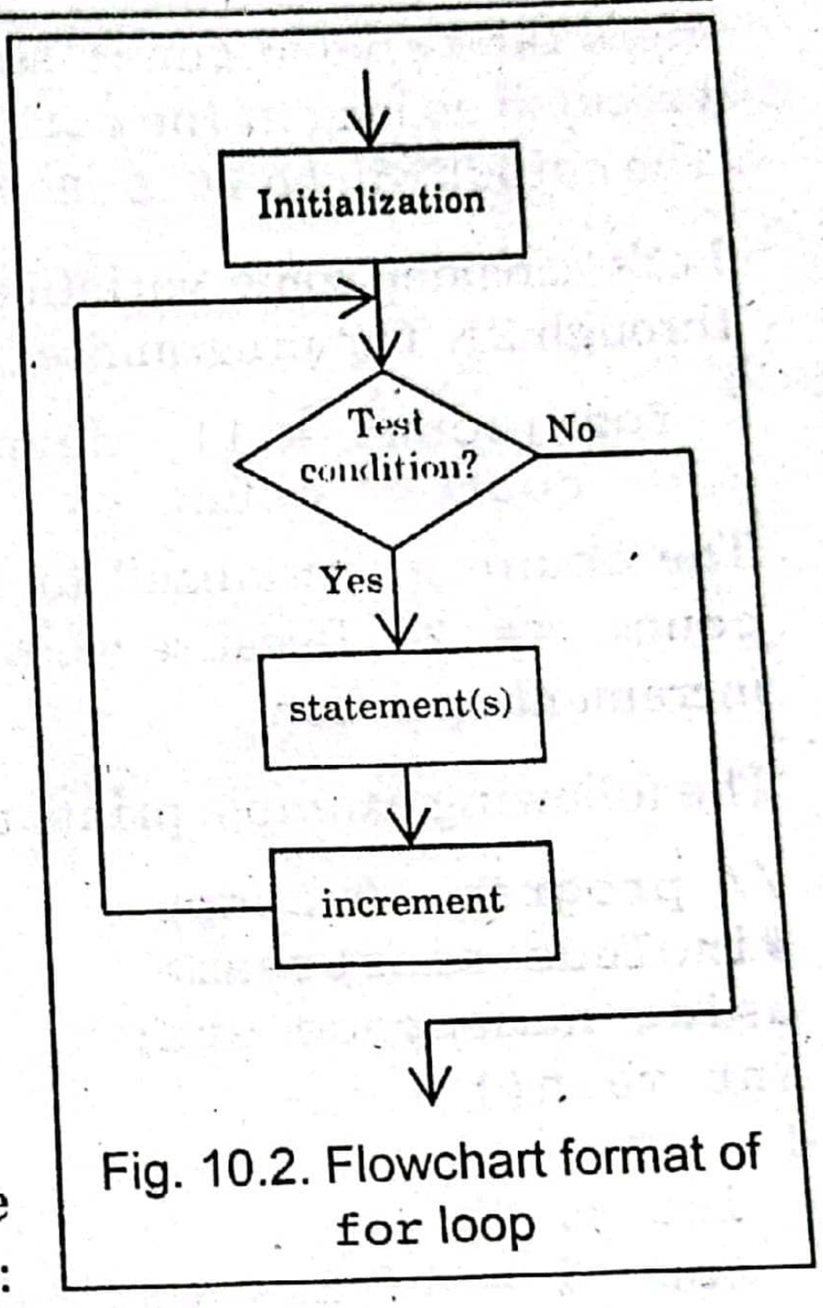


Fig. 10.2. Flowchart format of for loop

For example, this program uses a for loop to print numbers from 1 to 10.

```
// program 10-1.cpp
#include <iostream>
using namespace std;
int main()
{
  int count;
  for (count = 1; count <= 10; count++)
    cout << count << " ";
  return 0;
}
```

Let's analyze the syntax of the for loop using the output of the program which is:

```
1 2 3 4 5 6 7 8 9 10.
```

`count = 1` Initializes the starting value of the variable count. You can use any name of the variable but it must be an integer type.

`count <= 10` Sets the condition to repeat the loop as long as count has a value less than or equal to 10.

`count++` Increments the value of count after each repetition.

With each repetition of the loop, the program displays the current value of the variable count.

When this program first encounters the loop, it sets the value of count to 1. It then checks the condition to see if the value of count is less than or equal to 10. Since that condition is true, it will perform the instruction that is associated with the loop, displaying the value of the variable.

After performing the instruction, it increments the value by 1, and again checks the condition. Since the condition is still true, it performs the instruction a second time,

CamScanner

displaying the current value of the variable. The process is repeated until the value is incremented to 11. The condition count <= 10 is then false, so the instruction is not performed and the loop stops.

You may decrement the counter, but you have to use suitable test expression and initial value, for example:

```
for (count = 10; count >= 1; count--)
   cout << count << " ";
```

Here the variable count starts from 10 and is decremented by 1. The loop will be executed as long as the count is greater than or equal to 1, in other words, ten times. The output will be 10 9 8 7 6 5 4 3 2 1.

Let's consider some variations. Suppose we want to print the odd integers from 11 through 21. The program segment would look like this:

```
for (count = 11; count <= 21; count = count + 2)
   cout << count << " ";
```

The count is initialized to 11 and the expression has been changed to check for count <= 21. Because we want counting by 2, we substitute an assignment for the increment operator.

The following example prints the message "Take Care" five times on the screen:

```
// program 10-2.cpp
#include <iostream>
using namespace std;
int main()
{
  int i;
  for (i = 1; i < 6; i++)
  {
     cout << "Take Care" << endl;
  }
  return 0;
}
```

The following program continues to loop until a q is entered from the keyboard. Instead of testing the repeat condition, the test expression in this for loop checks the value of a character entered by the user.

```
// program 10-3.cpp
#include <iostream>
using namespace std;
int main()
{
  int i;
  char ch = 'a';  // declare and give ch an initial value
  for (i = 0; ch != 'q'; i++)
  {
     cout << "Pass: " << i << endl;
     cin >> ch;
```

CamScanner

```
  }
  return 0;
}
```

Here, the condition that controls the loop has nothing to do with the initialization. The reason ch is given an initial value is to prevent it from accidentally containing a q when the program begins.

Each component of the for loop control structure is optional but the semicolons are needed. For example, this program asks the user for a value and then counts down to 0 from this number. Here, the loop is initialized by the user outside the loop, so initialization portion of the loop is empty.

```
// program 10-4.cpp
#include <iostream>
using namespace std;
int main()
{
  int i;
  cout << "Enter an integer: ";
  cin >> i;

  for ( ; i; i--)
  {
     cout << "Take Care" << endl;
  }
  return 0;
}
```

Multiple Statements

To perform more than one statement in the loop, enclose the whole series in braces. For example, here is a program that converts 9 consecutive temperatures (from 32 to 40) to Celsius:

```
// program 10-5.cpp
#include <iostream>
#include <iomanip>                    // for setprecision
using namespace std;
int main()
{
  int f;
  float c;
  cout << "Fahrenheit\tCelsius\n" << endl;
  for (f = 32; f <= 40; f++)
  {
     c = 5.0/9.0*(f-32.0);
     cout << f << "\t\t" << fixed << setprecision(2) << c << endl;
  }
  return 0;
}
```

CamScanner

Two statements are performed in each repetition of the loop. The value of the initialization determines both the number of repetitions and the values to be converted to Celsius.

Nested for Loops

When one `for` command is performed within another, they are said to be nested. The inside loop is completely repeated for each repetition of the outside loop. Picture nested for loops as being two-dimensional and single `for` loops as one-dimensional.

As an example, look at the following program:

```
// program 10-6.cpp
#include <iostream>
using namespace std;
int main()
{
  int row, column;
  for (row = 1; row <= 10; row++)   // outer loop
  {
     for (column = 1; column <= 10; column++)   // inner loop
        cout << "*";

     cout << endl;       // outside of second loop, inside first
  }
  return 0;
}
```

The program displays 10 rows of 10 asterisks. It uses two integer variables, `row` and `column`. The outer loop increments the value of `row` from 1 to 10. With each repetition of the loop, however, the inner loop is performed 10 times, incrementing `column` and displaying a row of ten asterisks. So, in total, 100 asterisks are displayed—10 inner loops (column positions) for each of 10 outer loops (rows). The output is shown in Fig. 10.3.

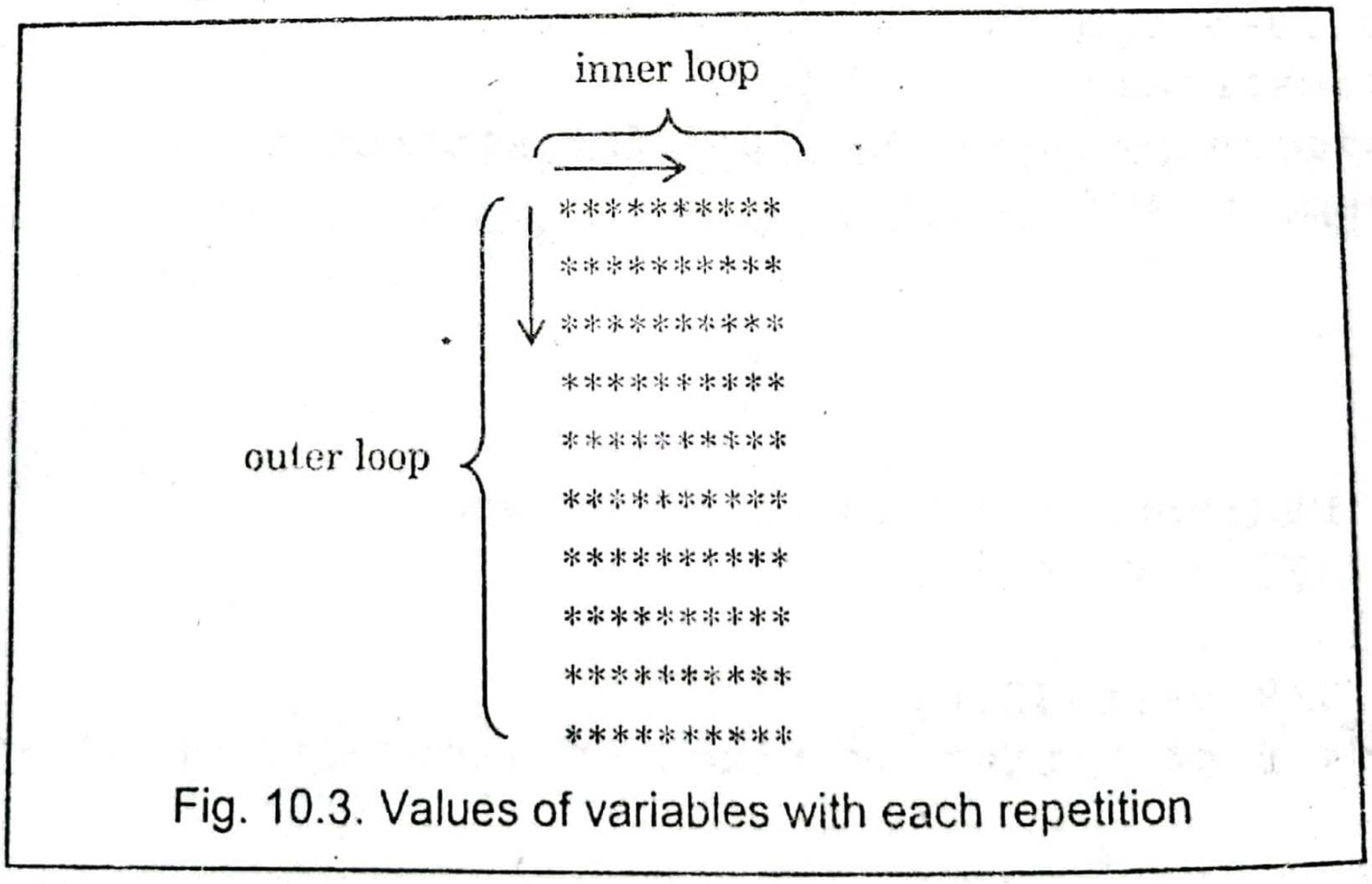


Fig. 10.3. Values of variables with each repetition

Program 10-7 illustrates another example of nested loops. It performs ten outer and ten inner loops to create a multiplication table. Instead of merely displaying an

asterisk, it displays the value of row multiplied by the value of column.

```
// program 10-7.cpp
#include <iostream>
#include <iomanip>
using namespace std;
int main()
{
  int row, column;
  cout << "\t\tMy Handy Multiplication Table\n\n";
  for (row = 1; row <= 10; row++)
  {
     for (column = 1; column <= 10; column++)
     cout << setw(6) << row*column;

     cout << endl;       // outside of second loop, inside first
  }
  return 0;
}
```

The output of the program is shown below:

```
                My Handy Multiplication Table

     1     2     3     4     5     6     7     8     9    10
     2     4     6     8    10    12    14    16    18    20
     3     6     9    12    15    18    21    24    27    30
     4     8    12    16    20    24    28    32    36    40
     5    10    15    20    25    30    35    40    45    50
     6    12    18    24    30    36    42    48    54    60
     7    14    21    28    35    42    49    56    63    70
     8    16    24    32    40    48    56    64    72    80
     9    18    27    36    45    54    63    72    81    90
    10    20    30    40    50    60    70    80    90   100
```

Finally, consider this program:

```
// program 10-8.cpp
#include <iostream>
using namespace std;
int main()
{
  int row, column;
  for (row = 1; row <= 10; row++)    // outer loop
  {
     for (column = 1; column <= row; column++)    // inner loop
     cout << "*";

  cout << endl;    // outside of second loop, inside first
  }
  return 0;
}
```

CamScanner

It displays the series of asterisks shown below:

```
*
**
***
****
*****
******
*******
********
*********
**********
```

Each row of asterisks has a different length. Instead of the inner loop always repeating the same number of times, the number increases with each repetition of the outer loop—one asterisk on the first line, two asterisks on the second line, and so on. The number of repetitions of the column is the same number as the row. We do this by using the variable row in the test expression of the inner loop.

Be cautious when writing programs that have two or more levels of for statements. A program that repeats 100 outer and 100 inner loops actually performs 10,000 repetitions!

Example 10.1

Write a program to print the sum of odd numbers from 1 to 100, *i.e.*,

sum = 1 + 3 + 5 + ... + 99

```
// program 10-9.cpp
#include <iostream>
using namespace std;
int main()
{
  int i, sum = 0;
  for (i = 1; i < 100; i += 2)
     sum = sum + i;              //can use sum += i;
  cout << "The sum is = " << sum;
  return 0;
}
```

Example 10.2

Write a program which uses for loop to print out a table of ASCII codes from 32 to 127 (compare it with program 6-1).

```
// program 10-10.cpp
#include <iostream>
using namespace std;
int main()
{
  int i;
  for (i = 32; i < 128; i++)  // code_from 32 to 127
```

CamScanner

```
    // print as number and as char
    cout << i << " = " << char(i) << "\t";
  return 0;
}
```

The output is shown below:

```
32 =    33 = !  34 = "  35 = #  36 = $  37 = %  38 = &  39 = '  40 = (  41 = )
42 = *  43 = +  44 = ,  45 = -  46 = .  47 = /  48 = 0  49 = 1  50 = 2  51 = 3.
52 = 4  53 = 5  54 = 6  55 = 7  56 = 8  57 = 9  58 = :  59 = ;  60 = <  61 = =
62 = >  63 = ?  64 = @  65 = A  66 = B  67 = C  68 = D  69 = E  70 = F  71 = G
72 = H  73 = I  74 = J  75 = K  76 = L  77 = M  78 = N  79 = O  80 = P  81 = Q
82 = R  83 = S  84 = T  85 = U  86 = V  87 = W  88 = X  89 = Y  90 = Z  91 = [
92 = \  93 = ]  94 = ^  95 = _  96 = `  97 = a  98 = b  99 = c 100 = d 101 = e
102 = f 103 = g 104 = h 105 = i 106 = j 107 = k 108 = l 109 = m 110 = n 111 = o
112 = p 113 = q 114 = r 115 = s 116 = t 117 = u 118 = v 119 = w 120 = x 121 = y
122 = z 123 = { 124 = | 125 = } 126 = ~ 127 =
```

Example 10.3

Write a program which reads a number and prints its factorial using `for` loop.

```
// program 10-11.cpp
#include <iostream>
using namespace std;
int main()
{
  double f = 1;
  int num, i;
  // computes the product n*(n-1)*(n-2)*(n-3)*...*2*1
  cout << "Enter the number to find its Factorial: ";
  cin >> num;

  if (num >= 0)
  {
     for (i=num; i>1; --i)
     f = f * i;
     cout << "The factorial of " << num << " is " << f << endl;
  }
  else
     cout << "Sorry!!" << endl;
  return 0;
}
```

The for Loop Variations

The `for` loop in C++ is significantly more powerful and flexible than in most other computer languages. The reason that `for` is so flexible is that each of the three parts inside the parentheses of the `for` statement are optional. You may omit any of them as long as your program contains the necessary statements to take care of the loop execution. For example, the loop

```
for( ; index>20; )
   statement(s);
```

CamScanner

contains only the condition for termination. The initialization is done before the loop. The incrementing must be included inside the loop block.

You can even omit all the expressions as in this example:

```
for(;;)
   cout << "Hello\n";
```

This type of loop is usually called an *infinite loop*. It is repeated continuously unless you include a suitable condition inside the loop block to terminate the execution. If there is no such condition, you can only stop the repeated loop by pressing `Ctrl-Break` to abort the program execution. Even the statement(s) inside the loop are optional. Although accidentally creating an infinite loop is a bug, you will sometimes want to create one on purpose.

Further, there does not need to be any loop-control variable because the test expression may use some other means of stopping the loop. For example, following segment of program continue to loop until a p is entered at keyboard. Instead of testing a loop-control variable, the test expression in this `for` checks the value of a character.

```
int i;
char ch;
ch = 'a';              // give ch an initial value
for(i=0; ch != 'p'; i++)
   {
   cout << "pass: " << i << endl;
   ch = getchar();
   }
```

The initialize expression and increment expression of the `for` loop can also contain more than one statement separated by a comma. For instance,

```
for(j=0, k=100; k-j>0; j++, k--)
```

In this example, both `j` and `k` are initialized before the loop is entered. After each iteration, `j` is incremented and `k` is decremented. The use of multiple statements in `for` loop expression also demonstrates why semicolons are used to separate the three expressions. If commas had been used, they could not also have been used to separate multiple statements in the initialize expression and increment expression, without confusing the compiler.

Example 10.4

Write a program which uses for loop statement, (generate the multiplication tables from 2 to 20).

```
/* program 10-12.cpp Writing a Program which uses 'for' loop
statement, (generate the multiplication tables from 2 to 20). */
#include <iostream>
using namespace std;
int main()
{
  int i, n, j=0;
```

CamScanner

```
cout << "Enter Table No.: ";
cin >> n;

if (n>=2 && n<=20)
{
 for (i = 1; i <= 10; i++)
  {
   j = n * i;
   cout << n << " * " << i << " = " << j << endl;
  }
}
else
  {
    cout << "Error!!! Enter a number from 2 to 20" << endl;
  }
 return 0;
}
```

Output

```
Enter Table No.: 7

7 * 1 = 7
7 * 2 = 14
7 * 3 = 21
7 * 4 = 28
7 * 5 = 35
7 * 6 = 42
7 * 7 = 49
7 * 8 = 56
7 * 9 = 63
7 * 10 = 70
```

Example 10.5

Here is another example of using setw( ) manipulator.

```
// program 10-13.cpp using setw() manipulator
#include<iostream>
#include<iomanip>
using namespace std;
int main()
{
  int i, num;
  cout << "Enter a number: ";
  cin >> num;
  cout << "\nThe multiplication table of " << num << " is:\n";
  for(i=1; i<=10; i++)
  cout << num << setw(3) << "*" << setw(4) << i << setw(4) << "="
<< setw(4) << num*i << "\n";
  return 0;
}
```

CamScanner

Now if we compile and execute the above code, and supply the input as a number, say 12, it produces the following result:

```
Enter a number: 12
The multiplication table of 12 is:
12   *    1    =   12
12   *    2    =   24
12   *    3    =   36
12   *    4    =   48
12   *    5    =   60
12   *    6    =   72
12   *    7    =   84
12   *    8    =   96
12   *    9    =  108
12   *   10    =  120
```

See how well-organized the output becomes after formatting it.

Practical 5

Object

Writing a program to calculate factors of a given number.

Source Code

```
/* program 10-14.cpp: Practical 5 - Writing a program to calculate
factors of a given number. */
#include <iostream>
using namespace std;
int main()
{
  int n, i;

  cout << "Enter a positive integer: ";
  cin >> n;

  cout << "Factors of " << n << " are: ";

  // Iterate from 1 to n and check for factors
 for(i = 1; i <= n; ++i)
  {
    // If i is a factor of n, print it
    if(n%i == 0)
      cout << i << " ";
  }
  return 0;
}
```

Output

```
Enter a positive integer: 70
Factors of 70 are: 1 2 5 7 10 14 35 70
```

CamScanner

In this program, an integer entered by user is stored in variable n. Then, for loop runs from 1 and checked whether n is perfectly divisible by i or not. If n is perfectly divisible by i then, i will be the factor of n. In each iteration, the value of i is updated (increased by 1).

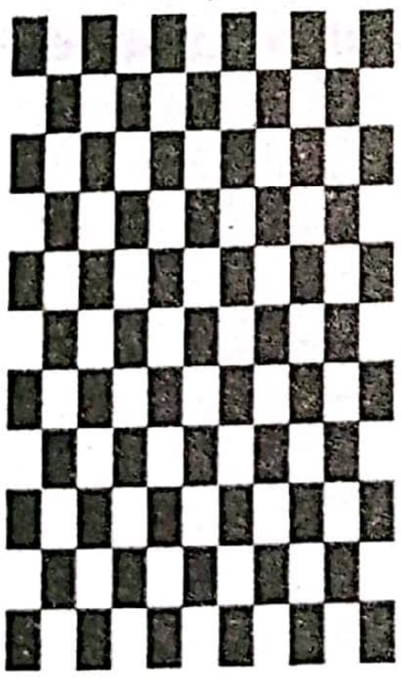
This process goes until test condition i <= n becomes false, *i.e.*, this program checks whether number entered by user n is perfectly divisible by all numbers from 1 to n and displays factors of that number.

Example 10.6

Draw a Check-board and print it using if-else statement.

```
/* program 10-15.cpp Draw a Check-board and print it using if-else
statement. */
#include <iostream>
using namespace std;
int main()
{
  int a, b;
  for (b = 1; b < 12; b++)              // stepping down
  {
   for (a = 1; a < 12; a++)             // stepping right
    {
     if ((a+b)%2 == 0)  // even numbered rectangle
       cout << "\xDB";  // print filled rectangle
     else
       cout << " ";     // print blank rectangle
    }
    cout << endl;       // new line
  }
  return 0;
}
```

Output



The `while` Loop

The `while` loop allows the programmer to specify that an action is to be repeated while some condition remains true. It is used when the exact number of loop repetitions before the loop execution begins are not known. The general form of the `while`

CamScanner

statement is:

```
while(condition)
    statement;
```

In other words, as long as the condition is true, the statement continues to execute. More specifically, the *condition* is evaluated and if it is false, the structure is skipped. If the condition is true, the statement executes. The statement can be simple or compound enclosed in braces. Once the statement executes, control is automatically returned to the condition, which is then re-evaluated. If it is false, the loop is terminated; otherwise, the entire structure is repeated. This is diagrammed in Fig. 10.4. Consider the example:

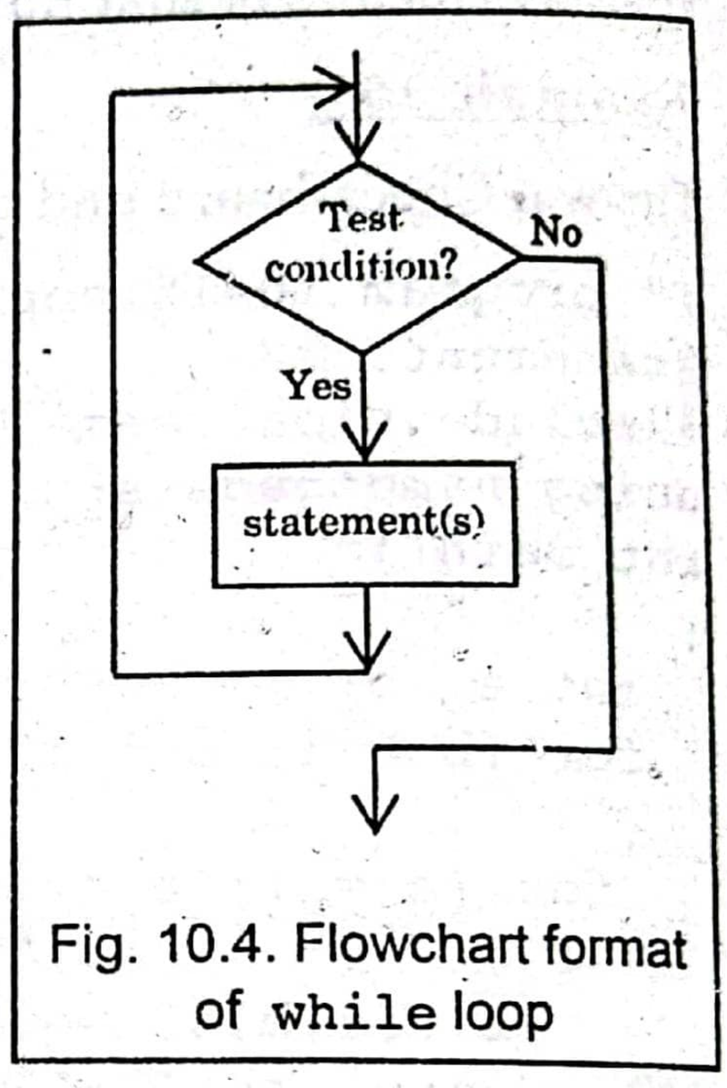


Fig. 10.4. Flowchart format of while loop

```
// program 10-16.cpp
#include <iostream>
using namespace std;
int main()
{
  char ch;
  cout << "Enter a character: ";
  cin >> ch;

 while (ch != 'q')
 {
    cout << "Input character = " << ch << endl;
    cout << "\nEnter a character: ";
    cin >> ch;
 }
 return 0;
}
```

As long as the user does not type q, this `while` loop continues to output the input character (*echo* the input) and request another input character.

As another example, this program uses a `while` loop to print numbers from 1 to 10.

```
// program 10-17.cpp
#include <iostream>
using namespace std;
int main()
{
   int count = 1;

   while (count <= 10)
   {
      cout << count << " ";
      ++count;
   }
     return 0;
}
```

CamScanner

This program can be written more concisely by modifying cout and increment statements with one as:

```
while (count <= 10)
   cout << count++ << " ";;
```

Now look at this simple example, which takes an input number and displays an equivalent number of asterisks.

```
// program 10-18.cpp
#include <iostream>
using namespace std;
int main()
{
   int num;
   cout << "Enter a number: ";
   cin >> num;

   while (num != 0)
   {
      cout << "*";
      num--;
   }
   cout << "\n";
   return 0;
}
```

To ensure proper input, place the first input command outside the loop. You can then continue asking for input as long as the value is invalid, like this:

```
// program 10-19.cpp
#include <iostream>
using namespace std;
int main()
{
  float num;
  cout << "\nEnter an input number: ";
  cin >> num;

  while (num < 0 || num >= 1)
  {
     cout << "\n\nYou got it wrong";
     cout << "\n\nEnter a number: ";
     cin >> num;
  }
  return 0;
}
```

Example 10.7

Write a program which uses `while` loop to print out a table of ASCII codes from 32 to 127 (compare it with program 10-10).

CamScanner

```
// program 10-20.cpp
#include <iostream>
using namespace std;
int main()
{
  int i;

  i = 32; // initialization to 32
  while (i < 128) // code from 32 to 127
  {
   cout<<i<< " = " <<char(i)<< '\t'; // print as number and as char
   i++;     // increment
  }
  return 0;
}
```

Example 10.8

Write a program to calculate average of a list of numbers using `while` loop.

```
// program 10-21.cpp
#include <iostream>
using namespace std;
int main()
{
  int n, count = 1;
  float x, average, sum = 0;

// Initialize and read in a value for n
cout << "How many numbers? ";
cin >> n;

// Read in the numbers
while (count <= n)
{
  cout << "X = ";
  cin >> x;
  sum += x;
  ++count;
}

  // Calculate the average and print the answer
  average = sum/n;
  cout << "\nThe average is " << average;
  return 0;
}
```

The Relationship of the `for` and `while` Loops

In many other languages the `for` loop and the `while` loop are very different constructs that are used for different purposes. In C++ the two loops are simply two ways of doing

CamScanner

the same thing. The `while` structure can be used to increment through a loop much like a `for` loop. For example, instead of

```
for(i = 1; i <= n; i++)
   cout << i;
```

you could write

```
i = 1;
while(i++ <= n)
   cout << i;
```

The decision to use a `for` loop or a `while` loop is not determined by C++ but by conventional usage, readability, and consistency.

Example 10.9

Finding the Factorial of n using `while` loop, read value of n using `cin`, and print the factorial of n.

```
/* program 10-22.cpp Finding the Factorial of n using while loop,
read value of num using cin, and print the factorial of n. */

#include <iostream>
using namespace std;
int main()
{
  double f = 1;
  int num, i;
  cout << "\n\t\tEnter the number to find its Factorial: ";
  cin >> num;
  if (num >= 0)
  {
    i = 1;
    while (i <= num)
    {
     f = f * i;
     i++;
    }
    cout << "\n\t\tThe factorial of " << num << " is " << f;
  }
  else
  {
    cout << "\n\t\tSorry!!";
  }
  return 0;
}
```

Output

```
1st Run
      Enter the number to find its Factorial: 4
      The factorial of 4 is 24
```

CamScanner

```
2nd Run
      Enter the number to find its Factorial: 75
      The factorial of 75 is 2.48091e+109
3rd Run
      Enter the number to find its Factorial: -7
      Sorry!!
4th Run
      Enter the number to find its Factorial: 0
      The factorial of 0 is 1
```

Nested while Loops

The `while` loop can be nested just like `for` loop. The following program shows a `while` loop nested within another `while` loop to read a number and print its factorial (compare this program with program 10-11).

```
// program 10-23.cpp
#include <iostream>
using namespace std;
int main()
{
  double f;
  int num = 1;

  // computes the product n*(n-1)*(n-2)*(n-3)*...*2*1
  while (num != 0)
  {
  cout << "Enter the number to find its Factorial: ";
  cin >> num;

  if (num >= 0)
  {
     f = 1;
     while (num > 1)
     {
     f = f * num;
     num--;
     }
     cout << "The factorial is " << f << endl;
}
else
     cout << "Sorry!!" << endl;
  }
  return 0;
}
```

This program uses an outer `while` loop to recycle until a `0` value is entered. The inner loop uses the decrement operator to reduce the variable `num`—which starts from the value typed in by the user—by 1 each time through the loop. When `num` reaches 1, the loop terminates.

CamScanner

The do-while Loop

One important characteristic of the while statement is that the test condition is at the top of the loop. This means that if the condition is false the *first* time, the body of while loop will never be executed. But there are certain situations where you need to execute the body at least once. These situations are not common, but when they do occur, you should use the do-while statement.

The only difference between a do-while and a while loop is that the test condition is at the bottom of the loop. This means that the program always executes the statement(s) at least once, even if the condition fails for the first time itself. Then, depending on the value of test expression, it may loop back to do or it may continue with the next statement. The general form of the do-while statement is:

```
do
   statement;
while(condition);
```

The statement will be performed once even if the condition is false when the loop first begins because the condition is not tested at the top. Then the statement is performed repeatedly as long as the condition in the while statement is true. The statement can be simple or compound enclosed in braces. The do-while loop statement can be visualized using flowchart as shown in Fig. 10.5. This flowchart makes it clear that the test condition is not executed until after the action is performed at least once.

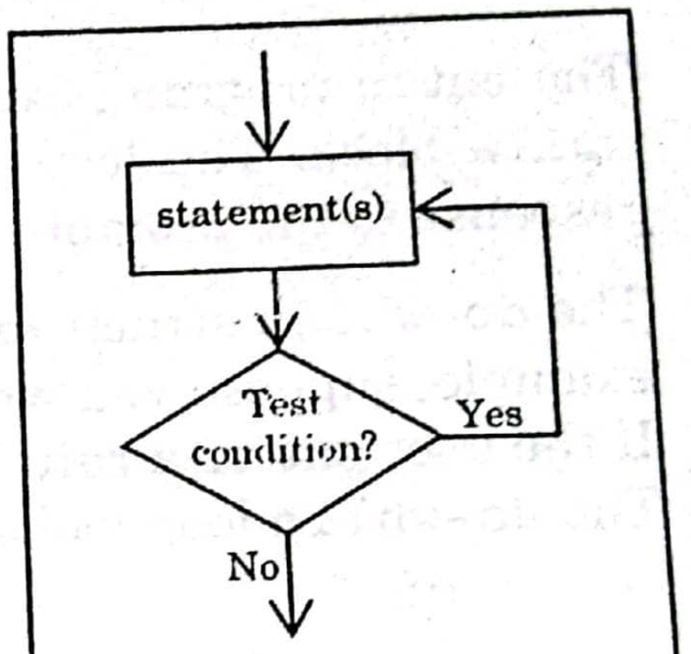


Fig. 10.5. Flowchart format of do-while loop

For example, we could print numbers from 1 to 10 using C++ code (compare it with program 10-17):

```
// program 10-24.cpp
#include <iostream>
using namespace std;
int main()
{
  int count = 1;
  do
    {
     cout << count << " ";
    }
  while (++count <= 10);
  return 0;
}
```

The do-while structure is often used to repeat a program until the user decides there is no more input:

```
// program 10-25.cpp
#include <iostream>
using namespace std;
```

CamScanner

```
int main()
{
  int num;
  char repeat;
  do
  {
  cout << "\nPlease enter a number: ";
  cin >> num;
  cout << "The square of your number is = " << num*num;
  cout << "\n\nDo you wish to continue y/n: ";
  cin >> repeat;
  } while (repeat == 'y' || repeat == 'Y');
  return 0;
}
```

The entire program, except for the variable declarations enclosed in one large do-while block. The loop continues until you enter a character other than y or Y in response to the prompt.

The do-while structure is also used to ensure that the proper input is entered. For example, suppose you write a program that inputs a discount rate in decimal format. If the user enters a rate less than 0, or 1 or above, you want to ask for another input. The do-while loop looks like this:

```
do
{
  cout << "Please enter the discount: ";
  cin >> discount;
}
while(discount < 0 || discount >= 1);
```

The while condition uses the logical OR operator to test whether the value entered is outside of the range. The loop will repeat as long as the user enters an invalid number.

Example 10.10

The do-while loop is especially useful when your program is waiting for some event to occur. For example, this program waits for the user to type a q.

```
// program 10-26.cpp
#include <iostream>
using namespace std;
int main()
{
  char ch;
  do
  {
   ch = getchar();
  }
  while (ch != 'q');
  cout << "\nFound the q";
  return 0;
}
```

CamScanner

Since the loop condition is tested at the bottom, it is not necessary to initialize ch prior to entering the loop..

Example 10.11

Modify the program 10-21 to calculate average of a list of numbers using do-while loop.

```
// program 10-27.cpp
#include <iostream>
using namespace std;
int main()
{
  int n; count = 1;
  float x, average, sum = 0;

  // initialize and read in a value for n
  cout << "How many numbers? ";
  cin >> n;

  // read in the numbers
  do
  {
   cout << "X = ";
   cin >> x;
   sum += x;
   ++count;
  }
  while (count <= n);

    // calculate the average and print the answer
    average = sum/n;
    cout << "\nThe average is " << average;
    return 0;
}
```

Nested do-while Loops

You can nest do-while loops to provide two levels of repetition. You can use an outer loop to repeat the entire program until the user decides otherwise, and inner loops to take a series of inputs, as shown in program 10-28. The inner loop is used to calculate average of a list of numbers. The outer loop is repeated as long as this condition is true:

```
while (repeat == 'y' || repeat == 'Y');
```

When you enter any other character than y or Y, the outer loop, and thus the program, ends. (Compare this program with program 10-27 and note the changed position of initialization. Why it is necessary to make this change?)

```
// program 10-28.cpp
#include <iostream>
```

CamScanner

```
using namespace std;
int main()
{
  int n, count;
  float x, average, sum;
  char repeat;

  // initialize and read in a value for n
  do
  {
   cout << "How many numbers? ";
   cin >> n;
   sum = 0;
   count = 1;

   // read in the numbers
   do
   {
    cout << "X = ";
    cin >> x;
    sum += x;
    ++count;
   } while (count <= n);

// calculate the average and print the answer
average = sum/n;
cout << "\nThe average is " << average;
cout << "\n\nDo you wish to continue y/n: ";
cin >> repeat;
cout << "\n\n";
}
while (repeat == 'y' || repeat == 'Y');
return 0;
}
```

Inner loop takes a series of inputs

Outer loop repeats the entire

Combining Loop Types

You can nest any combination of `for`, `while`, and `do-while` loops if the program logic requires it. The following program, for instance, nests a `while` loop within a `for` repetition to convert 10 temperatures, between 0 and 100, input from the keyboard:

```
// program 10-29.cpp
#include <iostream>
using namespace std;
int main()
{
  int temp, count;
  float celsius;
  for (count = 1; count <= 10; count++)
  {
```

CamScanner

```
    cout << "Enter a temperature between 1 and 100: ";
    cin >> temp;
    while (temp < 0 || temp > 100)
    {
     cout << "Invalid temperature, try again: ";
     cin >> temp;
    }
    celsius = (5.0/9.0)*(temp-32);
    cout << temp << " fahrenheit is " << celsius << " celsius\n";
  }
  return 0;
}
```

The outer `for` loop will only be repeated 10 times. The inner `while` loop will be repeated as long as you enter a valid entry.

The `break` Statement

The `break` statement allows you to exit a loop or a switch from any point within its body, bypassing its normal termination expression. It can be used within any C++ control structure *e.g.*, `for`, `while`, `do-while`, or `switch` statement.

The general form of the `break` statement is

```
loop(condition)
    {
        ...
        if(...)
        break;
        ...
    }
```

When the `break` statement is encountered inside a loop, the loop is immediately terminated, and program control resumes at the next statement following the loop. But if a `break` statement is found in nested loops, the `break` only transfers control to the outside of the loop actually containing the `break`.

Program 10-30 demonstrates the `break` statement in a `for` loop. When the `if` structure detects that `count` has become `10`, `break` is executed. This terminates the `for` statement and loop only fully executes 10 times.

```
// program 10-30.cpp
#include <iostream>
using namespace std;
int main()
{
  int count;
  for (count = 1; count <= 100; count++)
  {
   cout << count << " ";
   if (count == 10)
     break;          // exit the loop
```

CamScanner

```
  }
  return 0;
}
```

The break statement is commonly used in loops in which a special condition can cause immediate termination. This is an example of such a situation, where a key press can stop the execution of the program.

```
// program 10-31.cpp
#include <iostream>
using namespace std;
int main()
{
  int i;
  char ch;

  // display all numbers which are multiples of 6
  for (i = 1; i < 10000; i++)
  {
   if (!(i%6))
   {
    cout << i << " more (y/n) ";
    cin >> ch;
    if (ch == 'n')
      break;
    cout << endl;
   }
  }
  return 0;
}
```

Example 10.12

A break will cause an exit from only the innermost loop. For example, this program prints the numbers 1 to 5 five times.

```
// program 10-32.cpp
#include <iostream>
using namespace std;
int main()
{
  int i, j;
for(i = 0; i < 5; i++)
{
 for(j = 0; j < 100; j++)
  {
   cout << j << " ";
   if(j == 5)
   break;
  }
   cout << endl;
```

CamScanner

```
    }
    return 0;
}
```

The continue Statement

The continue statement is just the opposite of the break statement. It takes the control to the beginning of the loop, bypassing the statements inside the loop which have not yet been executed.

The general form of the continue statement is

```
loop(condition)
    {
    …
    if(…)
    continue;
    …
}
```

In while and do-while structure, the loop-continuation test is evaluated immediately after the continue statement is executed. In the for structure, the increment expression is executed, then the loop-continuation test is evaluated. The continue statement is not used with the switch statement.

Program 10-33 uses the continue statement in a for structure to skip the cout statement in the structure if count is equal to 5 and begin the next iteration of the loop.

```
// program 10-33.cpp
#include <iostream>
using namespace std;
int main()
{
  for(int count = 1; count <= 10; count++)
   {
    if(count == 5)
     continue;     // skip remaining code in loop if count==5
    cout << count << " ";
   }
  return 0;
}
```

The output of the program will be

```
1  2  3  4  6  7  8  9 10
```

The program used continue to skip printing the value 5.

CamScanner

Example 10.13

Write a program to calculate average marks of a student using while and nested while loop.

```
// program 10-34.cpp
#include <iostream>
using namespace std;
int main()
{
  int n, count = 1;
  float marks, average, sum = 0;

  // initialise and read in a value for n
  cout << "How many Subjects? ";
  cin >> n;
  cout << endl;

  // read marks
  while (count <= n)
   {
    cout << "Marks in Subject " << count << " = ";
    cin >> marks;

    while (marks < 0 || marks > 100)
    {
     cout << "Invalid marks, try again: ";
     cin >> marks;
    }
    sum += marks;
    ++count;
   }

    // calculate the average and print the answer
  average = sum/n;
  cout << "\nThe average marks are = " << average << endl;
  return 0;
}
```

Output

```
How many Subjects? 5
Marks in Subject 1 = 69
Marks in Subject 2 = 58.5
Marks in Subject 3 = 37
Marks in Subject 4 = 83
Marks in Subject 5 = 65

The average marks are = 62.5
```

CamScanner

Example 10.14

Write a program to calculate average marks of a student using for and nested while loop.

```
// program 10-35.cpp
#include <iostream>
using namespace std;
int main()
{
  int n, count;
  float mark, average, sum = 0;

  // initialise and read in a value for n
  cout << "How many Subjects? ";
  cin >> n;
  cout << endl;

  // read marks
  for (count = 1; count <= n; count++)
   {
    cout << "Marks in Subject " << count << " = ";
    cin >> marks;
    while (marks < 0 || marks > 100)
     {
      cout << "Invalid marks, try again: ";
      cin >> marks;
     }
    sum += marks;
   }

  // calculate the average and print the answer
  average = sum/n;
  cout << "\nThe average marks are = " << average << endl;
  return 0;
}
```

Output

```
How many Subjects? 5

Marks in Subject 1 = 69
Marks in Subject 2 = 58.5
Marks in Subject 3 = 37
Marks in Subject 4 = 83
Marks in Subject 5 = 65

The average marks are = 62.5
```

CamScanner

Exercise

10.1. What is meant by looping?

10.2. What is the purpose of the `for` statement? How does it differ from the `while` statement and the `do-while` statement?

10.3. How many times will a `for` loop be executed? Compare with the `while` loop and the `do-while` loop.

10.4. How would you use a compound assignment operator to count by 2?

Ans.
```
for(count = 11; count <= 21; count += 2)
```

10.5. Write a `for` loop that counts backwards from 100 to 0 by 10.

Ans.
```
for(count = 100; count >= 0; count -= 10)
```

10.6. Find the error in the following statement (Note: there may be more than one error).

```
for(i = 100; i >= 1; i++)
```

10.7. What are the functions performed in a `for` argument?

10.8. What stops a `for` loop?

10.9. What is a nested loop?

10.10. Write a program that displays the multiplication table for numbers 0 to 9.

10.11. What is displayed by the following program segment, assuming `m` is 3 and `n` is 5?

```
for(i = n; i > 0; --i)
   {
      for(j = m; j > 0; --j)
      {
        cout << "*";
      }
      cout << "\n";
}
```

10.12. Write a program that generates the following output: (use any loop)

```
2       4
3       6
4       8
5       10
```

Ans.
```
// program 10-36.cpp
#include <iostream>
using namespace std;
int main()
{
  int num = 2, i;
  for (i=2; i<=5; i++)
  {
  cout << i << "\t" << i*num << endl;
  }
  return 0;
}
```

CamScanner

10.13. Write a program that prints the following pattern, use nested for loop.

```
+++++
++++
+++
++
+
```

Ans.
```
// program 10-37.cpp
#include <iostream>
using namespace std;
int main()
{
  for (int row = 5; row >= 1; row--)
  {
    for (int column = 1; column <= row; column++)
    {
      cout << "+";
    }
    cout << endl;
  }
  return 0;
}
```

10.14. Write nests of loops that cause the following output to be displayed:

```
* * * * *
* * * *
* * *
* *
*
* *
* * *
* * * *
* * * * *
```

10.15. Write output of the given code.

10.16.
```
// program 10-38.cpp
#include <iostream>
using namespace std;
int main()
{
  for (int i = 1; i <= 5; i++)
  {
    for (int j = 1; j <= i; j++)
      cout << j * 2;
    cout << endl;
  }
  return 0;
}
```

CamScanner

Ans. 2
24
246
2468
246810

10.17. During execution of the following program segment, how many lines of asterisks are displayed?

```
for(i = 0; i < 10; ++i)
   for(j = 0; j < 5; ++j)
      cout<< "**********\n";
```

Ans. 50.

10.18. Explain what is wrong with this program:

```
#include <iostream>
using namespace std;
int main()
{
  float row, column;
  cout << "\t\tMy Handy Multiplication Table\n\n";
  for (row = 1; row <= 10; row++)
  {
    for (column = 1; column <= 10; column+)
    {
     cout << row * column << "\t";
    }
    cout << "\n";
  }
  return 0;
}
```

10.19. What does the following program output?

```
// program 10-39.cpp
#include <iostream>
using namespace std;
int main()
{
  int i, j, k;
  k = 0;
  for (i = 1; i <= 30; i++)
  {
   for (j = 1; j <= 20; j++)
   {
    if (j%3 == 0)
     k++;
   }
  }
  cout << "\nk = " << k;
  return 0;
}
```

CamScanner

10.20. What will be the output of the following programs:

(a)
```
int main()
{
  char i = 1;
  while (i <= 212)
   {
    cout << i << endl;
    i = i + 1;
   }
  return 0;
}
```

(b)
```
int main()
{
  int i = 1;
  while (i <= 212)
    cout << ++i << endl;
  return 0;
}
```

(c)
```
int main()
{
  int i;
  for (i = 1; i <= 212; i++)
    cout << i;
  return 0;
}
```

10.21. Write a program that inputs ten numbers between 0 and 25.

10.22. Write a program to print the sum of even numbers from 1 to 100.

10.23. Write a program to print the squares of all the numbers from 1 to 20.

10.24. Write a program to print prime numbers up to a specified range using `for` and nested `for` loop.

Ans. We can test whether a number is prime or not, by dividing it successively by all numbers from 2 to one less than itself. If remainder of any of these divisions is zero, the number is not a prime.

Following is the program to do this

```
// program 10-40.cpp
#include <iostream>
using namespace std;
int main()
{
int i, j, num;

cout << "Enter the number: ";
cin >> num;
for (i = 2; i <= num; i++)
{
  for (j = 2; j < i; j++)
  {
```

CamScanner

```
        if (i%j==0)
          break;
      }
      if (i==j)
        cout << i << "\t";
    }
    return 0;
}
```

10.25. What is the purpose of the `while` statement? When is the logical expression evaluated? What is the minimum number of times that a `while` loop can be executed?

10.26. How is the execution of a `while` loop terminated?

10.27. When are a `for` loop and a `while` loop the same?

10.28. What is the purpose of the `do-while` statement? How does it differ from the `while` statement?

10.29. What is the minimum number of times that a `do-while` loop can be executed? Compare with a `while` loop and explain the reasons for the differences.

10.30. What is the value of `i` after each of these segments?

(a)
```
while(i++ <= 10);
```

(b)
```
while(++i <= 10);
```

(c)
```
i = 0;
do
  ++i;
while(i++ < 10);
```

10.31. What criteria do you use to select a `for`, `while` or `do-while` loop?

10.32. State all the loop control structures supported by C++ and compare them.

10.33. What does the following code segment display? Try each of these inputs: 345, 82, 6. Then, describe the action of the code.

```
cout << "Enter a positive integer: ";
cin >> num;
do
{
  cout << num%10;
  num /= 10;
}
while (num > 0);
cout << endl;
```

Ans.
```
Enter a positive integer: 345
543
Enter a positive integer: 82
28
Enter a positive integer: 6
6
```

The code displays the digits of an integer in reverse order.

10.34. Write a loop that will calculate the sum of every third integer, beginning with

CamScanner

`i` = 2 (*i.e.*, calculate the sum 2 + 5 + 8 + 11 + ...) for all values of `i` that are less than 100. Write the loop in three different ways.

(a) Using a `for` statement.

(b) Using a `while` statement.

(c) Using a `do-while` statement.

10.35. What is the purpose of the `break` command?

10.36. State the use of the following giving an example of each.

(a) break

(b) continue

Objectives and MCQs

Choose the answer that best completes the statement.

10.1. The main difference in operation between an `if` statement and a `while` statement is

(a) the conditional expression following the keyword is evaluated differently

(b) the `while` loop body is always executed, the `if` loop body only if the condition is true

(c) the body of the `while` statement may be executed many times, the body of the `if` statement only once

(d) the conditional expression is evaluated before the `while` loop body is executed but after the `if` loop body

10.2. Assuming `x` does not equal 0, the statement `while(x==0) cout<<"X=0"`

(a) is an infinite loop

(b) will never output

(c) causes a syntax error

(d) will output once

10.3. Assuming `x` does not equal 0, the statement `do cout<<"X=0" while(x==0)`

(a) is an infinite loop

(b) will never output

(c) causes a syntax error

(d) will output once

CamScanner

10.4. A single-statement `for` loop is terminated with a
- (a) right bracket
- (b) right brace
- (c) comma
- (d) semicolon

10.5. The statement `for(x = 0; x < 1; x++) cout<<"X=0"`
- (a) is an infinite loop
- (b) will never output
- (c) causes a syntax error
- (d) will output once

10.6. The statement `for(x = 0; x <= -1; x++) cout<<"X=0"`
- (a) is an infinite loop
- (b) will never output
- (c) causes a syntax error
- (d) will output once

10.7. A `while` loop is more appropriate than a `for` loop when
- (a) the terminating condition occurs unexpectedly
- (b) the body of the loop will be executed at least once
- (c) the program will be executed at least once
- (d) the number of times the loop will be executed is known before the loop is executed

10.8. A `do-while` loop is useful when
- (a) the body of the loop will never be executed
- (b) the body of the loop will be executed at least once
- (c) the body of the loop may never be executed
- (d) none of the above

10.9. The advantage of putting complete assignment statements inside loop expressions is
- (a) to avoid awkward program constructions
- (b) to simplify the flow of control in the program
- (c) to reduce the number of program statements
- (d) to clarify the operation of the program

10.10. The `break` statement is used to exit from which part of a loop?
- (a) beginning
- (b) middle
- (c) end
- (d) none of the above

CamScanner

Objectives

10.1. The `for` loop is generally used with a counter.

10.2. The `for` loop counter variables should not be of the `float` type or of the `char` type. They should only be of the `int` type.

10.3. The three parts of the `for` loop expression are (i)the initialize expression, (ii)the test expression, and the (iii)increment expression.

10.4. A semicolon is used to separate the three parts of the loop expression in a `for` loop.

10.5. The `while` and `do-while` loops are used for indefinite repetitions.

10.6. The `for` loop is used for definite repetition.

10.7. The `do-while` loop uses post testing.

10.8. A program that repeats 5 outer and 10 inner loops actually performs 50 repetitions.

10.9. The `continue` statement, when executed in a repetition structure causes the next iteration of the loop to be performed immediately.

10.10. The `break` statement, when executed in a repetition structure or a `switch` causes immediate exit from the structure.

CamScanner

CHAPTER 11

FUNCTIONS

By: Sir Ahad

One of the most fundamental ideas in problem solving is dividing a problem into sub-problems and solving each sub-problem independently. C++ facilitates this approach through the use of functions.

A function is a block of instructions that together perform a specific task. Every C++ program has at least one function, `main()`. You can also write your own functions and use them just as you use functions in a C++ library.

When your program starts, `main()` is called automatically and it is always executed first. `main()` might called other functions, some of which might called still others.

Advantages of Functions

The following are some advantages of using functions.

1. The complexity of the entire program can be divided into simple subtasks and function subprograms can be written for each subtask.
2. Functions help us to avoid unnecessary repetition of code. Using functions, a single section of code can be used many times in the same program.
3. Functions can have inputs and outputs and can process information.
4. The functions or subprograms are short, easier to write, understand and debug.
5. A function can be shared by other programs by compiling it separately and loading them together.
6. Functions are mostly independent of each other, just as one program is independent of another program. Most of the activities in a function are hidden from the rest of the program.
7. A function can call itself again and again. It is called *recursiveness*. Many calculations can be done easily using recursive process such as calculation of factorial of a number etc.

CamScanner

Classification of Functions

Functions are of two types. These are:

1. Built-in Functions/Library Functions
2. User-Defined Functions

Built-in Functions

Many of the operations, like taking the square root of a number, sine value of a number etc., will be frequently used by many programmers in their programs. Writing a program to find the square root of a number or sine value of a number every time in each program is an unnecessary and lengthy job. Such operations are programmed and stored in C++ library, so that they can be called through any program in the form of functions. These functions are called *library functions* or *built-in functions*. Table 7.5 shows some of the commonly used mathematical library functions defined in the <cmath> header.

User-Defined Functions

Apart from the library functions that are built in C++, users can also define functions to do a task relevant to their programs. Such functions are called *user-defined functions*. These functions should be codified by the user, so that any call to the function can refer to it.

General Form of Function

The overall structure of the function looks almost exactly like that of the `main()` function in a program. That is because `main()` is also a function. The only difference is that `main` is a special name. Use of the function in a program requires three things:

1. Function declaration, which is also called function prototype.
2. Function definition.
3. Function calling.

The general form of a C++ function definition is as follows:

```
return_type function_name(parameter list)
{
  body of the function
}
```

A C++ function definition consists of a function header and a function body.

Here are all the parts of a function:

Return Type: A function may return a value. The `return_type` is the data type of the value the function returns. Some functions perform the desired operations without returning a value. In this case, the `return_type` is the keyword `void`.

Function Name: This is the actual name of the function. The function name and the parameter list together constitute the function signature.

Parameters: A parameter is like a placeholder. When a function is invoked, you pass a value to the parameter. This value is referred to as actual parameter or argument.

CamScanner

The parameter list refers to the type, order, and number of the parameters of a function. Parameters are optional; that is, a function may contain no parameters.

Function Body: The function body contains a collection of statements that define what the function does.

Program 11-1 shows the form of a user-defined `display()` function.

```
// program 11-1.cpp
#include <iostream>
using namespace std;

int y = 4;
void display(void);
int main()
{
  display();
  cout << "\nThe capital city is ISLAMABAD\n";
  display();
  cout << "\nThe color of its flag is Green and White";
  cout << "\nPakistan has " << y << " provinces";
  return 0;
}

void display(void)
{
  int x;
  for (x = 0; x < y; x++)
    cout << "*";

  cout << "\n*****PAKISTAN*****\n";

  for (x = 0; x < y; x++)
    cout << "*";
}
```

global variable

function declaration or function prototype

function calling

return data type

function name

function header

note: no semicolon at the end

accept data type

local variable

body of function

function definition for display()

The declaration tells the compiler that at some later point we plan to present a function called `display()`. The `void` in the beginning of function header means that `display()` does not return anything. Note that the function header does not end with a semicolon. It is not a program statement, whose execution causes something to happen. Rather, it tells the compiler that a function is being defined.

The function definition continues with the *body of the function*: the program statements that do the work.

The variable `x` used in `display()` function is only known to `display()`. It is invisible to the `main()` function. If we use this variable in `main()` function without declaring it, the compiler will give an error message because `main()` would not know anything about this variable.

This is very important to know while writing functions in C++. Variables used in a function are unknown outside the function. Thus, a *local variable* will be known to the

CamScanner

function it is defined in, but not to others. A local variable used in this way in a function is known in C++ as an *automatic variable*, because it is automatically created when a function is called and destroyed when the function returns. On the other hand, placing a variable *globally* (outside a function) allows any other function to call it.

Function Prototype: Using functions in your program requires that you first declare the function and then you define the function. No function can be called from any other function that has not first been declared. The declaration of a function is called its *prototype*. It is a line written *before* the `main()` function. The prototype tells the compiler in advance about some characteristics of a function used in the program.

Function Calling: A user defined function is called from the main program simply by using its name, including the parentheses which follow the name. The parentheses are necessary so the compiler knows you are referring to a function and not a variable that you forgot to declare.

Functions with or without Arguments

It would not be useful if all functions were as independent of each other as `main()`. Most functions depend on other functions to some extent. For example, it does not make sense to call the pow (*raise to the power*) function without specifying values for the base and the exponent. Moreover, pow is not useful unless it sends back the result of the computation. Therefore, we might say that pow depends on the caller to provide it with the two values (for the base and the exponent, respectively). Similarly, the caller depends on pow to return a meaningful result. These dependencies are sometimes called the *interface* between the caller and the called function.

Sending data to a function is called *passing arguments*—that is, sending variables, constants, or expression whose values are needed by the function. The arguments are the input data that the function needs to compute its result. This result is then returned for use by the caller.

Functions can be used in four variations:

1. Function that does not need any argument and also returns no value.
2. Function that does not need any argument but returns a value.
3. Function that needs one or more arguments but returns no value.
4. Function that needs one or more arguments and also returns a value.

Some examples of prototypes of user-defined functions are:

```
void display(void);
int average(void);
void subset(int x);
float big_power(double x, double y);
```

The first function neither returns nor accept any arguments. Such type of function is called *simple function*. The second function returns an integer value and does not accept any arguments. The third function returns no value but it accepts an argument of the type integer. The fourth function accepts two `double` arguments (the value of x and the value of y) and returns a value of the type `float`.

CamScanner

Notice that in the function definition at the beginning of the function itself, the semicolon is not used at the end. Consider the program:

```
// program 11-2.cpp
#include <iostream>
#include <windows.h> // Required for the Beep() function
using namespace std;

void two_beeps();                    // function prototype
int main()
{
  cout << "My computer will beep twice" << endl;
  two_beeps();                       // function calling
  return 0;
}

// This function will make a beep sound twice
void two_beeps()  // function header
{
  int x;
  for (x = 0; x < 2; ++x)
  {
    Beep(1000, 500); // Beep at 1000 Hz for 500 milliseconds
    Sleep(500); // Pause for 500 milliseconds between beeps
  }
}
```

This program first prints *My computer will beep twice* and then calls a function, `two_beeps()`, which sounds two beeps at 1000 Hz for 500 milliseconds. This is all we need to call a function: the function name, followed by parentheses.

Passing Arguments to a Function

You can pass an argument to your own functions. Any data you want to send to the function must be in the parentheses. You can pass more than one argument, as long as the number of arguments in the function call, and their data types, match those expected by the function. The list of variables being passed to a function is called an *argument list*.

Now let's see how to pass an argument to your own custom function. Consider this program:

```
// program 11-3.cpp
#include <iostream>
using namespace std;

void table(int num);
int main()
{
  int num1, num2;
```

CamScanner

```
  cout << "Enter Table No. ";
  cin >> num1;
  table(num1);

  cout << "\n\nEnter Table No. ";
  cin >> num2;
  table(num2);

  return 0;
}

void table(int num)
{
  int i;
  cout << "\n***Table of " << num << "***";
  for(i = 1; i <= 10; i++)
    cout << "\n" << num << " * " << i << " = " << num * i;
}
```

The function `table()` is called twice in the `main()`, first with the argument num1 and second time with the argument num2. The `table()` function receives the value of the argument into a variable named num each time. The variable num will have the same value as the variable num1 for the first call and equal to the mum2 for the second call. Also notice that the variable num is declared as `int` in braces. It tells the compiler what type of value will be received by the function. If the function needs any other variables, they must be declared after the brace as usual.

Here is what happens in the program:

1. The `table()` function is called and is passed the value of num1.
2. Suppose the value of 5 is received by the argument named num in the function.
3. The program will print the table of 5.
4. The process will be repeated second time for variable num2.

Passing Multiple Arguments to a Function

You can pass as many items to the function as you need, of any type. The function `area()` calculates the area of a floor. The length and width of the floor, and the floor's number, are input in `main()`, then passed to the function in the call

```
area(length, width, fnum);
```

Here is the program:

```
// program 11-4.cpp
#include <iostream>
using namespace std;

void area(float size, float wide, int num);
int main()
{
```

CamScanner

```
    float length, width;
    int fnum;

    cout << "Enter the floor number: ";
    cin >> fnum;
    cout << "Enter the length of the floor: ";
    cin >> length;
    cout << "Enter the width of the floor: ";
    cin >> width;

    //length, width and fnum are actual arguments
    area(length, width, fnum);

    return 0;
}

//size, wide and num are formal arguments
void area(float size, float wide, int num)
{
  float area;
  area = size * wide;
  cout << "\nThe area of floor " << num << " is: " << area;
}
```

The output from the program will be as:

```
Enter the floor number: 4
Enter the length of the floor: 17.4
Enter the width of the floor: 21.7
The area of floor 4 is: 377.58
```

In the above example, the function call `area()` contain arguments `length`, `width` and `fnum` which are different from the corresponding argument names in the function header (which are `size`, `wide` and `num`). This is not a C++ requirement and in fact we sometimes use same names. But the complier would still treat them as different variables since they are in different functions.

The variables `length`, `width` and `fnum` are called *actual arguments*, whereas the variables `size`, `wide` and `num` are called *formal arguments*. Any number of arguments can be passed to a function being called. However, the type, order and number of actual and formal arguments must always be same.

In program 11-4 notice that three arguments are received in the same order as they are passed. In this case, the value of `length` is received by `size`, the value of `width` is received by `wide`, and the value of `fnum` is received by `num`. The argument types match—two `floats` are received by two `floats`, and an `int` is received by an `int`.

The function `area()` also needs a variable to store the results of the calculation. The variable, also named `area`, is declared after the opening brace.

If you made a mistake and called the function like this:

CamScanner

```
area(width, length, fnum);
```

the value of `width` would be received by `size`, and the value of `length` would be received by `wide`. Because the types match, your compiler would not generate an error, and in this case the program would still generate the correct result—the order in which you multiply the numbers does not matter. But suppose you called the function using this argument list:

```
area(fnum, width, length);
```

The function would receive `fnum` into the argument `size`; `width` into `wide`, and `length` into num. Two of the arguments do not match in type. If your compiler did not detect an error, the results would certainly be incorrect.

Function that Return a Value

A function can receive values and return them as well. You call the function, it gets a certain piece of information and returns it to you.

Your own functions return a value or return `void`. As stated earlier that `void` is a signal to the complier that no value will be returned. To return a value from a function, write the keyword `return( )` followed by the value you want to return. The `return( )` statement has two purposes:

1. Executing it immediately and transferring control from the function back to the calling program.
2. Whatever is inside the parentheses following return is returned as a value to the calling program.

Fig. 11.1 shows a function returning a value to the calling program.

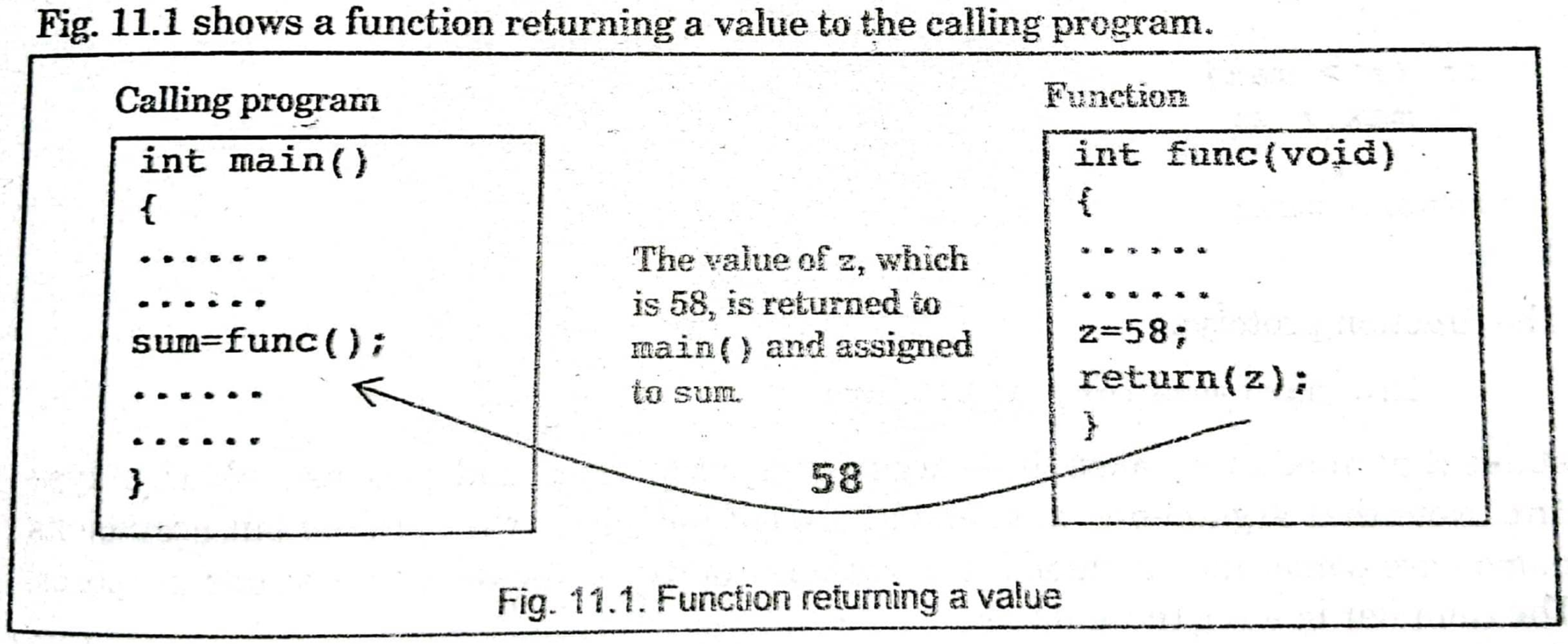


Fig. 11.1. Function returning a value

The `return` statement need not be at the end of the function. It can be placed anywhere in the function; as soon as it is encountered, control will return to the calling program, and any statement following the `return` will not be executed.

Main limitation in the use of the `return( )` statement is that you can only use it to return *one value*.

The program 11-5 uses a user-defined function `maximum` to determine and return the largest of the three integers. The three integers are input with `cin`. Next, the integers are passed to `maximum` which determines the largest integer. This value is returned

CamScanner

main by the return statement in maximum. The value returned is assigned to the variable largest which is then printed in the cout statement.

```
// program 11-5.cpp
#include <iostream>
using namespace std;

int maximum(int, int, int);    // function prototype
int main()
{
   int a, b, c;

   cout << "Enter three integers: ";
   cin >> a >> b >> c;
   cout << "Maximum is: " << maximum(a, b, c) << endl;

   return 0;
}

// Function to find maximum of three integers
int maximum(int x, int y, int z)
{
   int max = x;

   if (y > max)
      max = y;

   if (z > max)
      max = z;

   return max;
}
```

The function prototype

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

states that maximum takes three arguments of type `int`, and returns a result of type `int`. Note that arguments (x, y, and z) are not included. It is optional but arguments names are sometimes included in function prototypes for documentations purposes. The compiler ignores these names.

Here is an example in which a function returns type `float`. In this program, the `main()` function asks the user for the radius of a sphere, and then calls a function named `area()` to calculate the area of the sphere. The `area()` function returns the area of the sphere in the form of a floating point value.

```
// program 11-6.cpp
#include <iostream>
using namespace std;
```

CamScanner

```
float area(float rad);          // function prototype
int main()
{
  float radius;

  cout << "Enter radius of sphere: ";
  cin >> radius;
  cout << "Area of sphere is " << area(radius) << endl;

  return 0;
}

float area(float rad)   // returns area of sphere
{
  return (4*3.14159*rad*rad);
}
```

Practical 6

Object

Writing a function that returns larger number from 2 arguments.

Source Code

```
/* program 11-7.cpp: Practical 6 - Writing a function that returns
larger number from 2 arguments. */
#include <iostream>
using namespace std;

int maximum(int, int);   // function prototype
int main()
{
  int a, b;
  cout << "Enter first integer: ";
  cin >> a;
  cout << "Enter second integer: ";
  cin >> b;
  cout << "\nLarger number is: " << maximum(a, b) << endl;

  return 0;
}

// Function to find larger of two integers
int maximum(int x, int y)
{
  int max = x;
  if (y > max)
    max = y;

  return max;
}
```

CamScanner

Output

```
Enter first integer: 47
Enter second integer: 93

Larger number is: 93
```

Example 11.1

```
/* program 11-8.cpp Writing a function, which generates Factorial
of n and calls this function in the 'main' program. */
#include <iostream>
using namespace std;

double factorial(int x);    // function prototype
int main()
{
  int num;
  double fact;

  cout << "\nEnter the number to find its Factorial: ";
  cin >> num;

  if (num >= 0)
  {
    fact = factorial(num);
    cout << "\nThe factorial of " << num << " is " << fact << endl;
  }
  else
    cout << "\nSorry!!" << endl;

  return 0;
}

double factorial(int x)    // factorial function
{
  int i;
  double f = 1;
  for (i = 1; i <= x; i++)
    f = f * i;

  return f;          // return function
}
```

Output

1st Run

```
Enter the number to find its Factorial: 4
The factorial of 4 is 24
```

2nd Run

```
Enter the number to find its Factorial: 75
The factorial of 75 is 2.48091e+109
```

CamScanner

```
3rd Run
        Enter the number to find its Factorial: -7
        Sorry!!

4th Run
        Enter the number to find its Factorial: 0
        The factorial of 0 is 1
```

Using More Than One Function in a Program

You can use as many functions as you want in a program, and any of the functions can call any of the other functions. All the functions are accessible to all other functions.

Let us look at a program 11-7 that uses three functions including the `main()`. In this program `main()` calls one of the two functions, `fact()` or `table()`. The function `fact()` calculates the factorial of a number and returns its value to `main()` whereas the function `table()` prints the table of a number.

```
// program 11-9.cpp
#include <iostream>
using namespace std;

double fact(int m);       // function prototypes
void table(int n);
int main()
{
 int c, num;              // user-supplied values
 double answer;

cout << "\n\n\n";
cout << "\t\t\tMENU";
cout << "\n\t\t\t____\n\n";
cout << "1. Factorial of a number\n";
cout << "2. Table of a number\n";

cout << "\n";
cout << "Enter the Menu option: ";
cin >> c;

if(c == 1)
{
   cout << "\nEnter the number to find its Factorial: ";
   cin >> num;
   answer = fact(num);
   cout<<"\nThe factorial of " << num << " is " << answer << endl;
}
else if(c == 2)
{
   cout << "\nEnter the number whose table is required: ";
   cin >> num;
   table(num);
```

CamScanner

```
 }
 else
 {
    cout << "\n\nInvalid Menu option";
    cout << "\nPress any key to continue...";
 }
 return 0;
}

double fact(int m)          // factorial function
{
 int i;
 double f = 1;

 for(i = 1; i <= m; i++)
    f = f*i;
 return f;              // returns factorial
}

void table(int n)           // table function
{
 int i;
 cout << "\n***Table of " << n << "***";
 for(i = 1; i <= 10; i++)
    cout << "\n" << n << " * " << i << " = " << n*i;
}
```

Here is example of output from the program:

```
      MENU
1. Factorial of a number
2. Table of a number

Enter the Menu option: 1

Enter the number to find its Factorial: 21

The factorial of 21 is 5.10909e+19
```

Notice that none of the functions is nested inside any other. Another point to note is that functions can appear in any order in the program listing. But arranging the functions in an order makes them easier to read, especially in larger programs with number of functions. Also note that the `main()` function does not need to be the first one in the program, although it usually is.

Example 11.2

```
/* program 11-10.cpp Writing a program which uses multiple arguments
in a function. (Develop a user-defined function to generate a
rectangle. Use the function for passing arguments to draw different
sizes of rectangles and squares). */
```

CamScanner

```
#include <iostream>
using namespace std;

void rectangle(int length, int width);   // function prototype
int main()
{
 cout << "\nDrawing room\n";  // print room name
  rectangle(6, 3);                    // draw room
 cout << "\nBedroom\n";              // etc.
  rectangle(8, 2);
 cout << "\nBathroom\n";
  rectangle(2, 1);
 cout << "\nKitchen\n";
  rectangle(4, 3);

  return 0;
}

// function draws rectangle of particular length and width
void rectangle(int length, int width)
{
  int j, k;
  for (j = 1; j <= width; j++)        // number of lines
  {
    for (k = 1; k <= length; k++)    // line of rectangles
     cout << "\xDB";                 // print one rectangle
   cout <<endl;                    // new line
  }
}
```

Output

```
Drawing room
```



```
Bedroom
```



```
Bathroom
```



```
Kitchen
```



CamScanner

Practical 7

Object

Writing a function to draw stars of a given size and shape.

Source Code

```
/* program 11-11.cpp: Practical 7 - Writing a function to draw
stars of a given size and shape. */
#include<iostream>
using namespace std;

// Function declarations
void halfPyramid();
void invertedHalfPyramid();
void fullPyramid();
void invertedFullPyramid();
int main()
{
  int ch;
  do
  {
      // Menu options
      cout<<"1. Print Half Pyramid of Stars\n";
    cout<<"2. Print Inverted Half Pyramid of Stars\n";
    cout<<"3. Print Full Pyramid of Stars\n";
    cout<<"4. Print Inverted Full Pyramid of Stars\n";
    cout<<"5. Exit\n";
    cout<<"Enter the Choice: ";
  cin>>ch;
  switch(ch)
    {
      case 1:
        halfPyramid();
        break;
      case 2:
        invertedHalfPyramid();
        break;
      case 3:
        fullPyramid();
        break;
      case 4:
        invertedFullPyramid();
        break;
    case 5:
      return 0;
    default:
      cout<<"\nWrong Choice!";
      break;
}
```

CamScanner

```
    }
    while(ch>=1 && ch<=4);
    cout<<endl;
    return 0;
}

// Function to print half pyramid of stars
void halfPyramid()
{
    int i, j;
    for(i=0; i<6; i++)
    {
     for(j=0; j<=i; j++)
       cout<<"* ";
     cout<<endl;
    }
    cout<<endl;
}

// Function to print inverted half pyramid of stars
void invertedHalfPyramid()
{
    int i, j;
    for(i=0; i<6; i++)
    {
     for(j=i; j<6; j++)
       cout<<"* ";
     cout<<endl;
    }
    cout<<endl;
}

// Function to print full pyramid of stars
void fullPyramid()
{
    int i, space, j;
    for(i=1; i<=6; i++)
    {
     for(space=1; space<=(6-i); space++)
       cout<<" ";
     for(j=1; j<=i; j++)
       cout<<"* ";
     cout<<endl;
    }
    cout<<endl;
}

// Function to print inverted full pyramid of stars
void invertedFullPyramid()
```

CamScanner

```
{
   int i, space, j;
   for(i=1; i<=6; i++)
   {
    for(space=1; space<i; space++)
      cout<<" ";
    for(j=i; j<=6; j++)
      cout<<"* ";
    cout<<endl;
   }
   cout<<endl;
}
```

Output

```
1. Print Half Pyramid of Stars
2. Print Inverted Half Pyramid of Stars
3. Print Full Pyramid of Stars
4. Print Inverted Full Pyramid of Stars
5. Exit
Enter the Choice: 3
      *
     * *
    * * *
   * * * *
  * * * * *
 * * * * * *
```

Recursive Function

Recursion is a method of problem solution that depends on successively restating the problem in a consistently similar manner, but in a simpler form until a position is reached where no more simplification can take place and the simplest case reaches a *base value.* This fundamental solution can then be used to recycle through the decomposed problem elements thus reaching a solution.

When applied to computer languages, recursive function means that a function can call *itself.* Although at first this seems mysterious, recursive solutions can sometimes be simpler to write (and read) than iterative solutions.

We shall consider an example which is easy to describe and evaluate.

Multiplication can be considered in a recursive manner since it can be written as a repeated addition, each successive term being a smaller instance of the problem with the terminating condition occurring when the last term evaluates to zero

$$x \times n = x \times (n-1) + x$$
$$x \times (n-1) = x \times (n-2) + x$$
$$\text{until } x \times 0 = 0$$

the function could be called multiply(x,n); multiply(x,n) can be decomposed until $n = 0$. Consider the multiplication of 5×3 and decompose it successively into 'smaller' instances

CamScanner

5×3 decomposes to $5 \times 2 + 5$
5×2 decomposes to $5 \times 1 + 5$
5×1 decomposes to $5 \times 0 + 5$
5×0 decomposes to 0, *i.e.*, the termination value.

recycling from this position the evaluation is

5×1 becomes $5 \times 0 + 5 = 5$
5×2 becomes $5 \times 1 + 5 = 10$
5×3 becomes $5 \times 2 + 5 = 15$

In computing terms, the procedure or function is made to *called itself* until the final case is reached. The function calls are then recycled in reverse order to produce the solution.

A recursive definition must have some terminating value. Note that if there is not a *base case* then disaster will occur since the recursion cannot conclude.

The factorial is another common recursively defined structure, *i.e.*

$$\text{factorial } n = n! = n \times (n-1) \times (n-2) \times \ldots \times 1 \qquad \text{for } n > 0$$

This can clearly be expressed recursively as

$$\text{factorial}(n) = \text{factorial}(n-1) \times n$$

and the terminating condition happens at

$$\text{factorial}(1) = 1$$

for example, $7! = 7 \times 6 \times 5 \times 4 \times 3 \times 2 \times 1$

Also,

$n! = n \times (n-1)!$ *e.g.*, $7! = 7 \times 6!$
$(n-1)! = (n-1) \times (n-2)!$ *e.g.*, $6! = 6 \times 5!$
$(n-2)! = (n-2) \times (n-3)!$ *e.g.*, $5! = 5 \times 4!$ and so on.

Here $n!$ is defined in terms of $(n-1)!$, which is defined in terms of $(n-2)!$, which is defined in terms of $(n-3)!$, and so on until we reach 0! which is defined as 1.

Example 11.3

Let us calculate 4! using the recursive definition. This calculation requires the following nine steps.

1. $4! = 4 \times 3!$
2. $3! = 3 \times 2!$
3. $2! = 2 \times 1!$
4. $1! = 1 \times 0!$
5. $0! = 1$
6. $1! = 1 \times 1 = 1$
7. $2! = 2 \times 1 = 2$
8. $3! = 3 \times 2 = 6$
9. $4! = 4 \times 6 = 24$

CamScanner

That is

Step 1. This defines 4! in terms of 3!, so we must postpone evaluating 4! until we evaluate 3! This postponement is indicated by indenting the next step.

Step 2. Here 3! is defined in terms of 2!, so we must postpone evaluating 3! until we evaluate 2!

Step 3. This defines 2! in terms of 1!

Step 4. This defines 1! in terms of 0!

Step 5. This step can explicitly evaluate 0!, since 0 is the *base value* of the recursive definition.

Step 6 to 9. We backtrack, using 0! to find 1!, using 1! to find 2!, using 2! to find 3!, and finally using 3! to find 4! This back-tracing is indicated by the reverse indention.

The following is the program that calculates factorial of a given number.

```
// program 11-12.cpp
#include <iostream>
using namespace std;

double factorial(int num);     // function prototype
int main()
{
  int num;
  double fact;

 cout << "\nEnter the number to find its Factorial: ";
 cin >> num;
 fact = factorial(num);
  cout << "\nThe factorial of " << num << " is " << fact;
  return 0;
}

double factorial(int num)
{
  if(num == 0)                          // Base case
     return 1;
  else
     return num*factorial(num-1);    // Recursive case
}
```

When num is positive, the function computes num *times the result of the next lower value of num factorial*. Each successive call diminishes the value of num until it reaches 0. At that point the function stops the *recursive decent* and returns 1. As it would with any chain of unfinished function calls, the computer retraces its steps and completes the processing, filling in the missing values. At the final retrace, the function returns the value of num!.

Function Overloading

Two or more functions can have the same name but different parameters; such

CamScanner

functions are called *function overloading* in C++. The function overloading in the C++ feature is used to improve the readability of the code. It is used so that the programmer does not have to remember various function names.

In Function Overloading "Function" name should be the same and the arguments should be different. For example:

```
// same name different arguments
int test() { }
int test(int a) { }
float test(double a) { }
int test(int a, double b) { }
```

Here, all 4 functions are overloaded functions.

Notice that the return types of all these 4 functions are not the same. Overloaded functions may or may not have different return types but they must have different arguments. For example,

```
// Error code
int test(int a) { }
double test(int b){ }
```

Here, both functions have the same name, the same type, and the same number of arguments. Hence, the compiler will throw an error.

Example 11.4

Overloading using different types of parameters

```
// program 11-13.cpp
#include <iostream>
using namespace std;
// function with int type parameters
void add(int a, int b)
{
  cout << "sum = " << (a + b);
}
// function with double type parameters
void add(double a, double b)
{
  cout << endl << "sum = " << (a + b);
}
// Driver code
int main()
{
  // call function with int type parameters
  add(10, 2);
  // call function with double type parameters
  add(5.3, 6.2);
   return 0;
}
```

CamScanner

Output

```
sum = 12
sum = 11.5
```

Example 11.5

Overloading using different types of parameters

```
// program 11-14.cpp
#include<iostream>
using namespace std;

 // function with int, double type parameters
 void add(int a, double b)
 {
   cout << "sum = " << (a + b);
 }

 // function with double, int type parameters
 void  add(double a, int b)
 {
   cout << endl << "sum = " << (a + b);
 }

// Driver code
int main()
{
   // call function with int, double type parameters
   add(10, 2.5);
   // call function with double, int type parameters
   add(5.5, 6);

   return 0;
}
```

Output

```
sum = 12.5
sum = 11.5
```

Example 11.6

Overloading using different number of parameters

```
// program 11-15.cpp
#include <iostream>
using namespace std;

// function with two int type parameters
void add(int a, int b)
{
   cout << "sum = " << (a + b);
```

CamScanner

```
}

// function with three int type parameters
void  add(int a, int b, int c)
{
  cout << endl << "sum = " << (a + b + c);
}

main()
{
  // call function with two int type parameters
  add(10, 2);
  // call function with three int type parameters
  add(5, 6, 4);

  return 0;
}
```

Output

```
sum = 12
sum = 15
```

CamScanner

Exercise

11.1. What is the difference between a function in the C++ library and a function you write yourself?

11.2. What are five advantages of using functions?

11.3. What is a function prototype?

11.4. What occurs after a function is performed?

11.5. Explain the difference between local and global variables.

11.6. How do you pass a value to a function?

11.7. How can you return a float value?

11.8. Is the use of functions a more efficient use of the programmer's time or the computer's time? Explain your answer.

11.9. When do you use the keyword `return` while defining a function? When do you not use the keyword return when defining a function?

11.10. Writ a quiz program that asks four questions, each question and answer in a different function.

11.11. Write a program in C++ to call a function which returns the cube of a given number.

11.12. Write a C++ program that asks for a distance in meters and converts it into feet.

11.13. Write a program in C++ that prints out the larger of four numbers entered from the keyboard. Use a function to do the actual comparison of the four numbers. Pass the four numbers to the function as arguments, and have the function return the answer.

11.14. Write a program that inputs a number and then uses a function to calculate and display the number to the fourth power.

11.15. Write a program in C++ that asks the user to input length of the two sides of a right-angled triangle. Make use of the built-in functions `pow` and `sqrt` functions to compute the length of the hypotenuse using the Pythagorean theorem.

11.16. Explain what is wrong with the following program:

```
#include <iostream>
using namespace std;
void dothis()
{
  cout << "This is first" << endl;
  main();
}
int main()
{
  cout << "This is second" << endl;
  return 0;
}
```

11.17. Write a program using a function to calculate the area of a rectangle.

11.18. Write a function `pow(x,y)` to calculate the value of x raised to y.

CamScanner

11.19. Write a function which receives a `float` and an `int` from `main()`, finds the product of these two and returns the product which is printed through `main()`.

11.20. Write a function which receives 5 integers and returns the product, sum and average of these numbers. Call this function from `main()` and print the results in `main()`.

11.21. What happens if the user calls recursive function with a negative argument?

Ans. The function goes into infinite recursion if the argument is negative since the base case is never reached.

11.22. Rewrite the following mathematical expressions using C++ functions:

(a) $\sqrt{(x-y)^4}$ (b) $\sqrt{u+v} \times w^4$ (c) $\left|xy - \frac{w}{z}\right|$

11.23. A 5-digit positive integer is entered through the keyboard, write a function to calculate sum of digits of the 5-digit number:

(a) without using recursion.

(b) using recursion.

11.24. Write a recursive function to obtain the first 25 numbers of a Fibonacci sequence. In a Fibonacci sequence the sum of two successive terms gives the third term. Following are the first few terms of the Fibonacci sequence:

0 1 1 2 3 5 8 13 21 34 55 89 ...

11.25. What is a function overloading? Write a C++ program to shown function overloading.

Objectives and MCQs

11.1. True or false: Functions are called mini-programs because they are usually short and have input, output, and computations.

11.2. A program module in C++ is called a function.

11.3. According to the principles of software design, functions

(a) should compute at least two results to be worth the overhead.

(b) should be short and compute one result.

(c) should be as long as necessary but compute one result.

(d) are not really necessary for good software design.

11.4. Which of these are valid reasons for using functions?

(a) they use less memory than repeating the same code.

(b) they run faster.

(c) they keep different program activities separate.

(d) they keep variables safe from other parts of the program.

11.5. True or false: The function header and the function call have identical syntax.

11.6. A function prototype

(a) is an outline of the purpose of the function.

(b) is the same as the function header.

CamScanner

(c) is the same as the function call.
(d) is not necessary in C++.

11.7. True or false: Function prototypes are not needed in C++.

11.8. Function is invoked with a function call.

11.9. Use the `void` data type when the function does not return a value.

11.10. True or false: A function can still be useful even if you cannot pass it any information and cannot get any information back from it.

11.11. True or false: In C++ language, you can place the function `main()` anywhere in your program but it will still be executed first.

11.12. A variable that is known only within the function in which it is defined is called a local variable.

11.13. A global variable is defined in a declaration
(a) in `main()` only.
(b) in the first function that uses it.
(c) in any function that uses it.
(d) outside of any function.

11.14. A global variable can be referenced in a declaration
(a) in `main()` only.
(b) in the first function that uses it.
(c) in any function that uses it.
(d) outside of any function.

11.15. Which of the following are valid reasons for using arguments in functions?
(a) to tell the function where to locate itself in memory.
(b) to convey information to the function that it can operate on.
(c) to return information from the function to the calling program.
(d) to specify the type of the function.

11.16. True or false: A global variable can be used in `main()` only.

11.17. True or false: The variables commonly used in C++ functions are accessible to all other functions.

11.18. Which of the following can be passed to a function via arguments?
(a) constants.
(b) variables (with values).
(c) preprocessor directives.
(d) expressions (that evaluate to a value).
(e) functions (that return values).

11.19. Which of the following is true?
(a) C++ functions are all equal.
(b) C++ functions can be nested within each other.
(c) C++ functions are arranged in a strict hierarchy.
(d) C++ functions can only be called from `main()`.

11.20. True or false: To return from a function you must use the keyword `return`.

11.21. A function that calls itself is a recursive function.

CamScanner

CHAPTER 12

ARRAYS

By: Sir Ahad

C++ has several methods for organizing variables of simple types (*e.g.*, int, float, char). These include grouping of homogeneous variables (arrays) and heterogeneous variables (structures).

An *array* is a collection of variables of the same type that are collectively referred to by the common name. In C++ all arrays are stored respectively in successive memory locations. We can manipulate array data type collectively or individually depending on the requirements of each particular application.

An individual variable in the array is called an *array element*. You can access individual array element by specifying the array name followed by an *index* (sometimes called *subscript*), with each index must be expressed as a nonnegative integer. In the past, computers did not display subscripts (literally, below the line script such as "x_5") so many languages use brackets or parentheses to denote an array index. C++ uses square brackets. For example, if x is an array then `x[5]` refers to the element in x whose index is 5. This is read *x sub 5* for x subscript 5.

Arrays may have from one to several dimensions.

One-Dimensional Arrays

You declare an array by designating the type of values it will store, name of the array and the maximum number of elements.

The general structure of the command is:

```
type array_name[size];
```

where *type* is a valid C++ data type (*e.g.*, int, float, char, double etc.), *array_name* is the name of the array, and *size* specifies the number of elements in an array.

CamScanner

For example, to declare an array of 5 integer numbers called *temps*, use this statement.

```
int temps[5];
```

Note the use of square brackets, rather than parentheses or braces. The square brackets [ ] tells the compiler that we are dealing with an array. As with any other variable type, you declare an array before `main()` to make it external (available to all functions), or inside `main()` or another function to make it automatic.

There are two restrictions in the use of arrays. First, you have to specify the amount of storage for a declared array at compile time (before execution). Therefore, an array must have a fixed (non-variable) size. Second, the data type of an array applies uniformly to all the elements and for this reason, an array is called a *homogeneous* data structure.

The first element in every array is the *zeroth element*. Thus, the first element of array `temps` is referred to as `temps[0]`. The second element is `temps[1]`, the third element is `temps[2]`, and so on. Notice that the elements are numbered starting with 0, so the five-element array shown in the Fig. 12.1 has elements numbered 0 to 4. There is no element `temps[5]`. This sometimes confusing style of indexing, called *zero-based indexing*, has to do with the method by which C++ accesses array elements.

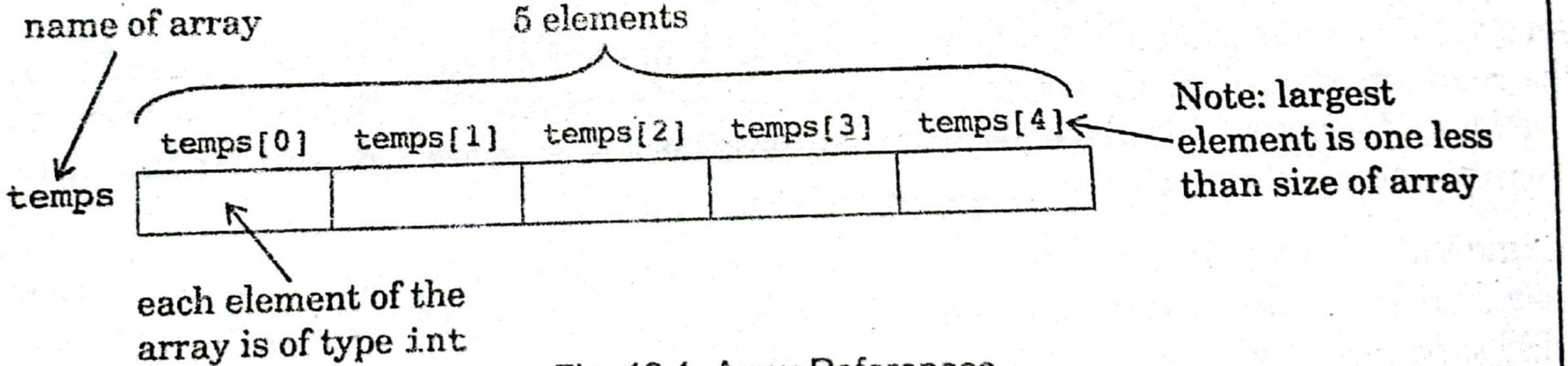


Fig. 12.1. Array References

You cannot use an index number higher than one less than the declared number of elements. If you declare a 5-element array, you must make sure you never use an index number higher than 4. Your compiler won't pick up this potential problem during the compilation process. Instead, you will either receive a runtime error when you execute the program, or get strange results.

Declaration of more than One Array

Declaration of more than one array can be made in one statement as given below.

```
int temps[5], index[7];
float fact[9], m, total;
```

In the above statements, *temps* and *index* are the arrays with five and seven elements of the data type `int`. The *fact* is an array with nine elements of the data type `float`. In addition, *m* and *total* are two simple variables of data type `float`.

CamScanner

Array Initialization

We can initialize a simple variable when we declare it as:

```
int total = 0;
```

Like this we can also initialize an array at the time of its declaration. Let `temps` is one-dimensional `int` array having 5 elements, can be declared and initialized as:

```
int temps[5] = {53, 85, 29, 19, 63};
```

This creates five elements with these values:

```
temps[0]    53
temps[1]    85
temps[2]    29
temps[3]    19
temps[4]    63
```

You can also omit the size of an array as

```
int temps[] = {53, 85, 29, 19, 63};
```

because size can be deduced from the initialization list.

A sequence of statements that manipulate this array is shown in Table 12.1.

Table 12.1. Statements that Manipulate Array temps

Statement	Explanation
`cout << temps[0];`	Displays the value of `temps[0]`, which is 53.
`temps[4] = 37;`	Stores the value 37 in `temps[4]`.
`sum = temps[2] + temps[4];`	Stores the sum of `temps[2]` and `temps[4]`, which is 66 in the variable sum.
`sum += temps[1];`	Adds `temps[1]` to sum. The new sum is 151.
`temps[4] += 2;`	Adds 2 to `temps[4]`. The new `temps[4]` is 39.
`temps[1] = temps[0] + temps[3];`	Stores the sum of `temps[0]` and `temps[3]` in `temps[3]`. The new `temp[1]` is 72.

Entering Values into an Array

Once you declare an array, you can enter information into it. If you want to assign values to every element, however, it is faster to use a loop. If you know the number of elements you want to assign, use a `for` loop:

```
// program 12-1.cpp
#include <iostream>
using namespace std;

int main()
{
  int temps[5];
  int index;
```

CamScanner

```
    for (index = 0; index < 5; index++)
    {
      cout << "Enter temperature " << index << ": ";
      cin >> temps[index];
    }
    return 0;
}
```

The variable index is used here to determine the number of repetitions. In this case, the loop is repeated 5 times, once for each element in the array. Each time the loop repeats, you will see the prompt

```
Enter temperature
```

followed by the subscript number of the element to be input and colon.

Notice that the 5 repetitions are performed by incrementing the variable from 0 to 4, not from 1 to 5.

If not enough data values are specified, the remaining elements of the array are set to 0. You can also quickly initialize an array to zero using this loop:

```
for(index = 0; index < 5; index++)
    temps[index]=0;
```

Using an array, you can input any number of elements by just changing the declaration and the loop condition.

Finding Length of an Array

The `sizeof()` operator in C++ returns the *size of the data in bytes*, which means if the operator is used with array, it returns the total number of bytes required to store an array. Hence, if we simply divide the size of the array by the size acquired by each element of the same, we can get the total number of elements present in the array.

For example, to get the size of an array, we can use the `sizeof()` operator as:

```
int temps[5] = {10, 20, 30, 40, 50};
cout << sizeof(temps);
```

But the result of this code will show 20 instead of 5, when the array contains 5 elements. It is because the `sizeof()` operator returns the size of a data type in bytes.

We have learned in Chapter 6 (data types section) that an `int` type is usually 4 bytes, so from the example above, 4 x 5 (4 bytes x 5 elements) = 20 bytes.

To find out how many elements an array has, we have to divide the size of the array by the size of the data type it contains:

```
int temps[5] = {10, 20, 30, 40, 50};
int length = sizeof(temps) / sizeof(temps[0]);
cout << length;
```

Let us see how that works,

CamScanner

```
// program 12-2.cpp
#include<iostream>
using namespace std;
int main()
{
  int  temps[] = {10 ,20 ,30, 40, 50};
  int length;

  length = sizeof(temps)/sizeof(temps[0]); //length calculation
  cout << "The length of the array is: " <<length;
  return 0;
}
```

The output of the above program is as follows:

```
The length of the array is: 5
```

As we can see, we get our desired output.

Practical 8

Object

Writing a program to calculate average of all array members.

Source Code

```
/* program 12-3.cpp: Practical 8 - Writing a program to calculate
average of all array members. */
#include <iostream>
using namespace std;

int main()
{
  int temps[7], index, total=0;
  float average;

  // Load the array with values
  for (index = 0; index < 7; index++)
  {
    cout << "Enter array element " << index << ": ";
    cin >> temps[index];
  }

  // Calculate the average temperature
  for (index = 0; index < 7; index++)
    total += temps[index];
  average = total/7.0;
  cout << "\nAverage of the array elements is: " << average <<endl;

  return 0;
}
```

CamScanner

Output

```
Enter array element 0: 65
Enter array element 1: 86
Enter array element 2: 49
Enter array element 3: 67
Enter array element 4: 74
Enter array element 5: 72
Enter array element 6: 59

Average of the array elements is: 67.4286
```

Manipulating Arrays

You can use an array element in any instruction where you would use a variable—in input or output commands or in expressions. You always refer to the individual element by its subscript number.

The values stored in an array will be accessible whenever needed in the program. The program 12-4, for instance, reads in seven temperatures of the week and uses an array in two different tasks. The values of the array elements are averaged, and then the array is used again to print a conversion chart. Each time, the array is accessed in a `for` loop that increments the index variable from 0 to the last array element.

```
// program 12-4.cpp
#include <iostream>
using namespace std;
int main()
{
int temps[7];
int index, total;
float average, celsius;
total = 0;

// Load the array with values
for (index = 0; index < 7; index++)
{
  cout << "Enter temperature " << index << ": ";
  cin >> temps[index];
}

// Calculate the average temperature
for (index = 0; index < 7; index++)
  total += temps[index];
average = total/7.0;
cout<<"\nAverage temperature of the week is: " <<average<<"\n\n";
cout << "Fahrenheit\tCelsius\n";

// Convert temperatures to Celsius and print them
for (index = 0; index < 7; index++)
{
```

CamScanner

```
      celsius = (5.0/9.0)*(temps[index]-32);
      cout << temps[index] << "\t\t" << celsius << "\n";
   }
   return 0;
}
```

In program 12-5, the `for` loops of the previous program have been replaced with do-`while` loops, and the instruction

```
index=0;
```

is used before each loop to initialize the repetition to start with the first array element. A separate instruction like this is not needed with a `for` loop, because the value is initialized in the loop arguments.

```
// program 12-5.cpp
#include <iostream>
using namespace std;
int main()
{
  int temps[7];
  int index, total;
  float average, celsius, count;
  total = 0;

  // Load the array with values
  index = 0;
  do
  {
    cout << "Enter temperature " << index << " (555 to quit): ";
    cin >> temps[index];
    index++;
  }
  while (index < 7 && temps[index-1] != 555);

  // Average the array
  index = 0;
  do
  {
    total += temps[index];
    index++;
  }
  while (index < 7 && temps[index] != 555);
  count = index;
  average = total/count;
  cout << "\nAverage temperature is: " << average << "\n\n";
  cout << "Fahrenheit\tCelsius\n";

  // Convert the values
  index = 0;
  do
```

CamScanner

```
    {
      celsius = (5.0/9.0)*(temps[index]-32);
      cout << temps[index] << "\t\t" << celsius << "\n";
      index++;
    }
    while (index < 7 && temps[index] != 555);

    return 0;
}
```

The values are input until all 7 array elements have been entered or until you enter the flag value 555, using this condition:

```
while(index < 7 && temps[index-1] != 555);
```

Notice that `1` is subtracted from the variable `index` because the variable is incremented after the number is entered. If you do not enter all 7 elements, the number 555 that you enter to end the repetition is inserted into the array element after the last actual temperature.

The number 555 is used as a flag. The program logic will watch for that value to determine whether you stopped entering temperatures before the end of the array.

When the array is averaged, a `do-while` loop totals all of the values up to the end of the array or until it reaches the value 555. While the variable `index` contains the number of temperatures, it must be an integer to be used as an array subscript. To obtain a `float` value from a division operation, however, the values divided must be `floats`. So, the program defines a `float` variable called `count` and assigns it the number of temperatures before using it to perform the mathematics:

```
count = index;
average = total/count;
```

Because the last line of the loop increments the `index` variable, the value of `index` will actually be one more than the last array element that contains a valid temperature. If you enter values into array elements 0 through 4, for example, the value of `index` will be 5. That value, however, also represents the number of elements you entered—0 through 4 is five elements—so it is used as the count to compute the average.

Examining an Array

By using repetition, you can scan the array elements, comparing the values in the array or searching for a specific value. Consider this program:

```
// program 12-6.cpp
#include <iostream>
using namespace std;
int main()
{
  int array[10];
  int index, large, small;
```

CamScanner

```
// Load the array with values
for (index = 0; index < 10; index++)
{
  cout << "Enter a number for position " << index << ": ";
  cin >> array[index];
}

// Find large and small numbers
large = array[0];
small = array[0];
for (index = 0; index < 10; index++)
{
   if (array[index] > large)
     large = array[index];
   if (array[index] < small)
     small = array[index];
}

  cout << "\n\nThe largest of the 10 elements is: "<<large << endl;
  cout << "The smallest of the 10 elements is: " << small << endl;

  return 0;
}
```

Assuming you have declared `large` and `small` as integer variables, this program examines each element in the array to determine the largest and smallest values. The key is to assign the first array element initially to both the `large` and `small` variables. If you only looked at one element, it would naturally be both the highest and the lowest. This "seeds" the variables, which are then compared against the other array variables. If an array value is greater than the current value of `large`, then it becomes the new largest value. If it is less than the current value in `small`, it becomes the new lowest value.

Example 12.1

The `#define` pre-processor directive is introduced in program 12-7. Line 4

```
#define SIZE 10
```

defines a *symbolic constant* `SIZE` whose value is `10`. A symbolic constant is a programmer-defined identifier that is replaced with replacement *text* by the C++ pre-processor before the program is compiled. When the program is preprocessed, all occurrences of the symbolic constant `SIZE` are replaced with the replacement text `10`. Using symbolic constants to specify array sizes makes programs more *scalable*. In program 12-7, the first `for` loop could fill a 1000-element array by simply changing the value of `SIZE` in the `#define` directive from `10` to `1000`. If the symbolic constant `SIZE` had not been used, we would have to change the program in three separate places to scale the program to handle 1000 array elements. As programs get larger, this technique becomes more useful for writing clear programs.

The program reads numbers from an array and graphs the information in the form of a bar chart or histogram—each number is printed, then a bar consisting of that many

CamScanner

asterisks is printed beside the number. The nested for loop actually draws the bars.

```
// program 12-7.cpp Histogram printing program
#include <iostream>
#include <iomanip>
#define SIZE 10
using namespace std;

int main()
{
  int n[SIZE];
  int i, j;

  for (i = 0; i < SIZE; i++) // load the array with values
  {
    cout << "Enter any number " << i << ": ";
    cin >> n[i];
  }

  cout << "\n" << setw(10) << "Element" << setw(13) << "Value" <<
setw(17) << "Histogram" << "\n";
  for (i = 0; i < SIZE; i++)
  {
    cout << setw(7) << i << setw(13) << n[i] << setw(9);
    for (j = 1; j <= n[i]; j++) // print one bar
      cout << "*";

    cout << "\n";
  }
  return 0;
}
```

Example 12.2

To copy the contents of one array to another, you must copy each element of the array separately. For example, this program loads array a1 with the numbers 1 through 5 and then copies them into array a2.

```
// program 12-8.cpp
#include <iostream>
using namespace std;

int main()
{
  int a1[5], a2[5];
  int i;

  for (i = 1; i < 6; i++)
  {
    cout << "Enter any number " << i << ": ";
    cin >> a1[i - 1];
  }
```

CamScanner

```
    for (i = 1; i < 6; i++)
      a2[i - 1] = a1[i - 1];

    cout << "\n\n\t" << "First array" << "\t" << "Second array" <<
"\n";

    for (i = 1; i < 6; i++)
      cout << "\t" << a1[i - 1] << "\t\t" << a2[i - 1] << "\n";

    return 0;
}
```

Example 12.3

Create three arrays. Read data into the first two of them. Subtract each element in the second array from the corresponding element in the first array. Store the differences in the third array. Print all the arrays.

Algorithm

Step 1. Loop all array elements
  Read first[*i*].

Step 2. Loop all array elements
  Read second[*i*].

Step 3. Loop all array elements
  difference[*i*] = first[*i*] – second[*i*].

Step 4. Loop all array elements
  Write first[*i*], second[*i*], difference[*i*].

Step 5. Exit.

Source Code

```
// program 12-9.cpp
#include <iostream>
#define SIZE 5
using namespace std;

int main()
{
  int first[SIZE], second[SIZE], diff[SIZE];
  int i;

  // load the first array with values
  for (i = 0; i < SIZE; i++)
  {
    cout << "Enter data item for first array for position " << i
<< ": ";
    cin >> first[i];
  }
  cout << "\n\n";
```

CamScanner

```
    // load the second array with values
    for (i = 0; i < SIZE; i++)
    {
      cout << "Enter data item for second array for position " << i
<< ": ";
      cin >> second[i];
    }

    for (i = 0; i < SIZE; i++) // compute differences
      diff[i] = first[i] - second[i];

    cout << "\n\n" << "First Array" << "\t" << "Second Array" << "\t"
<< "Difference";
      for (i = 0; i < SIZE; i++)
        cout << "\n" << first[i] << "\t\t" << second[i] << "\t\t" <<
diff[i];

    return 0;
}
```

Output

```
Enter data item for first array for position 0: 37
Enter data item for first array for position 1: 6
Enter data item for first array for position 2: 106
Enter data item for first array for position 3: 84
Enter data item for first array for position 4: -24
Enter data item for second array for position 0: 5
Enter data item for second array for position 1: 6
Enter data item for second array for position 2: 0
Enter data item for second array for position 3: 114
Enter data item for second array for position 4: 17

First Array     Second Array    Difference
37              5               32
6               6               0
106             0               106
84              114             -30
-24             17              -41
```

Sorting an Array

Sorting refers to the operation of arranging data elements in a desired sequence (order), such as increasing or decreasing. For example, a bank sorts all cheques by account number so that it can prepare individual bank statements at the end of each month. Telephone companies sort their lists of accounts by last name and, within that, by first name to make it easy to find phone numbers. Virtually every organization must sort some data and, in many cases, massive amounts of data.

Some important and frequently used methods of sorting are:

CamScanner

1. Bubble sort.
2. Insertion sort.
3. Quick sort.
4. Merge sort.
5. Radix sort.

Of these five methods, bubble sort is very simple to understand and easy to code. The general concept behind the bubble sort, indeed how it got its name, is the repeated comparisons and, if necessary, exchanges of adjacent elements. This is a little like bubbles in a tank of water with each bubble, in turn, seeking its own level.

Suppose an array a with n elements a[1], a[2], a[3], ... , a[n] is in memory. The bubble sort algorithm works as follows:

Pass 1. Compare a[1] with a[2] and arrange them in the desired order, so that a[1] < a[2]. Then compare a[2] with a[3] and arrange them so that a[2] < a[3]. Continue until we compare a[n-1] with a[n] and arrange them so that a[n-1] < a[n].
Observe that Pass 1 involves n−1 comparisons. (During Pass 1, the largest element is "Bubble up" to the nth position). When Pass 1 is completed, a[n] will contain the largest element.

Pass 2. Repeat Pass 1 with one less comparison; that is, now we stop after compare and possibly rearrange a[n-2] and a[n-1]. (Pass 2 involves n−2 comparisons and, when Pass 2 is completed, the second largest element will occupy a[n-1]).

Pass 3. Repeat Pass 1 with two less comparisons; that is, we stop after we compare and possibly rearrange a[n-3] and a[n-2].

. .....

. .....

. .....

Pass *n*−1. Compare a[1] with a[2] and arrange them so that a[1] < a[2].

After n−1 passes, the list will be sorted in increasing order.

The process of sequentially traversing through all or part of a list is frequently called a "pass", so each of the above steps is called a *pass*. Accordingly, the bubble sort algorithm requires n−1 passes, where n is the number of array elements.

Practical 9

Object

Writing a program to sort list of items in an array.

Source Code

```
/* program 12-10.cpp: Practical 9 - Writing a program to sort list
of items in an array. */
#include <iostream>
using namespace std;
```

CamScanner

```
int main()
{
  int array[100];
  int i, j, t;
  int count;

  //read in numbers
  cout << "How many numbers? ";
  cin >> count;
  cout << "\n";

  for (i = 0; i < count; i++)
  {
    cout << "Enter number " << i << ": ";
    cin >> array[i];
  }
  cout << "\n";

  // now, sort them using a bubble sort
  for (i = 1; i < count; ++i)
    for (j = count-1; j >= i; --j)
    {
      // compare adjacent elements
      if (array[j - 1] > array[j])
      {
        // exchange element
        t = array[j - 1];
        array[j - 1] = array[j];
        array[j] = t;
      }
    }

    // dispaly sorted list
    cout << "The sorted list is:\n";
    for (t = 0; t < count; t++)
      cout << "\t" << array[t] << "\n";

    return 0;
}
```

Output

```
How many numbers? 5

Enter number 0: 21
Enter number 1: 34
Enter number 2: 67
Enter number 3: 58
Enter number 4: 9

The sorted list is:
```

CamScanner

```
9
21
34
58
67
```

Searching an Array

Suppose an array a with n elements a[1], a[2], a[3], … , a[n] is in memory, and suppose a specific *item* of information is given. *Searching* refers to the operation of finding the location of *item* in array a, or printing some message that *item* does not appear there. The search is said to be *successful* if *item* does appear in array a and *unsuccessful* otherwise.

There are many different searching methods. The method that one chooses generally depends on the way the information in array a is organized.

Two important and frequently used methods of searching are:

1. Linear search.
2. Binary Search.

Of these two methods we will discuss only linear search here.

Suppose a is a linear array with n elements in memory as:

a[1], a[2], a[3], … , a[n]

The most obvious way to search for a given *item* in array a is to compare *item* with each element of a one by one. That is, first we test whether a[1]=*item*, and then we test whether a[2]=*item*, and so on. This method, which traverses array a sequentially to locate *item*, is called *linear search* or *sequential search*.

If you are just checking for any occurrence of the *item*, you can stop the search after the first matching *item* is found. If you have a thousand-element array, and you find a match in the first element, there is no need to continue searching the remaining 999 values.

Here is a program to find a value in our sample array:

Practical 10

Object

Writing a program to search a number from inputted list of items.

Source Code

```
/* program 12-11.cpp: Practical 10 - Writing a program to search a
number from inputted list of items. */
#include <iostream>
using namespace std;

int main()
{
```

CamScanner

```
int array[10];
int index, num, found;

// Load the array with values
for (index = 0; index < 10; index++)
{
  cout << "Enter any number " << index << ": ";
  cin >> array[index];
}

cout << "\nEnter the value to search for: ";
cin >> num;

found = 0;
for (index = 0; index < 10; index++)
{
  if (num == array[index])
  {
    found = 1;
    break;
  }
}

if (found == 1)
  cout << "\nThe number is in position: " << index << endl;
else
  cout << "\nThe number is not in the list" << endl;

  return 0;
}
```

Output

```
Enter any number 0: 27
Enter any number 1: 43
Enter any number 2: 78
Enter any number 3: 23
Enter any number 4: 64
Enter any number 5: 12
Enter any number 6: 34
Enter any number 7: 61
Enter any number 8: 89
Enter any number 9: 77

Enter the value to search for: 64

The number is in position: 4
```

The program assumes you have declared an integer variable named found. The variable is used as a flag; it indicates that a match has been found, and so we can stop looking at the remaining array elements.

CamScanner

The linear searching method works well for small arrays or for unsorted arrays. However, for large arrays linear searching is inefficient.

Example 12.4

Array can be loaded using do-while loops and linear searching can be performed effectively using while loop instead of for loop. Consider program 12-12:

```
// program 12-12.cpp
#include <iostream>
using namespace std;
int main()
{
  int array[10];
  int index, num, found, spot;

  // Load the array with values
  index = 0;
  do
  {
    cout << "Enter any number " << index << ": (555 to quit): ";
    cin >> array[index];
    index++;
} while (index < 10 && array[index - 1] != 555);

cout << "\nEnter the value to search for: ";
cin >> num;

index = 0;
  found = 0;
  while (!found && index < 10 && array[index] != 555)
  {
    if (array[index] == num)
    {
      found = 1;
      spot = index;
    }
    else
      index++;
}

if (!found)
    cout << "\nThe number is not in the list" << endl;
  else
    cout << "\nThe number is in position: " << spot << endl;
  return 0;
}
```

Here we use found as a unary operator. The expression

```
while(!found)
```

CamScanner

means "while not found". This statement is true when found has a zero value. Since we assign found an initial value of 0 outside the loop, the condition is true for the first repetition, and the program examines the first array element (number 0) for a match.

When a match is located, the `if` condition assigns found the value of 1. This means that "while not found" is no longer true, and the loop ends. The final value of the variable `index` will contain the element number in which the match was located.

The final `if` condition prints an appropriate message depending on whether the match was found.

Notice that the `while` statement contains three conditions with AND logic operators. The loop will repeat as long as a match has not been found, the end of the array has not been reached, and the value 555 is not located.

Passing an Array to a Function

You can pass an *individual* array element or the entire array to a function. Passing an individual array element to a function is like passing a variable. You list the array name, followed by a subscript reference to the desired element. The program 12-13 computes the sum of the first three elements in an array (*i.e.* `sum = x[0]+x[1]+x[2]`) and show its output.

```
// program 12-13.cpp
#include <iostream>
using namespace std;

int sum_elements(int a, int b, int c);    // Function prototype

int main()
{
  int x[10], total;

  x[0] = 10;        // Assign values to the first three elements
  x[1] = 20;
  x[2] = 30;

  total = sum_elements(x[0], x[1], x[2]); // Compute the sum
  cout << "\nFor values: " << x[0] << " " << x[1] << " " << x[2];
// Output the results
  cout << " the sum = " << total << endl;
  return 0;
}

int sum_elements(int a, int b, int c)      // Function header
{
  int sum;
  sum = a + b + c;
  return sum;       // End of sum elements
}
```

CamScanner

When you pass a variable other than an array, C++ makes a copy of the data and places it in a memory location associated with the receiving variable. There are two copies of the data, and changing the variable in the receiving function does not change the original variable.

When you pass an array, you are actually passing the address of the array. C++ does not make a copy of the entire array but merely assigns the same address area, and thus its data, to a second array name. Program 12-14 shows how to pass an array as an argument. Notice that only the array name is used in the calling argument, without a subscript.

```
// program 12-14.cpp
#include <iostream>
using namespace std;
int large(int[], int);      // Function prototype
int main()
{
  int list[5];
  int index, num;

  // Load the list with values
  for (index = 0; index < 5; index++)
  {
    cout << "Enter a number for position " << index << ": ";
    cin >> list[index];
  }
  num = large(list, index - 1);  // Get largest number
  cout << "\nLargest number is " << num;   // Print it
  return 0;
}

int large(int list[], int index)
{
  int dex, large;
  large = list[0];     // Assume 1st element largest
  for (dex = 1; dex <= index; dex++)  // Check remaining elements
  {
    if (list[dex] > large)  // If one is bigger
      large = list[dex];    // Make it the largest
  }
  return large;
}
```

Because you are actually passing the address of the array, you do not have to specify the number of elements in the receiving argument; just use empty brackets to indicate an array.

Passing an array by its address rather than duplicating it saves memory, but it can have some side effects. Changing an array element in the receiving function will also change the value of the passed array.

CamScanner

Multi-Dimensional Array

An array of arrays is a multi-dimensional array. In other words, multi-dimensional array is an array with two or more dimensions. As we can visualize one-dimensional array as a contiguous list, we usually visualize a two-dimensional array as a matrix or table. You can also create higher-dimensional array. But higher-dimensional arrays are more complicated.

A two-dimensional array is declared with two pairs of square brackets. For example, the following are all valid array declarations.

```
int grades[20][5];       // 20 students, each having 5 grades
float rainfall[12][31]; // daily rain for one year
double table[50][100];  // spreadsheet with 50 rows, 100 columns
char puzzle[25][25];    // crossword puzzle
```

In each case the declaration indicates the maximum number of rows followed by the maximum number of columns. As with one-dimensional arrays, the indexing is zero-based. For the declaration above, the last element you can reference in the grades array is `grades[19][4]`.

You can initialize two-dimensional arrays when you declare them. The initializer list can be arranged by row, with each row enclosed in braces or arranged linearly, as shown.

```
int x[3][2] = { {84, 54},                    // Method 1
                {43, -12},
                {59, 93} };
```

or

```
int x[3][2] = {84, 54, 43, -12, 59, 93},     // Method 2
```

which results in the assignment

84	54
43	−12
59	93

Regardless of the style you use, make sure that the data are listed in row-by-row order because that is how the data are inserted into the array. As with one-dimensional arrays, if not enough data values are specified, the remaining elements of the array are set to 0.

A three-dimensional array is declared with three pairs of square brackets. For example, the following are all valid array declarations.

```
float records[A][B][C];
int enroll[course][campus][year];
int enroll[6][4][3];
double table[50][60][70];
```

The second and third definitions of arrays are similar except that the array size is defined by constants in the third array. In the third array first subscript may take on

CamScanner

values from 0 to 5; the second, from 0 to 3; and the third, from 0 to 2. A total of 6×4×3, or 72, type int values may be stored in the array enroll. All three subscripts must be specified in each reference to array enroll in order to access a single number (*e.g.*, enroll[2][4][1]). This array can be visualized in the form of three-dimensional table as shown in Fig. 12.2.

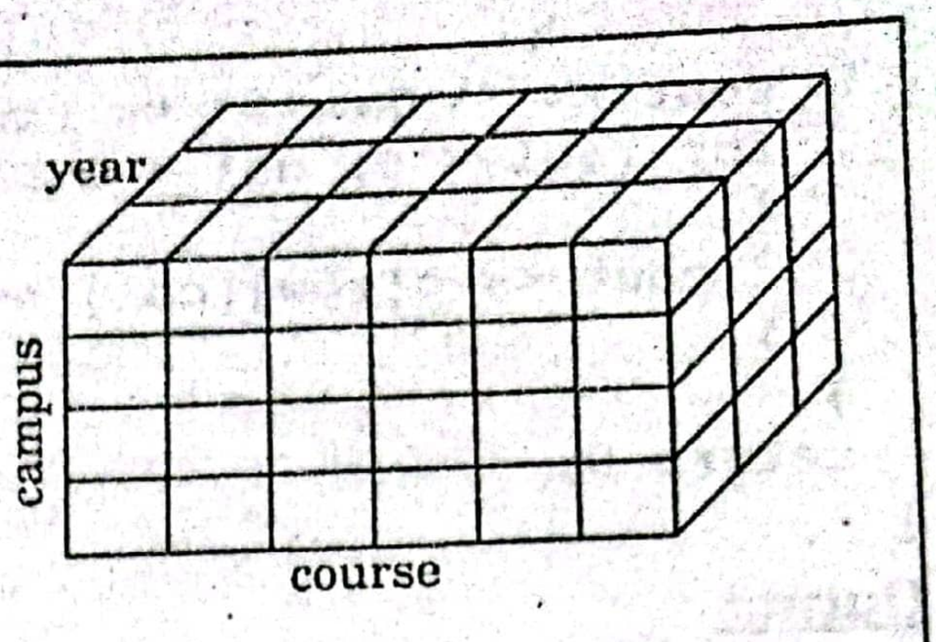


Fig. 12.2. A three-dimensional array

Practical 11

Consider an elementary problem of *matrix addition*: adding the corresponding elements of two arrays and storing the sums in a third array. All three arrays must have the same size and data type. We generally use a doubly nested for loop to add matrices. The first (outer) loop supplies the *row* index and the second (inner) loop supplies the *column* index. Assume that the input matrices are named a and b and the resulting output matrix is named c. Here is a program:

Object

Writing a program to find addition of two matrices.

Source Code

```
/* program 12-15.cpp: Practical 11 - Writing a program to find
addition of two matrices. */
#include <iostream>
using namespace std;

#define ROWS 3
#define COLS 2

int main()
{
  int a[ROWS][COLS] = {{23, 17}, {14, 34}, {3, 19}};
  int b[ROWS][COLS] = {{12, 16}, {42, 26}, {63, 20}};
  int c[ROWS][COLS];
  int row, col;

  for (row = 0; row < ROWS; row++)  // Perform matrix addition
  {
    for (col = 0; col < COLS; col++)
    {
      c[row][col] = a[row][col] + b[row][col];
    }
  }

cout << "Matrix 1 + Matrix 2 = \n"; // Output a heading
for (row = 0; row < ROWS; row++)   // Output resulting matrix
{
```

CamScanner

```
      cout << "\nRow #" << row + 1 << ":    ";
      for (col = 0; col < COLS; col++)
      {
        cout << c[row][col] << "    ";
      }
    }
    return 0;
}
```

Output

```
Matrix 1 + Matrix 2 =

Row #1:    35    33
Row #2:    56    60
Row #3:    66    39
```

Use a desk check to verify that the output matrix is indeed the sum of the two input matrices.

Example 12.5

The transpose of matrix is formed by interchanging the rows and columns of the original matrix. For instance,

$$A = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$$

then its transpose is

$$A = \begin{bmatrix} 1 & 4 & 7 \\ 2 & 5 & 8 \\ 3 & 6 & 9 \end{bmatrix}$$

and if the matrix has equal number of rows and columns, it is called a square matrix. The program 12-16 reads a square matrix and prints its transpose.

```
// program 12-16.cpp
#include <iostream>
#include <iomanip>
using namespace std;

int main()
{
  int a[10][10];
  int n, i, j, holder;

  cout << "\nEnter the size of square matrix (maximum 10): ";
  cin >> n;
  cout << "Enter each row in one line: ";
  for (i = 0; i < n; i++)
```

CamScanner

```
{
  for (j = 0; j < n; j++)
  {
    cin >> a[i][j];
  }
}

for (i = 0; i < n - 1; i++)
{
  for (j = i + 1; j < n; j++)
  {
    holder = a[i][j];
    a[i][j] = a[j][i];
    a[j][i] = holder;
  }
}

cout << "\nThe Transpose Matrix is\n";
for (i = 0; i < n; i++)
{
  for (j = 0; j < n; j++)
  {
    cout << setw(3) << a[i][j];
  }
  cout << "\n";
  }
  return 0;
}
```

CamScanner

Exercise

12.1. What is an array?

12.2. Can an array store more than one type of variable?

12.3. How do you declare an array?

12.4. What is the difference in meaning between `x3` and `x[3]`?

12.5. The simple variable `x` can take on different values, such as 5 and 12. Why is `x` not called an array?

Ans. The value of any simple variable can change overtime. Yet at any given time, only one value can be associated with a simple variable. There is a one-to-one correspondence between its name and its current value. By contrast an array has a one-to-many correspondence between its name and its current values.

12.6. For the declaration

```
char num[7];
```

how many memory cells are allocated for data storage? What type of data can be stored there? How does one refer to the initial array element? To the final array element?

12.7. What is the relationship between the value of the subscript and the value of the array element?

12.8. If an array is declared as `float x[10]`, does that mean that the indices are also floating point?

Ans. No! The data type of an array is the *kind* of data that can be stored in the individual cells of the arrays. The index of any array is *always* an integer in the range of 0 to `(array size - 1)`.

12.9. Initialize a five-element array to the values 10, −43, 32, 93, 76.

12.10. Declare one array for storing the square roots of the integers from 0 through 10 and a second array for storing the cubes of the same integers.

12.11. Write a program that reads n integers and prints the smallest along with its index in the list.

12.12. What happens if an array is referenced outside of its bounds?

Ans. Out-of-bounds references are allowed but they are not checked and the results are unpredictable! As the programmer you are responsible for ensuring that all arrays are correctly referenced.

12.13. If an array has been defined this way:

```
float cost[MAX];
```

is the following a good way to read values into all the elements of the array?

```
for(j=0; j <= MAX; j++)
    cin >> cost[j];
```

12.14. How do we set all the cells of a 100-element array to 1?

Ans. Assume that the array `j` has been declared, use a `for` loop:

```
for(index=0; index < 100; index++)
    j[index]=1;
```

CamScanner

12.15. How can you change array data?

Ans. As with simple variables, you can change array data with an input statement such as `cin` or with an assignment statement as in `list[4]=9;` As the programmer you must be sure to supply a well-defined subscript for the array.

12.16. Is this a correct way to initialize a one-dimensional array?

```
int array = {1, 2, 3, 4};
```

12.17. Write a program that reads two integer arrays, x and y, having 4 elements each and prints the sum of the products as given below.

```
sum = x[1]×y[1] + x[2]×y[2] + x[3]×y[3] + x[4]×y[4]
```

12.18. Explain what is wrong with this program?

```
#include <iostream>
using namespace std;
int main()
{
  int count[10];
  int index;
  for(index = 0; index < 100; index++)
     {
      cout << "Enter a number " << index << ": ";
      c count[index];
     }
.
.
.
}
```

12.19. A problem calls for a list of numbers to be input. The user decides how many numbers, as shown below:

```
int size, list[size];
cout << "\nEnter the size of the list";
cin >> size;
```

What is wrong with this program segment?

Ans. The size of a declared array must be a constant. Because the value of `size` is not fixed at compile time, the array is not described correctly and the declaration generates a syntax error. We have to estimate a reasonable maximum size for the array and use that number in the array declaration.

12.20. Write a program that reads ten numbers entered by the user and reports if any of them match.

12.21. Change the sorting program shown in program 12-10 (Practical 9) so that it sorts data of type `float`.

12.22. What are the two- and three-dimensional arrays? How do you declare them?

CamScanner

Objectives and MCQs

12.1. An array is a collection of variables of
- (a) different data types scattered throughout memory.
- (b) the same data type scattered throughout memory.
- (c) the same data type placed next to each other in memory.
- (d) different data types placed next to each other in memory.

12.2. A simple variable differs from an array variable because a simple variable
- (a) can have only one value during the entire execution of a program.
- (b) can have many values at one time during the execution of a program.
- (c) can have many values but only one value at any given time.
- (d) has a many-to-one relationship between the variable and its associated data.

12.3. Individual elements of an array are referred to by means of an index or subscript.

12.4. Is this a correct array declaration?

```
int max(7);
```

12.5. Which element of the array does this expression reference?

```
list[2]
```

12.6. What is the difference between the 5s in these two expressions?

```
int temps[5];
temps[5] = 84;
```

- (a) first is particular element, second is type.
- (b) first is size, second is particular element.
- (c) first is particular element, second is array size.
- (d) both specify array elements.

12.7. The following declares a `float` array whose elements are numbered 0 through 10:
- (a) `float rainfall[10];`
- (b) `float rainfall[11];`
- (c) `float rainfall[9];`
- (d) `float rainfall[0:10];`

12.8. What does this combination of statements do?

```
#define MAXSIZE 10
int list[MAXSIZE];
```

- (a) make `MAXSIZE` a subscript.
- (b) makes `MAXSIZE` a variable of type `float`.
- (c) makes `list[]` an array of type `MAXSIZE`.
- (d) makes `list[]` an array of size `MAXSIZE`.

12.9. The following sets all cells of a five-element array to the value of 10?
- (a) `int x = {10, 10, 10, 10, 10};`
- (b) `int x[] = {5*10};`
- (c) `int x[5] = {5*10};`
- (d) `int x[] = {10, 10, 10, 10, 10};`

CamScanner

12.10. What will happen if you try to put so many elements into an array when you initialize it that the size of the array is exceeded?

(a) nothing.
(b) possible system malfunction.
(c) error message from the compiler.
(d) other data may be overwritten.

12.11. What will happen if you put too few elements in an array when you initialize it?

(a) nothing.
(b) possible system malfunction.
(c) error message from the compiler.
(d) unused elements will be filled with 0s or garbage.

12.12. What will happen if you assign a value to an element of an array whose subscript exceeds the size of the array?

(a) the element will be set to 0.
(b) nothing, it is done all the time.
(c) other data may be overwritten.
(d) possible system malfunction.

12.13. A one-dimensional array is sometimes called a contiguous list.

12.14. When you pass an array as an argument to a function, what is actually passed?

(a) the address of the array.
(b) the values of the elements in the array.
(c) the address of the first element in the array.
(d) the number of elements in the array.

12.15. True or false: A function operates on an integer array passed to it as an argument by placing the values of that array into a separate place in memory know only to the function.

12.16. True or false: Individual array elements cannot be passed easily to a function.

12.17. True or false: To pass a whole array to a function, you specify the array name.

12.18. True or false: Address of a floating-point variable is always a whole number.

12.19. To pass an individual element of an array to a function, you must specify

(a) the data type of the array, its name, and the element number.
(b) the name of the array and the element number as two separate arguments.
(c) the name of the array and the element number as one argument.
(d) just the element number.

12.20. Assuming the array declaration `float values[MAX_VALUE];` which statement will pass the whole array to function `calc_max`?

(a) `calc_max(float values[]);`
(b) `calc_max(values[MAX_VALUE]);`
(c) `calc_max(float values[MAX_VALUE]);`
(d) `calc_max(values);`

12.21. True or false: Multi-dimensional arrays are sometimes called arrays of arrays.

12.22. True or false: Up to three dimensions for an array are allowed in C++.

CamScanner

12.23. True or false: Multi-dimensional arrays allow multiple data types.

12.24. True or false: In a two-dimensional array, the first dimension always refers to the row and the second dimension always refers to the column.

12.25. Two-dimensional arrays are sometimes called matrices or tables.

12.26. The following declares a two-dimensional double array with four rows and three columns.

(a) `double x[3,4];`

(b) `double x[4,3];`

(c) `double x[3][4];`

(d) `double x[4][3];`

12.27. Can you initialize a two-dimensional array this way?

```
int array[3][3] = { {1, 2, 3},
                    {4, 5, 6},
                    {7, 8, 9} };
```

12.28. The following initializes a two-dimensional array to all ones?

(a) `int x[3][2]={7*1};`

(b) `int x[3][2]={1,1,1,1,1,1};`

(c) `int x[3][2]={1};`

(d) `int x[3][2]={{1,1},{1,1},{1,1}};`

12.29. Which statement initializes the array so that it looks like the pictures?

7	7	6	6	6
7	6	6	6	6

(a) `x[5][2]={{7,7}, {7,6}, {6,6}, {6,6}, {6,6}};`

(b) `x[2][5]=({7,7,6,6,6}, {7,6,6,6,6});`

(c) `x[5][2]={{7,7,6,6,6}, {7,6,6,6,6}};`

(d) `x[2][5]={{7,7,6,6,6}, {7,6,6,6}};`

CamScanner

CHAPTER 13

By: Sir Ahad

STRINGS

A string is a sequence of characters that is used to represent text such as words, names and sentences. There are two types of strings commonly used in C++ programming language:

1. C-style strings (also called a character string)
2. C++-style strings. Strings that are objects of string class (the Standard C++ Library string class)

C-Style Strings

The C-style string originated within the C language and continues to be supported within C++. This string is actually a one-dimensional array of characters which is terminated by a null character '\0'. When you declare a C-style string, you give it a name and specify the maximum number of characters it can hold. The last array element, however, is reserved for the null terminator (\0), so you need to declare the array as one element larger than the maximum number of actual characters.

The following declaration and initialization create a string consisting of the word "Hello". To hold the null character at the end of the array, the size of the character array containing the string is one more than the number of characters in the word "Hello."

```
char greeting[6] = {'H', 'e', 'l', 'l', 'o', '\0'};
```

If you follow the rule of array initialization, then you can write the above statement as follows:

```
char greeting[] = "Hello";
```

Following is the memory presentation of above defined string in C/C++:

CamScanner

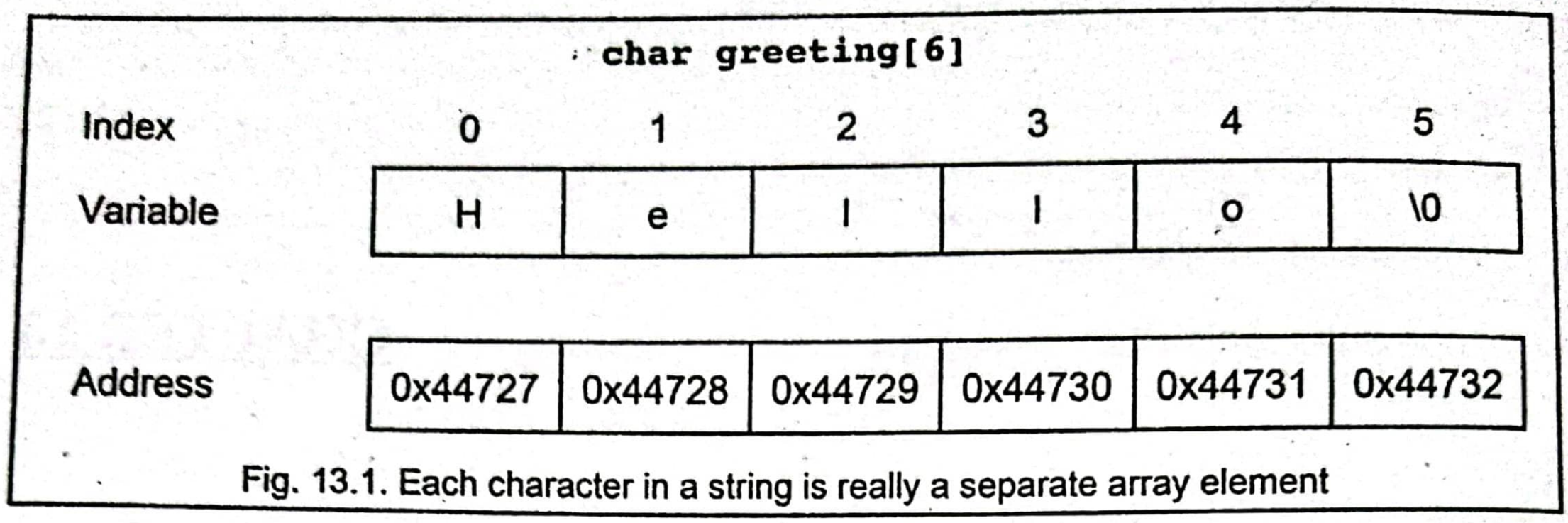


Fig. 13.1. Each character in a string is really a separate array element

Actually, you do not place the null character at the end of a string constant. The C++ compiler automatically places the '\0' at the end of the string when it initializes the array. Let us try to print above-mentioned string:

```
// program 13-1.cpp
#include <iostream>
using namespace std;
int main ()
{
char greeting[6] = {'H', 'e', 'l', 'l', 'o', '\0'};//C-style string
cout << greeting << endl;      // Output: Hello!
return 0;
}
```

When the above code is compiled and executed, it produces the following result:

```
Hello
```

In C-style strings, each character is really a separate array element, as shown in Fig. 13.1. For instance, you can input a string and then display its individual characters, like this:

```
// program 13-2.cpp
#include <iostream>
using namespace std;

int main()
{
  char name[20];
  int index;
  cout << "Enter your name: ";
  cin.getline(name, 20);
  for (index = 0; index < 20; index++)
    cout << name[index] << endl;
  return 0;
}
```

If you do not enter all 20 characters, those after the null terminator will contain unknown values. Do not overflow the array that holds the string.

CamScanner

In the program 13-2, we used `cin.getline(name, 20);` instead of `cin >> name;` to get the string content, including the space. As with `cin >> name`, where name is a character array, you only get the first word of the string. Therefore, use `getline()` as indicated in the program 13-2 to receive the whole line, regardless of whether the line contains a space or not.

Another issue is that because we set the character limit to 20, entering data longer than that results in data loss. That is, after the first 19 characters, all data will be lost. because name can only hold up to 20 characters of string data.

C-style strings have a number of functions available in the C++ library (`cstring`) that can be used to manipulate and process them, such as `strlen()`, `strcpy()`, `strcat()`, and `strcmp()`. However, working with C-style strings requires manual memory management and can be error-prone if not handled carefully.

C++-Style Strings

C++-style strings are part of the C++ Standard Library and are represented using the string class. They provide a more convenient and safer way to work with strings in C++. Here is an example:

```
// program 13-3.cpp
#include <iostream>
#include <string>
using namespace std;

int main()
{
  string greeting = "Hello"; // C++-style string
  cout << greeting << endl; // Output: Hello, world!

  return 0;
}
```

C++-style strings offer a rich set of member functions that can be used to manipulate and process strings. Some commonly used member functions include `length()`, `append()`, `find()`, `substr()`, and `compare()`. They handle memory management automatically and provide many useful features, making them the preferred choice for string manipulation in C++.

In addition, C++-style strings can be easily converted to C-style strings using the `c_str()` member function when necessary.

Built-in String Functions

C++ supports special functions for working with C-style strings. While you can perform the same functions by writing your own instructions, the built-in functions are more efficient. Table 13.1 lists some typical string functions from `<cstring>` header found in C++ libraries. The `cstring` header is the C++ version of the classic `string.h` header from C.

CamScanner

Table 13.1. Some Typical String Functions

Function	Description
`strcat`	Adds the character of one string to the end of another.
`strchr`	Returns the position of a specified character in the string.
`strcmp`	Compares two strings alphabetically.
`strcmpi`	Compares two strings; not case sensitive.
`strcpy`	Copies one string, or string constant, to another.
`strncpy`	Copies specified characters from one string to another.
`strcspn`	Returns the position of a character in the string from a specified character set.
`strdup`	Duplicates a string.
`strlen`	Calculates the string length.
`strlwr`	Converts a string to lowercase.
`strncat`	Appends specified characters from one string to another.
`strncmp`	Compares specified characters of two strings.
`strcmpi`	Compares two strings, not case sensitive.
`strncpy`	Copies specified characters from one string to another.
`strnset`	Changes specified characters in a string to another character.
`strrev`	Reverses the characters in a string.
`strstr`	Finds one string within another.
`strupr`	Converts a string to uppercase.

As an example, we will look at some string-handling functions and see how you can perform them yourself.

1. Comparing two Strings; `strcmp(string1, string2)`

The `strcmp` function compares `string1` to `string2` and returns an integer value to show the status of the comparison. If the function returns a 0, then the two strings are identical. A value of less than 0 shows that `string1` is (according to its alphabetic ordering) less than `string2`. A value greater than 0 shows that `string1` is greater than `string2`.

```
char string1[] = "Syed Asif";
char string2[] = "Syed Asif";
char string3[] = "Muhammad Asif";
char string4[] = "Muhammad Iqbal";
      cout << strcmp(string1, string2) << endl;
      cout << strcmp(string1, string3) << endl;
      cout << strcmp(string3, string4) << endl;
                              // Output 0, >0, <0
```

CamScanner

Note that when the strings are not equal your concern is only whether the result is, positive or negative and not the exact value of the function.

2. Determining String Length; `strlen(string)`

The length of a string is not necessarily the same as the size of the array. For instance, you may declare a 10-element array called name but enter a name of less than 10 characters. C++ use the `strlen()` function to determine the actual number of characters in the string, as shown here:

```
char string1[10];
char string2[] = "Syed Asif";
cout << strlen(string1) << endl << strlen(string2);
                                    // Outputs 0, 9
```

3. Combining Strings; `strcat(string1, string2)`

The `strcat` function *concatenates* or joins `string1` and `string2`. A copy of `string2` is put at the end of `string1`. Make sure that `string1` is long enough to hold the resulting string.

If `string1` is not long enough to hold the result, the extra characters will be copied into the adjacent memory locations following those associated with `string1`. The extra characters will overwrite any legitimate data, causing unpredictable results!

```
char string1[10] = "abc";
char string2[] = "def";
strcat(string1, string2);
cout << string1;                    // Output "abcdef"
```

4. Assigning Strings; `strcpy(string1, string2)`

The `strcpy` function copies `string2` into `string1`. Again, make sure that `string1` is long enough to hold the resulting string.

```
char string1[10];
char string2[] = "xyz";
strcpy(string1, string2);
cout << string1;                    // Output "xyz"
```

5. Finding Substring; `strstr(string1, string2)`

The `strstr` function takes two strings `string1` and `string2` as an argument and finds the first occurrence of the substring `string2` in the `string1`. The process of matching does not include the terminating null-characters (`'\0'`), but function stops there. This function returns a pointer points to the first character of the found `string2` in `string1` otherwise a null pointer if `string2` is not present in `string1`. If `string2` points to an empty string, `string1` is returned.

```
char string1[] = "ComputerStudies";
char string2[] = "tud";
char* p;
p = strstr(string1, string2);
if (p)
   cout << "\nString found";
else
```

CamScanner

```
    cout << "\nString not found";
```

6. Searching String; strchr(string, ch)

The `strchr` function searches `string` for the first occurrence of `ch`. We might expect this function to return the index of the search character in the string. Instead, the function returns the *memory location* (address) of the character. Significantly, the address is set to the predefined value `NULL` if the character is *not* in the string.

```
char string[10] = "Syed Asif";
char search     = 'i';
if (strchr(string, search) != NULL)
   cout << "\nSearch character found";
else
   cout << "\nSearch character not found";
```

As with `strcmp`, your only concern is whether or not the character was found and not where in the string the character resides.

Example 13.1

This program request input of two strings, then demonstrates the four string functions with them.

```
// program 13-4.cpp
#include <iostream>
#include <cstring>
using namespace std;
int main()
{
 char str1[80], str2[80];
 int i;

 cout << "Enter first string: ";
 cin.getline(str1, 80);
 cout << "Enter second string: ";
 cin.getline(str2, 80);

 // see how long the strings are
 cout << "\n\n";
 cout << str1 << " is " << strlen(str1) << " chars long\n";
 cout << str2 << " is " << strlen(str2) << " chars long\n";

 // compare the strings
 i = strcmp(str1, str2);
 if (i == 0)
   cout << "The strings are equal\n";
 else if (i < 0)
   cout << str1 << " is less than " << str2 << "\n";
 else
   cout << str1 << " is greater than " << str2 << "\n";
```

CamScanner

```
    // concatenate str2 to the end of str1 if there is enough room
    if (strlen(str1) + strlen(str2) < 80)
    {
      strcat(str1, str2);
      cout << str1 << "\n";
    }

    // copy str2 to str1
    strcpy(str1, str2);
    cout << str1 << " " << str2 << "\n";
    return 0;
}
```

One of the outputs will look like this:

```
Enter first string: Syed Asif
Enter second string: Muhammad Iqbal

Syed Asif is 9 chars long
Muhammad Iqbal is 14 chars long
Syed Asif is greater than Muhammad Iqbal
Syed AsifMuhammad Iqbal
Muhammad Iqbal Muhammad Iqbal
```

Example 13.2

This program request input of a string, then prints it diagonally.

```
// program 13-5.cpp
#include<iostream>
#include <cstring>
using namespace std;
int main()
{
  char name[30];
  int index1, index2;

  cout << "Enter your name: ";
  cin.getline(name, 30);

  // Iterate over each character of the string
  for(index1 = 0; index1 < strlen(name); index1++)
  {
   // Print spaces before the character to create diagonal effect
   for(index2 =0; index2 < index1; index2++)
     cout<<" ";          // Print spaces for horizontal offset

    // Print the character at the current position
    cout << name[index1] << endl;
   }
return 0;
}
```

CamScanner

The output of the program will be:

```
Enter your name: Syed Asif
```

```
S
 y
  e
   d

     A
      s
       i
        f
```

The size of the string is provided by the function `strlen()`, which returns the length of the value of its string argument, not counting the `'\0'` character.

The program prompts the user to enter their name using the `cout` statement. The `cin.getline()` function is used to read the input string from the user and store it in the name array. It ensures that only up to 30 characters are read, including spaces.

The program then enters a loop that iterates over each character of the string. The loop condition `index1 < strlen(name)` ensures that the loop runs until all characters of the string have been processed. The `strlen()` function is used to determine the length of the input string.

Inside the loop, there is another nested loop using the `index2` variable. This loop is responsible for printing the necessary number of spaces before each character to create the diagonal effect. The loop condition `index2 < index1` ensures that the loop runs for each character position before the current character. Within the nested loop, the statement `cout << " ";` is used to print a space character for horizontal offset before printing the actual character.

After printing the required number of spaces, the program prints the character at the current position using `cout << name[index1]`. It then adds a newline character with `endl` to move to the next line and create the diagonal effect.

Practical 12

Object

Writing a program to display your inputted name in reverse order using `strlen()`.

Source Code

```
/* program 13-6.cpp: Practical 12 - Writing a program to display
your inputted name in reverse order using strlen(). */
#include <iostream>
#include <cstring>
using namespace std;

int main()
{
```

CamScanner

```
    char str[80];
    int i; // Declare a variable to be used as a loop counter

    cout << "Enter your string: ";
    cin.getline(str, 80); // Read a line of input from the user and
store it in the 'str' array

    cout << endl << "String after reverse: ";

    // Iterate through the characters of the string in reverse order
    // Start from the last character (strlen(str) - 1) and go
backwards until the first character (0)
    for(i = (strlen(str) - 1); i >= 0; i--)
      cout << str[i];      // Output each character in reverse order

    return 0;
}
```

This program takes a string as input from the user, and then prints the reversed version of the string. The for loop is used to iterate through the characters of the string in reverse order. The loop initializes `i` to the index of the last character in the string `(strlen(str) - 1)`. The loop condition checks if `i` is greater than or equal to 0 and decrements `i` by 1 in each iteration. Inside the loop, the program outputs each character of the string in reverse order using `cout << str[i];`.

Output

```
Enter your string: Computer Studies

String after reverse: seidutS retupmoC
```

Array of Strings

You can have an array of strings, just as you can have an array of any other data type. But an array of strings is really an array of arrays of characters. An array whose elements are arrays is called a two-dimensional array.

You can picture a two-dimensional array as a spreadsheet with rows and columns. You must define such an array with two subscripts—one specifies the number of rows in the spreadsheet; the other represents the number of columns. The following instruction declares an array of 5 rows and 20 columns, with 100 integer variables in all:

```
int table[5][20];
```

Visualize each element as an integer value in its own cell of the 5×20 spreadsheet. The element `table[0][0]` is in the top-left cell, and element `table[0][1]` is in the cell to its right.

When you declare an array of strings, you must also use two subscripts. The first subscript represents the maximum number of strings in the array; the second represents the maximum length of each string. So, if you declare

CamScanner

```
char names[5][20];
```

you can have five names up to 19 characters each.

To initialize the array of names, consider the following declaration:

```
char names[5][20] = {"Abbas", "Qasim", "Azam", "Umar", "Abbas"};
```

The names in quotes are already a one-dimensional array, therefore, we do not need braces around each name as we did for two-dimensional numeric array. We do need braces around all the names because this is an array of strings.

Example 13.3

The program in this example asks you to type in your name. When you do, it checks your name against a master list to see if your name is in the list or not. It displays the message accordingly.

```
// program 13-7.cpp
#include <iostream>
#include <cstring>
using namespace std;
int main()
{
  int index, enter = 0;
  char name[20];
 const char list[5][20] = {"Abbas", "Qasim", "Azam", "Umar", "Abbas"};

  cout << "Enter your name: ";
  cin.getline(name, 20);                    // get name

 for (index = 0; index < 5; index++)        // go thru list
 {
   if (strcmp(list[index], name) == 0)      // if match
   {
     enter = 1;  // set flag
     break;
   }
 }

 if (enter == 1)                            // if flag set
   cout << "\nYour Name is in the list"; // one response
 else                                       // otherwise
   cout << "\nYour Name is not in the list";   // different one
   return 0;
}
```

The run of the program shows:

```
1st Run
        Enter your name: Arif
        Your Name is not in the list
```

CamScanner

2nd Run

```
Enter your name: Umar
Your Name is in the list
```

You will notice in the program, that we used a function `strcmp` which is available in the header file `<cstring>`. Accordingly, `<cstring>` file is included in the beginning of the program. This function will compare the string entered from the keyboard with the strings already available in the program. If the comparison is true, then the message *"Your Name is in the list"* will be displayed otherwise message *"Your Name is not in the list"* will be displayed.

Example 13.4

One common use of strings is to support a *command-based interface*. Unlike a menu, which allows the user to make a selection, a command-based interface displays a prompting message, waits for the user to enter a command, and then does what the command requests. Many operating systems use command-line interfaces, for example. The following program is similar to the program 9-21. It allows the user to add, subtract, multiply, or divide, but does not use a menu. Instead, it uses a command-line interface.

```
// program 13-8.cpp
#include <iostream>
#include <cstring>
#include <cstdlib>        // atoi
using namespace std;

int main()
{
  char command[20], temp[20];
  int i, j;

  cout << "Operation? ";
  cin.getline(command, 20);

  cout << "Enter first number: ";
  cin.getline(temp, 20);
  i = atoi(temp);

  cout << "Enter second number: ";
  cin.getline(temp, 20);
  j = atoi(temp);

if (strcmp(command, "add") == 0)          // now perform operation
   cout << "Answer is " << i + j << endl;
else if (strcmp(command, "subtract") == 0)
   cout << "Answer is " << i - j << endl;
else if (strcmp(command, "divide") == 0)
{
   if (j <= 0)
```

CamScanner

```
      cout << "Second number should be greater than zero.";
    else
      cout << "Answer is " << i / j << endl;
  }
  else if (strcmp(command, "multiply") == 0)
    cout << "Answer is " << i * j << endl;
  else
    cout << "Unknown Command.";

  return 0;
}
```

One of the outputs will look like this:

```
Operation? multiply
Enter first number: 32
Enter second number: 5
Answer is 160
```

Notice that this example also introduces another of C++'s standard library functions: `atoi()`. The `atoi()` function returns the integer equivalent of the number represented by its string argument. For example, `atoi("50")` returns the value 50. The `atoi()` function uses the header file `<cstdlib>`.

Exercise

13.1. Is `'C'` the same as `"C"`?

Ans. No! `'C'` is a character constant and is not equal to the string constant `"C"`.

13.2. Where is the `'\0'` character in a string?

Ans. C++ supplies `'\0'` if you provide space for it in the array. The `'\0'` is important for string processing because it indicates the logical endpoint of a given string.

13.3. How do you compare two strings?

13.4. How do you assign a value to a string variable?

13.5. Assuming the following initialization:

```
char string[] = "Pakistan";
```

how would you refer to the string `"tan"` (the last three letters of the string)?

13.6. What is the wrong with the following declaration?

```
name = "Azeem";
```

13.7. Output the third character from a 40-character string.

CamScanner

13.8. Output the third string from a 40-string array of strings.

13.9. Copy the alphabetically smaller of two strings into the third position of a string array.

13.10. Copy the fifth string from an array of strings to a single string variable.

13.11. Copy the seventh string from an array of strings to the ninth string of a string array.

13.12. Write a program to find whether a character string is a palindrome. A palindrome reads the same whether it is read from left to right or right to left [ABBA is a palindrome].

13.13. What expression would you use to find the length of the string name?

13.14. Write a program to input word and then print number of vowels, with appropriate message. For example, if you enter word *Computer* it should print following message:

```
Number of vowels in word Computer is 3.
```

Objectives and MCQs

13.1. A string is

(a) a list of character.
(b) a collection of characters.
(c) an array of characters.
(d) an exaltation of characters.

13.2. True or false: C++ does not distinguish between `'C'` and `"C"`.

13.3. What is the following expression?

```
"Karachi\n"
```

(a) a string variable.
(b) a string array.
(c) a string constant.
(d) a string of characters.

13.4. True or false: Strings are stored as arrays of characters.

13.5. The string `"I like programming"` needs 19 memory cells.

13.6. String functions are accessible through the `<cstring>` header.

13.7. An array of strings is actually a character array.

13.8. Which is more appropriate for reading in a multiword string?

(a) `gets()`
(b) `getchar()`
(c) `getline()`
(d) `cin`

CamScanner

13.9. The statement `strlen(string)`

(a) returns the length of the string including the null character.
(b) returns the length of the string excluding the null character.
(c) returns the length of the string as specified in its declaration.
(d) returns the declared length of the string minus 1.

13.10. The statement `strcpy(string1, string2)`

(a) adds `string2` to the end of `string1`.
(b) adds `string1` to the end of `string2`.
(c) overwrites `string1` with `string2`.
(d) overwrites `string2` with `string1`.

13.11. The statement `strcmp("Syed", "Asif")`

(a) is not legal because `"Syed"` and `"Asif"` are not string variables.
(b) returns the value 0 because the strings are not equal.
(c) returns the value −1 because the strings are not equal.
(d) returns a positive value because `"Syed"` is greater than `"Asif"`.

13.12. True or false: The `getline()` function is accessed through `<iostream>`.

13.13. The statement `strchr("Asif", 'i')`

(a) returns 0 because `'i'` is contained in `"Asif"`.
(b) returns a negative value because `'i'` is contained in `"Asif"`.
(c) returns a positive value because `'i'` is contained in `"Asif"`.
(d) is a syntax error because `"Asif"` is not a string variable.

13.14. What does this program fragment display?

```
char x[80] = "gorilla";
char y[80] = "giraffe";
strcpy(x, y);
cout << x << " " << y;
```

(a) gorilla giraffe
(b) giraffegorilla gorilla
(c) gorilla gorilla
(d) giraffe giraffe

13.15. What does this program fragment display?

```
char x[80] = "gorilla";
char y[80] = "giraffe";
strcat(x, y);
cout << x << " " << y;
```

(a) gorillagiraffe giraffe
(b) giraffegorilla gorilla
(c) gorilla gorilla
(d) giraffe giraffe

CamScanner

CHAPTER 14

STRUCTURES

By: Sir Ahad

Structures are collections of related variables under one name. Structures may contain variables of many different data types—in contrast to arrays that contain only elements of the same data type. Using structure, we can create what are commonly called *records*: heterogeneous data structures containing logically related data.

Thus, for example, a structure may contain employee's data encompassing employee's name (a character array), gender (a character), grader (an integer), department number (an integer), salary (a floating-point number), and any other relevant information as a unit (Fig. 14.1). A structure can hold as many of these items as we need.

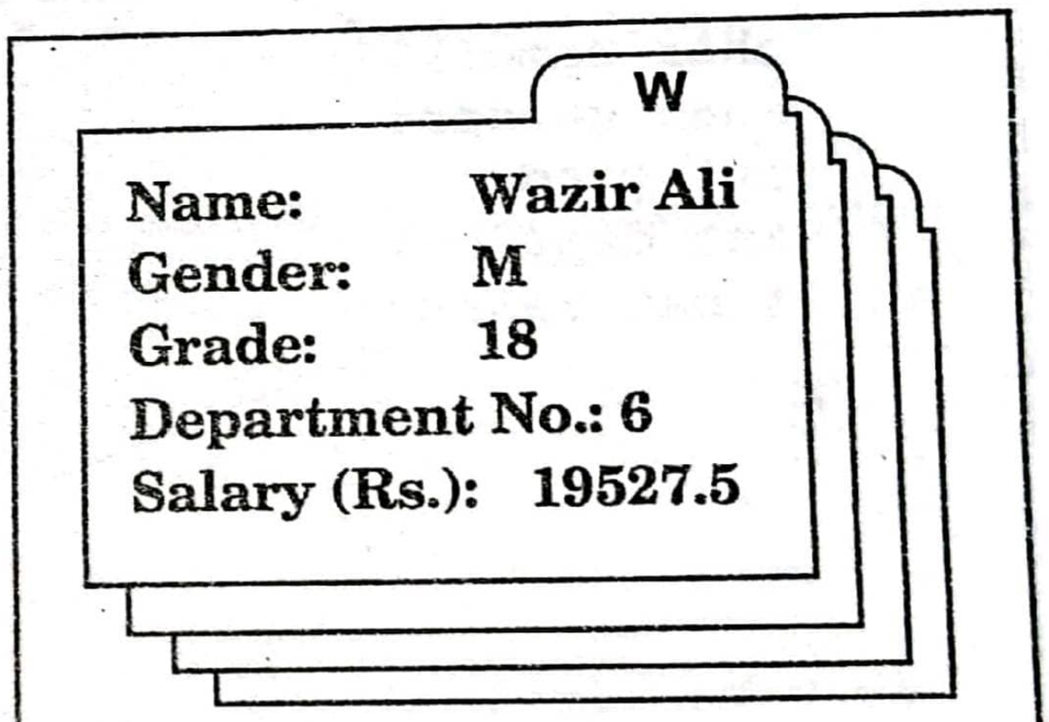
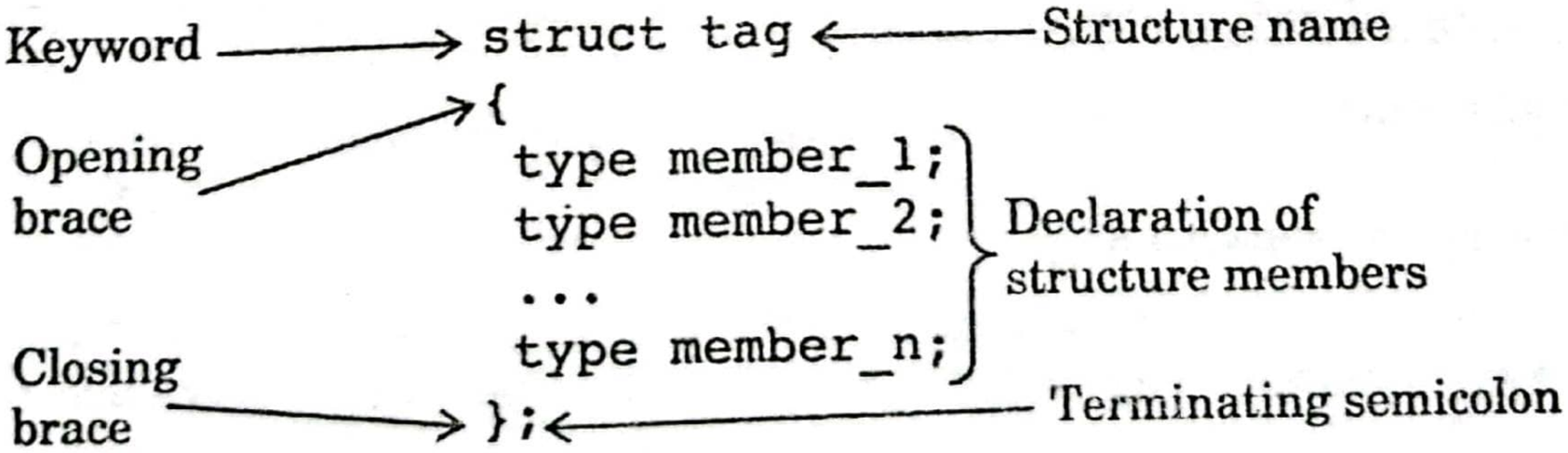


Fig. 14.1. Non-computerised record keeping

Defining a Structure

Structures are defined in C++ using the general form:



The definition of structure starts with the keyword `struct`, followed by the name of the structure, called the *tag*. The pieces of data that make up the structure are called *structure members*, and they are placed in a group between opening and closing braces.

CamScanner

You declare structure members using the same syntax you use to declare variables. You must specify the data type of each member, and the size of any strings or arrays. Place semicolons at the end of each member declaration, and at the end of the structure. Now you can define structure employee as shown below:

```
struct employee
{
 char name[50];
 char gender;
 int grade;
 int dept_no;
 float salary;
};
```

This creates a new data type called employee, which contains five pieces of information: one string, one character, two integers, and a float. The structure sets aside an area in memory 63 addresses wide (Fig. 14.2).

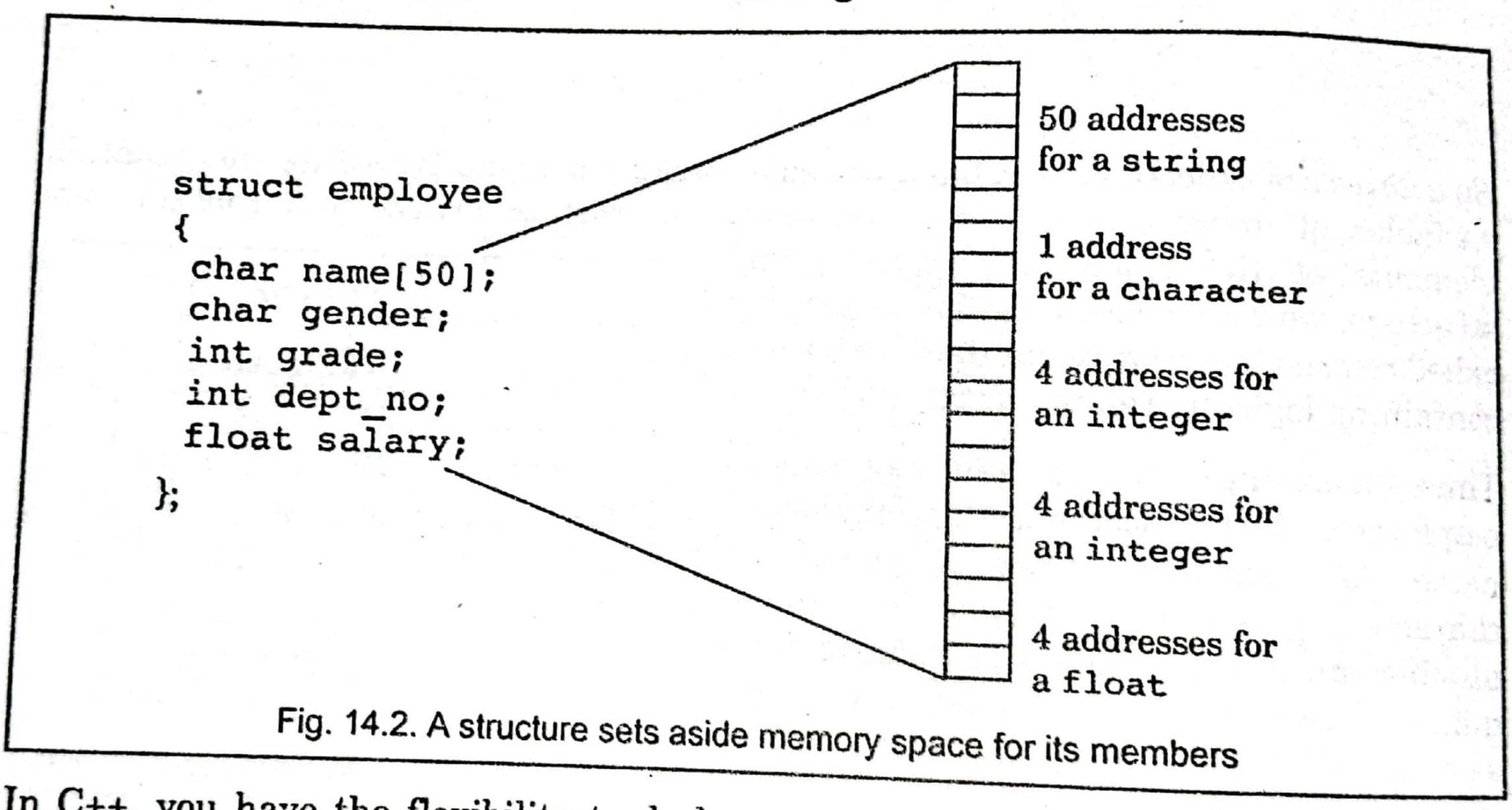


Fig. 14.2. A structure sets aside memory space for its members

In C++, you have the flexibility to declare a structure both inside and outside the main() function, depending on your program's needs. Here are the two common approaches:

Structure declared outside main():

```
struct MyStruct
{
  // Structure members
};
int main()
{
  // Your code here
  return 0;
}
```

CamScanner

By declaring the structure outside the `main()` function, you can use it throughout your program, in multiple functions if necessary. Other functions can also access and manipulate objects of this structure.

Structure declared inside `main()`:

```
int main()
{
  struct MyStruct
  {
     // Structure members
  };

  // Your code here
  return 0;
}
```

Declaring the structure inside `main()` limits its scope to the `main()` function. It will only be accessible within `main()`, and you won't be able to use it in other functions or in the global scope. This approach is typically used when the structure is specific to the `main()` function and doesn't need to be accessed elsewhere.

Both approaches have their own use cases, and the choice depends on the structure's purpose and its relevance to the rest of your program.

A `simple` Structure

Now we analyze a structure `simple` containing two data items—an integer variable `num` and a character variable `ch`. The program is shown as:

```
// program 14-1.cpp
#include <iostream>
using namespace std;
int main()
{
  struct simple // definition of data type struct simple
  {
  int num;
  char ch;
  };

  struct simple smp1; // declares smp1 of data type 'simple'

  smp1.num = 90;  // accessing members of smp1
  smp1.ch = 'p';

  cout<<"The integer member of the structure is: "<<smp1.num<<endl;
  cout<<"The character member of the structure is: "<<smp1.ch<<endl;

  return 0;
}
```

CamScanner

The program demonstrates the three fundamental aspects of using structures, namely

1. Declaring the structure type.
2. Declaring structure variables.
3. Accessing members of the structure.

1. Declaring the Structure Type

The following statement in program 14-1 declares the structure type with a tag *simple*.

```
struct simple
{
 int num;
 char ch;
};
```

The above statement defines `struct` data type with the *tag* `simple`. It consists of two members: an integer variable called *num*, and a character variable called *ch*. This statement just tells the compiler what the `struct` data type name *simple* looks like.

The tag is a *type* name and not the *variable* name. It can be used in the program as if it were a data type. The members of the structure are surrounded by braces (`{}`), and the entire statement is terminated by a semicolon (`;`).

2. Declaring Structure Variables

After the declaration of structure data type, you may declare one or more variables to be of that type. For example, in the program 14-1, you may declare variable `smp1` to be of type `struct simple` by writing the statement:

```
struct simple smp1;
```

This statement sets aside enough space in memory to hold all the items in the structure.

If you need more than one variable of the same type, you can declare them as:

```
struct simple smp1;
struct simple smp2;
```

or in one instruction

```
struct simple smp1, smp2;
```

This instruction declares two variables (`smp1` and `smp2`) that are of the `simple` structure type. Program 14-2 shows the declaration of two variables.

```
// program 14-2.cpp
#include <iostream>
using namespace std;
int main()
{
  struct simple // definition of data type struct simple
  {
  int num;
  char ch;
  };
```

CamScanner

```
struct simple smp1;    // declares smp1 of type 'simple'
struct simple smp2;    // declares smp2 of type 'simple'

smp1.num = 90;  // accessing members of smp1
smp1.ch = 'p';
smp2.num = 100;        // accessing members of smp2
smp2.ch = 'q';

cout << "The int and char members of smp1 are: " << smp1.num <<
", " << smp1.ch << endl;
cout << "The int and char members of smp2 are: " << smp2.num <<
", " << smp2.ch << endl;

return 0;
}
```

However, if you have more than one structure type defined, you must declare the variables of each type separately. For example, if you have defined the `simple` structure and another structure called `easy`, you might declare your variables like this:

```
struct simple smp1, smp2;
struct easy   esy1, esy2;
```

You can also create the structure and declare the variable in one step. Place the variable name (or names) between the structure's closing brace and semicolon, as in:

```
struct simple
{
 int num;
 char ch;
}smp1, smp2;
```

To help make the terminology clear, let's review each element in the declaration:

1. `simple` is the *structure tag*. It is the name of a new data type, a collection of items called a structure.
2. The elements that make up the structure are called *structure members*. They are declared using the same syntax as a variable declaration.
3. `smp1` (or `smp2`) is the variable name that you used in your program. It is a variable of the `simple` type, which means it contains all the members declared as part of the structure. It is sometimes called a *structure variable*.

3. Accessing Members of the Structure

A member of the structure can be accessed using structure *variable name* and the *member name* separated by the *dot operator* (`.`) as shown below:

```
smp1.num;
smp1.ch;
```

Here `smp1.num` gets member `num` from the structure variable `smp1`. Similarly, `smp1.ch` gets member `ch` from structure `smp1`.

CamScanner

Initializing Structure

If you know the initial values of the structure members, you can assign the values when you declare the variable. If you are only creating one variable of the structure type, you can initialize it as part of the structure definition:

```
struct employee
{
 char name[50];
 char gender;
 int grade;
 int dept_no;
 float salary;
}emp1={"Wazir Ali", "M", 18, 6, 19527.5};
```

This creates the employee structure, declares the emp1 variable, and assigns an initial value to each of the five structure members. If you are creating two or more than two variables you can initialize it as:

```
struct employee
{
 char name[50];
 char gender;
 int grade;
 int dept_no;
 float salary;
}emp1={"Wazir Ali", "M", 18, 6, 19527.5},
 emp2={"Zaheer Butt", "M", 19, 2, 25020.0};
```

As an alternative, you can initialize the members as:

```
struct employee
{
 char name[50];
 char gender;
 int grade;
 int dept_no;
 float salary;
};
struct employee emp1={"Wazir Ali", "M", 18, 6, 19527.5};
struct employee emp2={"Zaheer Butt", "M", 19, 2, 25020.0};
```

Because C++ is a free-form language, you can divide the assignment on several lines.

Practical 13

Object

Writing a program to create structure to display employee data with structure members as name, designation and salary.

Source Code

```
/* program 14-3.cpp: Practical 13 - Writing a program to create
```

CamScanner

```
structure to display employee data with structure members as name,
designation and salary. */
#include <iostream>
#include <cstring>
using namespace std;
int main()
{
  struct employee          // definition of data type struct employee
{
  char name[40];
  char designation[30];
  float salary;
};

  struct employee emp1; // declares emp1 of data type 'employee'
  struct employee emp2; // declares emp2 of data type 'employee'

  cout << "\nEmployee-1" << endl;
  cout << "Enter name: ";
  cin.getline(emp1.name, 40);
  cout << "Enter designation: ";
  cin.getline(emp1.designation, 30);
  cout << "Enter salary: ";
  cin >> emp1.salary;
  cin.ignore();  // Ignore the newline character in the input buffer

  cout << "\nEmployee-2" << endl;
  cout << "Enter name: ";
  cin.getline(emp2.name, 40);
  cout << "Enter designation: ";
  cin.getline(emp2.designation, 30);
  cout << "Enter salary: ";
  cin >> emp2.salary;

  cout << "\nList of Employees" << endl;

  cout << "\nName: " << emp1.name;
  cout << "\nDesignation: " << emp1.designation << ", Salary: " <<
emp1.salary << endl;

  cout << "\nName: " << emp2.name;
  cout << "\nDesignation: " << emp2.designation << ", Salary: " <<
emp2.salary << endl;

  return 0;
}
```

The structure variables `emp1` and `emp2` are declared to be of type struct `employee`. The database stores three items of information about each employee. The name and designation of the employee is represented by a character array, and a `float`

CamScanner

represents the employee salary. This information is entered into the program by the user. The program reads and prints this information. The output of the program shows:

Output

```
Employee-1
Enter name: Amir Abbas
Enter designation: Lecturer
Enter salary: 20337.5

Employee-2
Enter name: Bushra Rehman
Enter designation: Professor
Enter salary: 35946.8

List of Employees

Name: Amir Abbas
Designation: Lecturer, Salary: 20337.5

Name: Bushra Rehman
Designation: Professor, Salary: 35946.8
```

You can input and output the structure elements in any order; you do not have to use them in the order they are defined in the structure.

You can define a structure globally before the `main()` function, or locally within `main()` or another function.

Array of Structures

The real advantage in using structures comes about when we use a collection of structures, called an *array of structures*. By creating and maintaining an array of structures, a database of information can be manipulated for a wide range of items.

An array of structures is defined by the statement

```
struct employee
{
 char name[50];
 int grade;
};
struct employee emp[MAXNUM];
```

For example, to create an array of structures for 10 employees, the last statement in the above declaration would be

```
struct employee emp[10];
```

This sets aside ten sections of memory, each large enough to store a complete structure (Fig. 14.3). Refer to the array element by subscripting the structure variable, not the member name:

```
gets(emp[0].name);
gets(emp[1].name);
```

CamScanner

```
struct employee
{
 char name[50];
 int grade;
};
struct employee emp[10];
```

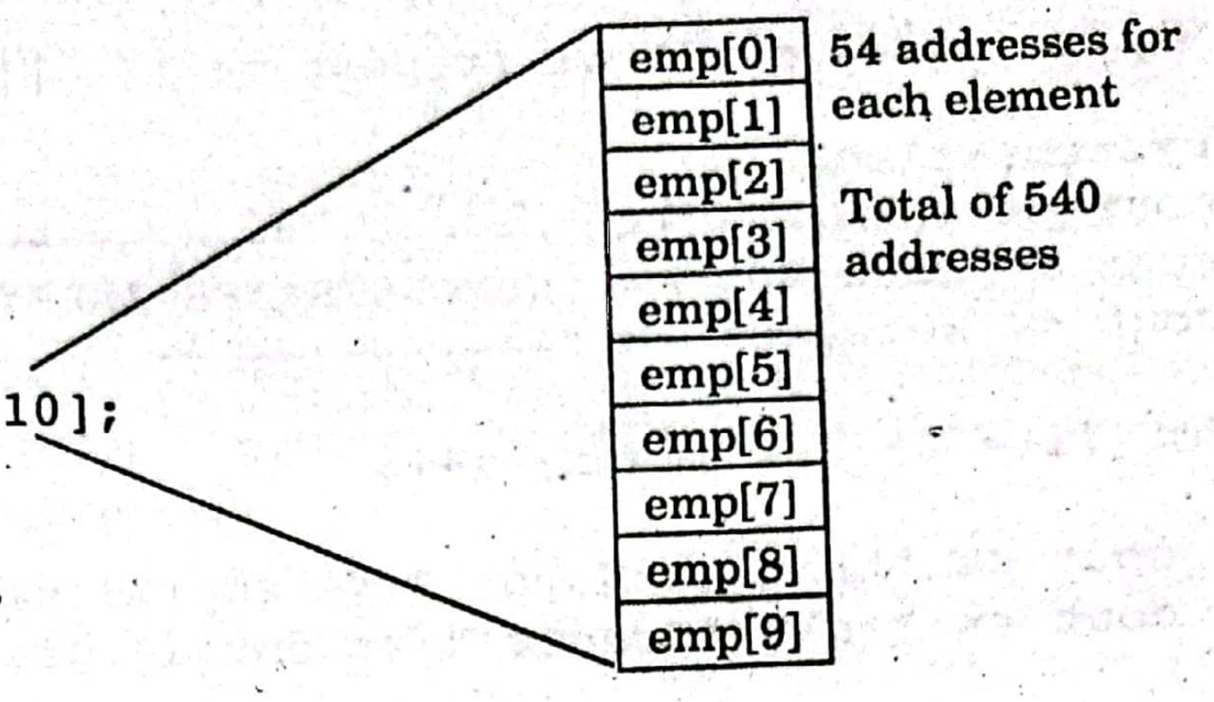


Fig. 14.3. An array of structures

The program 14-4, for instance, uses an array of structures to represent a database of hundred employees.

```
// program 14-4.cpp
#include <iostream>
#include <iomanip>
#define MAX 100
using namespace std;

struct employee   // definition of data type struct employee
{
  char name[50];;
  int grade;
};
struct employee emp[MAX];      // array of 100 structures

int main()
{
  int i, index = 0;
  char repeat = 'Y';

  do // input the employee
  {
    cout << "\nEntry No. " << index << endl;
    cout << "Enter name: ";
    cin.getline(emp[index].name, 50);
    cout << "Enter grade: ";
    cin >> emp[index].grade;
    index++;

    if (index < MAX)
    {
     cout << "Do you have another entry? Y or N: ";
     cin >> repeat;
     cin.ignore(); //Ignore the newline character in the input buffer
    }
```

CamScanner

```
}
while (index < MAX && (repeat == 'Y' || repeat == 'y'));

system("cls");
cout << setw(40) << "---------------------------------" << endl;
cout << setw(40) << "DEMONSTRATES ARRAY OF STRUCTURES" << endl;
cout << setw(40) << "---------------------------------" << endl;

for (i = 0; i < index; i++)                    // print
{
  cout << "\n\n\t\tName: " << emp[i].name;
  cout << "\n\t\tGrade: " << emp[i].grade;
  }
  return 0;
}
```

A 100-element array of emp is declared as:

```
struct employee emp[100];
```

The variable `repeat` is declared to indicate when the user is done with the entering information, if they have fewer than 100 emps. The do loop inputs the array elements: two pieces of data (structure members) for each array element. The variable `index` will be used as the subscript number.

After each array element is entered, the user is asked whether they have another entry. The question is not asked, however, if the user has already entered all 100 array elements.

The do loop uses this condition:

```
while(index<100 && (repeat == 'Y' || repeat == 'y'));
```

The AND condition means that both must be true: index is less than hundred and the variable `repeat` is `Y` or `y`. The program fills a series of inputs as:

```
Entry No. 0
Enter name: Amir Abbas
Enter grade: 17
Do you have another entry? Y or N: y

Entry No. 1
Enter name: Bushra Rehman
Enter grade: 19
Do you have another entry? Y or N: y

Entry No. 2
Enter name: Zia Awan
Enter grade: 14
Do you have another entry? Y or N: y

Entry No. 3
Enter name: Qasim Siddiqui
```

CamScanner

```
Enter grade: 16
Do you have another entry? Y or N: n
```

Without the nested parentheses, C++ would combine the first two conditions and internally interpret the instruction as an OR condition. The loop would continue even after all hundred elements are entered, if the `repeat` had the value of `y` from the last input.

When the last emp is entered, the variable `index` is one more than the number of array elements. The variable is used in the `for` loop to print a list of names and their grades. The output of the program will look like:

```
-------------------------------
DEMONSTRATES ARRAY OF STRUCTURES
-------------------------------

Name: Amir Abbas
Grade: 17

Name: Bushra Rehman
Grade: 19

Name: Zia Awan
Grade: 14

Name: Qasim Siddiqui
Grade: 16
```

Nested Structure

When a structure is a member of another structure, it is called a *nested structure*. Consider the example of creating statistical information of hockey players. The first structure could be the player's name.

```
struct pname
{
 int first[15];
 int last[25];
};
```

The next structure uses pname structure as its member.

```
struct player
{
 struct pname name;
 int number;
 char position[2];
}player1, player2;
```

The structure pname contains a player's first and last name. In the structure for a player, the name is declared to be a structure of type pname.

To assign the value `Amir` to `first[15]` of the second name structure of `player1`, use this statement.

CamScanner

```
player1.name.first[15] = "Amir";
```

As you see, the structures are referenced from the outer to the inner. This is also the general case. Whenever you have nested structures, you begin referencing with the outermost and end with the innermost.

A nested structure can be used to improve the utility of a program. Here, the mechanical information about a book is stored in a separate structure, which, in turn, is part of the `catalog` structure. The complete program is shown here. Notice how the program stores the year of publication, edition number and length of the book.

```
// program 14-5.cpp
#include <iostream>
#include <string>
using namespace std;

struct book_type
{
   unsigned int year;   // year of publication
   int ed;              // edition
   unsigned int pages;  // length of book
};

struct catalog
{
   string name;         // author name
   string title;        // title
   string pub;          // publisher
   struct book_type book;     // mechanical information
};

int main()
{
   catalog cat;

   cout << "\nEnter Information about a Book:\n";
   cout << "Enter the Author Name: ";
   getline(cin, cat.name);
   cout << "Enter the Title: ";
   getline(cin, cat.title);
   cout << "Enter Publisher: ";
   getline(cin, cat.pub);
   cout << "Enter the Year of Publication: ";
   cin >> cat.book.year;
   cout << "Enter the Edition: ";
   cin >> cat.book.ed;
   cout << "Enter Number of Pages: ";
   cin >> cat.book.pages;

   system("cls");
```

CamScanner

```
    cout << "\nThe Information about a Book is:\n";
    cout << "Name of Author:    \t\t\t" << cat.name << endl;
    cout << "Title:             \t" << cat.title << endl;
    cout << "Publisher:         \t" << cat.pub << endl;
    cout << "Year of Publication:  \t\t\t" << cat.book.year << endl;
    cout << "Edition:           \t" << cat.book.ed << endl;
    cout << "Number of Pages:   \t\t" << cat.book.pages << endl;

    return 0;
}
```

The program will take the input like:

```
Enter Information about a Book:
Enter the Author Name: Dr. Shafique Ali Khan
Enter the Title: Two Nation Theory
Enter Publisher: Royal Book Company
Enter the Year of Publication: 1985
Enter the Edition: 2
Enter Number of Pages: 494
```

and the output of the program will be:

```
The Information about a Book is:

Name of Author:                 Dr. Shafique Ali Khan
Title:                          Two Nation Theory
Publisher:                      Royal Book Company
Year of Publication:            1985
Edition:                        2
Number of Pages:                494
```

Structures and Functions

The value of a structure variable can be passed as an argument to a function just as it can be passed in an assignment statement. This feature greatly simplifies the use of functions which leads to writing simplified and short programs.

Program 14-6, for example, shows how to pass a structure variable to a function.

```
// program 14-6.cpp
#include <iostream>
using namespace std;

struct simple                   //definition of data type struct simple
{
  int num;
  char ch;
};
  struct simple smp1;    //declares smp1 of data type 'simple'

void show(struct simple smp1);       //prototype function simple
```

CamScanner

```
int main()
{
  show(smp1);     //calling function show in main()
  return 0;
}

void show(struct simple smp1)
{
  smp1.num = 90;
  smp1.ch = 'p';

  cout << "\nThe integer member of the structure is: " << smp1.num;
  cout << "\nThe character member of the structure is: " << smp1.ch;
}
```

Program 14-7 illustrates how to return a structure from a function.

```
// program 14-7.cpp
#include <iostream>
#include <string>
using namespace std;

struct CD
{
  string name;
  string singer;
  string category;
  float cost;
  int number;
};
  struct CD disc, inputdisc;

struct CD getDisc();    //function prototype

int main()
{
   CD disc = getDisc();        //function calling

   cout << "\nThe information on the CD is:\n";
   cout << "Name:          " << disc.name << endl;
   cout << "Singer:        " << disc.singer << endl;
   cout << "Category:      " << disc.category << endl;
   cout << "Cost(Rs.):     " << disc.cost << endl;
   cout << "Volume:        " << disc.number << endl;

   return 0;
}

struct CD getDisc()     //function definition
{
```

CamScanner

```
CD inputDisc;

cout << "\nEnter disk information\n";
cout << "Enter the name: ";
getline(cin, inputDisc.name);
cout << "Enter the singer: ";
getline(cin, inputDisc.singer);
cout << "Enter the category: ";
getline(cin, inputDisc.category);
cout << "Enter the cost (Rs.): ";
cin >> inputDisc.cost;
cout << "Enter the volume number: ";
cin >> inputDisc.number;

   return (inputDisc);
}
```

Program 14-7 allows the user to enter information about a CD and then displays that information. It defines a structure called `CD` with members such as name, singer, category, cost, and number to represent the `CD`'s attributes.

The `main` function calls the `getDisc()` function to get `CD` information from the user and assigns the returned value to the `disc` variable. The program then displays the `CD` information using `cout`, printing the `CD`'s name, singer, category, cost (in Rs.), and volume number.

The `getDisc()` function is responsible for collecting `CD` information from the user. It prompts the user to enter the name, singer, category, cost, and volume number using cout statements. It uses `getline()` to read the string inputs and `cin` for the cost and volume number. The function stores the entered information in a local variable `inputDisc` of type `CD` and returns it to the `main()` function.

The program will take the input like:

```
Enter disk information

Enter the name: Best Hits
Enter the singer: Noor Jehan
Enter the category: Classical
Enter the cost (Rs.): 97.5
Enter the volume number: 3
```

and the output of the program will be:

```
The information on the CD is:

Name:           Best Hits
Singer:         Noor Jehan
Category:       Classical
Cost(Rs.):      97.50
Volume:         3
```

CamScanner

Exercise

14.1. When should a structure type be used?

14.2. How is a structure declared?

14.3. What is the difference between the structure tag and a structure variable?

14.4. How do you refer to a structure element?

14.5. Explain when to use the dot (.) operator.

14.6. Can you have an array of structures?

14.7. Can a structure contain elements of the same type?

14.8. Write a segment of program to initialize the elements of a structure when it is declared?

14.9. Name some of the distinguishing differences between arrays and structures.

14.10. Show how to create a structure called `s_type` that contains these five elements:

```
char str[80];
char ch;
int i;
float f;
double d;
```

Also, define one variable called `s_var` using this structure.

14.11. Write a statement that will declare a structure variable `car` to be of the type `vehicle`.

14.12. Write a statement that declares a structure type consisting of two elements: a string of 40 characters and an integer.

14.13. Assume the following declaration for the structure `body`:

```
struct body
 {
  int arm;
  int legs:
 };
  struct body jameel;
```

Write an assignment statement that will set the number of arms in jameel's body equal to 2.

14.14. Assuming that `struct1` and `struct2` are structure variables of the same type and contain the same number of members, is the following statement possible?

```
struct1 = struct2;
```

14.15. Write a program in C++ that will:

(a) set up a structure to hold a date. The structure will consist of three integer values, for the day, month, and year.

(b) assign values to the members of the structure.

(c) printout the values in the format 23/8/2023.

CamScanner

14.16. Write a program that inputs product inventory information into a structure. The information includes the product name, cost, quantity, and the supplier's name.

14.17. Modify the above program to input information into a 20-element array of the structure.

14.18. Write a program in C++ to create a database with the following items using structured data type with the tag *patient history*.

(a) patient name
(b) patient admission number
(c) gender
(d) date of admission

14.19. Write a program in C++ to create a library information system which includes the following data members.

(a) access number
(b) author name
(c) book title
(d) price

14.20. Create a structure to represent an inventory item in a car-parts store. The item needs an item number, name, description, price, location code, and in-stock code.

14.21. Distinguish the structure data type from other data type variables.

Objectives and MCQs

14.1. Array elements must all be of the same data type, whereas structure members can be of different data type.

14.2. True or false: All structure elements are stored in contiguous memory locations.

14.3. True or false: An array is used to store dissimilar elements, and a structure to store similar elements.

14.4. True or false: An appropriate use for a structure is to store a list of prices.

14.5. The purpose of declaring a structure type is to

(a) set aside the appropriate amount of memory.
(b) define the format of the structure.
(c) specify a list of structure elements.
(d) define a new data type.

14.6. How many structure variables of a given type can you use in a program?

(a) one
(b) none
(c) as many as you like
(d) as many as there are members in the structure

14.7. True or false: It is possible to pass a structure to a function in the same way a simple variable is passed.

CamScanner

CHAPTER 15

DATA COMMUNICATION AND NETWORKS

By: Sir Ahad

Communications or *telecommunications* is the process of transmitting and receiving data across a distance. The data transferred can be voice, sound, images, video, text, or a combination thereof (multimedia). It is also the study and application of computer devices used to transmit and receive data. The four basic elements of any communication system are:

1. A *transmitter* (source or sender) which creates the message to be transmitted.
2. A *medium* through which message or the data is transmitted from on location to another.
3. A *receiver* which receives the message.
4. *Protocol*, a set of rules or procedures for transmitting data between electronic devices, such as computers.

For example, when you speak to your friend on the telephone, you are the *transmitter*. The telephone line through which your voice is transmitted is the *medium* and your friend is the *receiver*. The communication follows certain rules for transmitting voice from you to your friend are *protocols*. This is a simple example of voice communication. The same concept applies to the data communications also. But in this case, the transmitter and the receiver are normally machines, in particular, computer devices (computers, terminals, peripheral devices like printers, disks, etc.) and the transmission medium may be telephone lines, cables, microwaves links, satellite links, etc.

Computers communicate locally or within a region—for example, PCs on a network are linked directly by high-speed cables. In the larger domain, data is exchanged

CamScanner

among transmitting and receiving stations within a nation or among different nations, and along networks connected by telephone system, fiber optic cables and satellites.

A network *protocol* is a set of established rules that dictate how to format, transmit and receive data so that computer network devices can communicate, regardless of the differences in their underlying infrastructures, designs or standards. To successfully send and receive information, devices on both sides of a communication exchange must accept and follow protocol conventions. In networking, support for protocols can be built into software, hardware or both.

Without computing protocols, computers and other devices would not know how to engage with each other. As a result, except for specialty networks built around a specific architecture, few networks would be able to function, and the internet as we know it wouldn't exist. Virtually all network end users rely on network protocols for connectivity. Protocols are established by international or industrywide organizations. Perhaps the most important computer protocol is OSI (Open Systems Interconnection), a set of guidelines for implementing networking communications between computers. Among the most important sets of Internet protocols are TCP/IP (Transmission Control Protocol/Internet Protocol), HTTPS (Secure HyperText Transmission Protocol), SMTP (Simple Mail Transfer Protocol), and DNS (Domain Name System).

Direction of Transmission

An important characteristic of data transmission is direction. Data is sent in one direction, or it is sent in two directions in two types of ways (see Fig. 15.1). These three types of transmission are called simplex, half-duplex, and full-duplex.

Simplex Transmission

In *simplex transmission,* data is transmitted only in one direction. Device connected to such a circuit is either a send-only or a receive-only device. An example is data received by computer from radio or television stations in the form of voice, picture or video. Environmental devices send information about temperature and humidity to a computer, which adjusts environmental settings automatically. However, the computer does not send data back to these devices.

Half-Duplex Transmission

In *half-duplex transmission,* data communication is two-way, but the data travels in only one direction at a time. An example of a half-duplex device is a walkie-talkie, a two-way radio that has a push-to-talk button. When you press the transmit button, you can talk but not receive. When you release the transmit button, you can receive but not transmit.

Half-duplex transmission is somewhat slow, but it is often used in browsing the internet (requests and responses).

Full-Duplex Transmission

In full-duplex transmission, data is transmitted in both directions simultaneously. This method resembles two trains passing in opposite directions on side-by-side tracks.

For example, land-line telephone networks are full-duplex, since they allow both

CamScanner

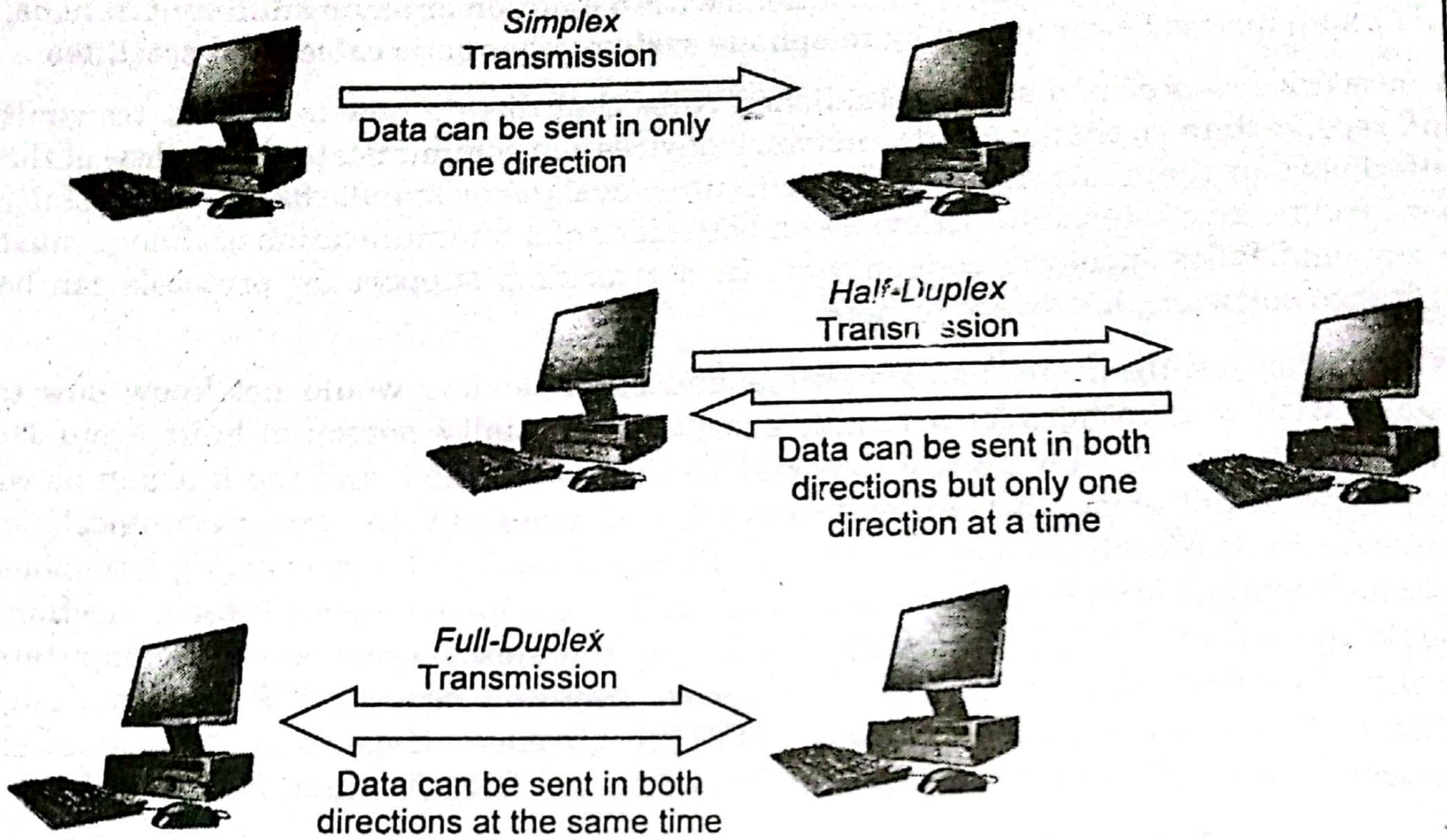


Fig. 15.1. Direction of transmission

callers to speak and be heard at the same time. Two computers connected via an Ethernet cable can send and receive data at the same time. Wireless networks also support full-duplex communication.

Data Transmission Medium

Transmission media can be broadly categorized into two main types:

1. Guided media, and
2. Unguided media

These categories refer to the physical pathways through which signals or data are transmitted in communication systems.

Guided Media

Guided media, also known as *bounded* or *wired media*, are communication channels that use physical conductors to guide and transmit signals. These conductors provide a controlled pathway for the signals to travel, which helps maintain signal quality and reduce interference. The most common guided media examples are *twisted-pair cable, coaxial cable,* and *fiber optic cable.*

Twisted-Pair Cable

Twisted-pair cable normally consists of two wires individually insulated in plastic, then twisted around each other. Generally, several such pairs are bundled together in a protective layer of plastic insulation (see Fig. 15.2). They are the most widely used Transmission Media. Twisted Pair is of two types:

CamScanner

Unshielded Twisted Pair (UTP): UTP consists of pairs of insulated copper wires twisted together. The twisting helps to reduce electromagnetic interference from external sources, enhancing signal quality. UTP cables are commonly used for Ethernet networks, telephone lines, and residential broadband connections. Different categories (Cat5e, Cat6, Cat6a, Cat7, etc.) offer varying levels of performance, with higher categories capable of supporting higher data rates and longer distances.

Shielded Twisted Pair (STP): STP adds an additional layer of shielding, usually made of metal, around the twisted pairs. This shielding provides better protection against electromagnetic interference, making STP suitable for environments with high level of interference, such as industrial settings. STP is commonly used in industrial automation, manufacturing, and data centers.

Coaxial Cable

Coaxial cable, also known as *coax*, features a central conductor, an insulating layer, a wire mesh shield, and an outer insulating cover. Fig. 15.3 shows a coaxial cable. The design provides better shielding against interference compared to twisted-pair cables. Coaxial cables are used for various applications, including cable television (CATV), broadband internet, and networking. They can support higher data rates over longer distances than twisted-pair cables.

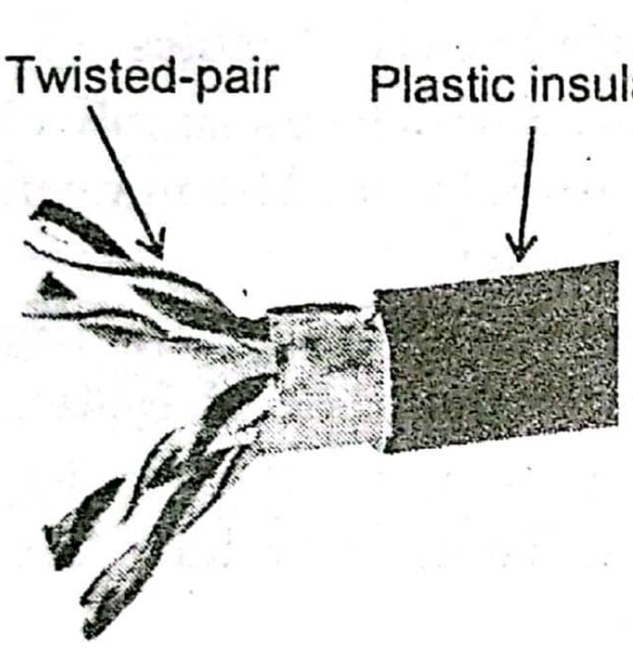


Fig. 15.2. Twisted-pair cable

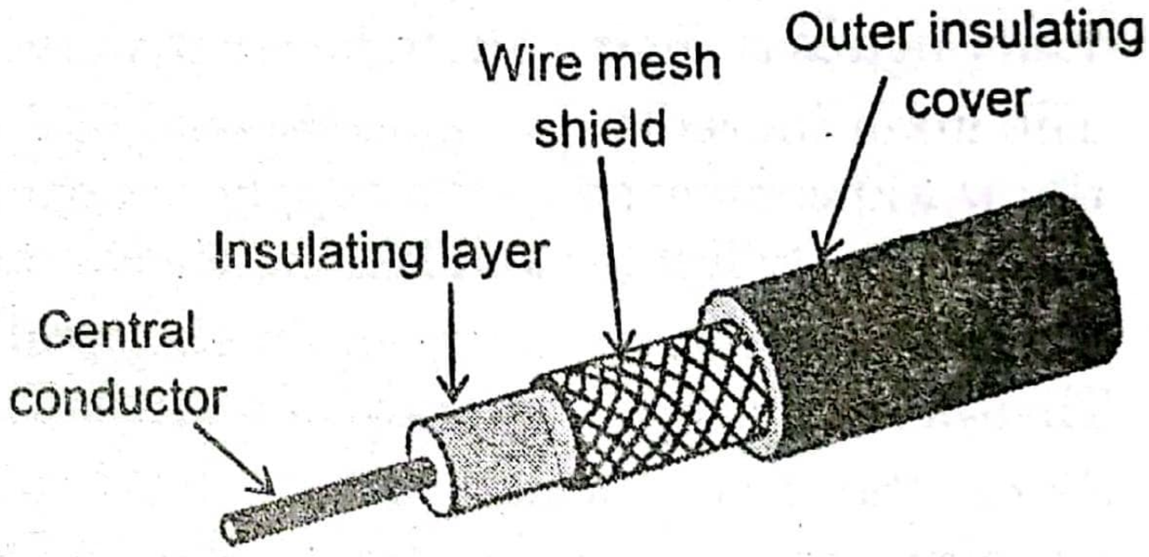


Fig. 15.3. Coaxial cable

Because of its structure, the coax cable is capable of carrying high frequency signals than that of twisted pair cable. The wrapped structure provides it a good shield against noise and cross talk. Coaxial cables provide high bandwidth rates of up to 450 megabits per second (Mbps).

Fiber Optic Cable

Fiber optic cables use thin strands of glass or plastic fibers (Fig. 15.4) to transmit data as pulses of light; each "on" pulse represents one bit. The cables are made of glass fibers, each thinner than a human hair, that can guide light beams for kilometres.

An optical fiber consists of an inner cylinder of glass, called the core, which is the actual propagating path for the light. It is surrounded by a cylindrical shell of glass or plastic of lower refractive index, called the *cladding* (see Fig. 15.5).

The transmission of light through a fiber is based on the phenomena of total internal reflection, which occurs when light is incident at an interface between two medium of different refractive indices *i.e.* core/cladding, at an angle greater than the critical *angle*.

CamScanner

In absence of the scattering defects along the interface, total internal reflection is a highly efficient phenomenon and thousands of such reflections can occur within a fiber with little loss of light.

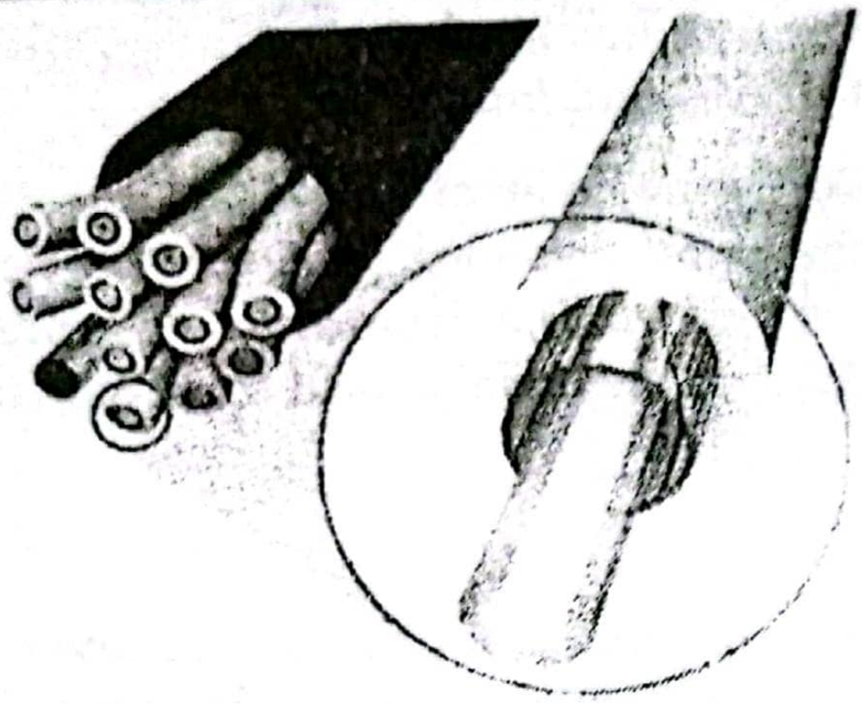


Fig. 15.4. Fiber optic cable

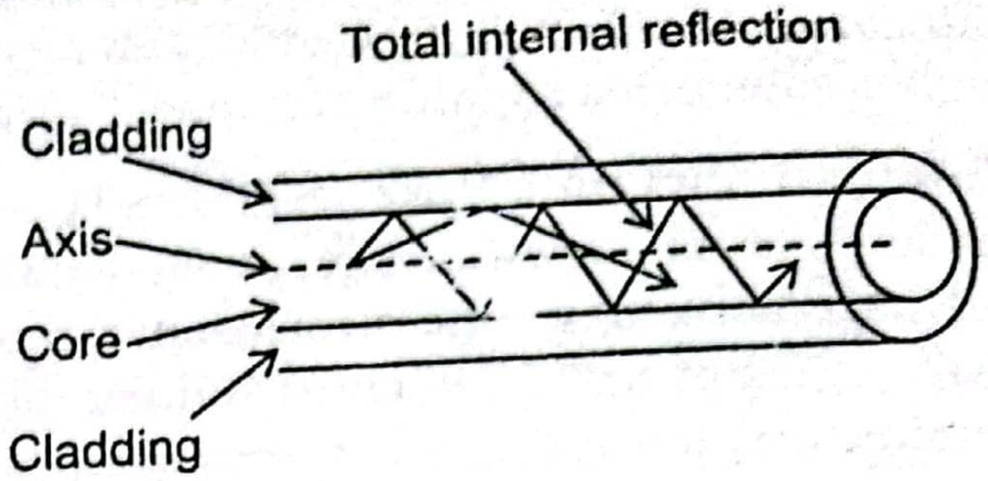


Fig. 15.5. Ray propagation through fiber optics

Advantages and Disadvantages of Fiber Optic Cable

Some of the advantages of fiber optic cable are:

1. Fiber optics transmits data faster than twisted-pair or coaxial cables.
2. The main ingredient in glass is sand and there is an enormous supply of sand in the world compared to the supply of Copper and Aluminum. The material is also substantially lighter and less expensive than wire cable.
3. Photons of light, rather than an electrical current move through the fiber optics. Therefore, there is no chance of a spark flash thereby making this system safer.
4. Since the fiber optics carries light pulses, it is not affected by random electromagnetic interference in the environment. Thus, they have much lower error rates than normal telephone wire and cable.
5. Because of the absence of current flow through the fiber optics, intrusion into the system is also prevented. Confidential information cannot be routed to unwanted receivers, nor can false information be fed into the data steam.
6. The transmission losses in fiber optics are much lower than that of the twisted-pair wires and coaxial cables.
7. Glass is immune to corrosive and oxide degradation. So, fiber-optic cable is more durable than twisted-pair and coaxial cable.

Some disadvantages of the fiber optic cable are:

1. All fiber optic cables are limited to fixed point-to-point ground installations. They cannot leave the ground nor be associated with a mobile communication station.
2. Popular light-emitting sources are restricted to very low power devices. There ar high-power devices available, but they are very expensive.
3. The ways in which the light source can be modulated are limited.
4. Because of low-power sources, the distance between repeater amplifiers must b relatively short for high data rates demanded in some systems.

CamScanner

Unguided Media

Unguided media, also known as *unbounded* or *wireless media*, do not require physical conductors to transmit signals. Instead, they use free space as the transmission medium, allowing signals to propagate through the air or vacuum. The most common unguided media examples are radio waves, microwaves, and infrared (IR).

Radio *waves, microwaves,* and *infrared waves* are all forms of electromagnetic radiation (Fig. 15.6), sharing the fundamental characteristic of propagating electric and magnetic fields perpendicular to each other and to the direction of wave travel. They all travel at the speed of light in a vacuum, enabling rapid transmission of information. While *radio waves* possess *longer wavelengths and lower frequencies*, making them suitable for broadcasting and wireless communication, *microwaves* have *shorter wavelengths and higher frequencies*, lending themselves to applications such as radar systems, and satellite communication. Infrared waves fall between visible light and microwaves in terms of wavelength, and they are absorbed by many materials, making them useful for short-range communication.

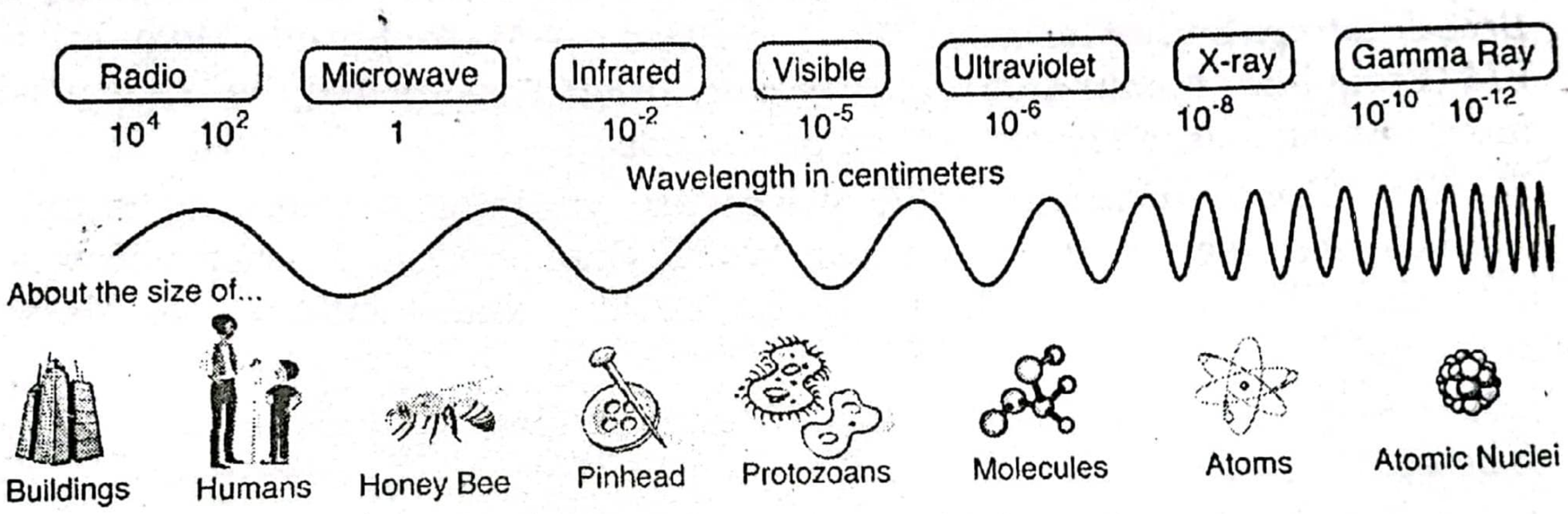


Fig. 15.6: Electromagnetic Spectrum

Despite these differences, all three types of waves are vital in modern technology, contributing to wireless communication, remote sensing, and scientific research while being considered non-ionizing radiation, meaning they generally lack sufficient energy to ionize atoms or molecules and cause harm like ionizing radiation (*e.g.*, X-rays and gamma rays).

Radio Waves

Introduction: Radio waves are the electromagnetic waves that are omnidirectional, *i.e.*, the signals are propagated in all the directions. Radio waves are the cornerstone of modern communication, serving as the foundation for broadcasting, wireless technologies, and various scientific applications. With their long wavelengths and low frequencies, radio waves have revolutionized how we connect, share information, and explore the world around us.

Characteristics of Radio Waves: Radio waves reside on the lower end of the electromagnetic spectrum, spanning a frequency range from approximately 30 kHz to 300 GHz. Due to their low frequencies, they boast long wavelengths that can stretch from meters to kilometers. This unique attribute enables them to propagate efficiently over considerable distances, making radio waves a key player in long-range communication.

CamScanner

Propagation and Behavior: Radio waves exhibit diverse propagation behaviors that contribute to their wide-ranging applications. They can reflect off surfaces, allowing them to bounce off the ionosphere for long-distance communication (known as *skywave* propagation). Additionally, radio waves can diffract around obstacles, enabling them to reach beyond line-of-sight and penetrate buildings. This makes them ideal for reaching remote or obstructed areas.

Challenges and Future Developments: Radio waves do face challenges, including interference from other electronic devices and natural sources of radio emissions. The increasing demand for wireless bandwidth poses a need for efficient spectrum management. However, advancements like 5G networks, employing advanced modulation techniques and spectrum sharing, promise enhanced data rates, lower latency, and improved connectivity. This evolution highlights the ongoing significance and adaptability of radio waves in shaping the way we communicate and interact with the world.

Applications of Radio Waves

1. *Broadcasting:* Radio waves are the foundation of AM (Amplitude Modulation) and FM (Frequency Modulation) *radio broadcasting*. They enable the transmission of music, news, and information to large audiences.
2. *Wireless Communication:* Radio waves are vital for wireless communication technologies, including cell phones, Wi-Fi, Bluetooth, and other wireless data networks. These technologies connect devices and facilitate seamless data exchange.
3. *Radar Systems:* Radar (Radio Detection and Ranging) systems utilize radio waves to detect the presence, location, and movement of objects. Radars have applications in aviation, weather monitoring, defense, and maritime navigation.
4. *Remote Sensing:* Remote sensing satellites utilize radio waves to gather critical data about Earth's surface, weather patterns, and environmental changes. This information aids in disaster management, resource monitoring, and scientific research.
5. *Radio Astronomy:* Radio telescopes capture radio waves emitted by celestial bodies, unveiling insights about the cosmos, from distant galaxies to pulsars and cosmic microwave background radiation.
6. *Radio Frequency Identification (RFID):* RFID systems use radio waves to identify and track objects, contributing to inventory management, supply chain optimization, and access control.

Microwaves

Introduction: Microwaves, a subset of electromagnetic radiation, occupy a crucial position in modern technology and communication. With their unique properties and versatile applications, microwaves have revolutionized wireless communication, radar systems, cooking (perhaps best known by most people), and scientific research, shaping the way we interact with the world.

Characteristics of Microwaves: Microwaves span a frequency range between approximately 300 MHz and 300 GHz, positioning them between radio waves and

CamScanner

infrared waves on the electromagnetic spectrum. Their relatively short wavelengths, ranging from millimeters to centimeters, grant them distinctive capabilities that are harnessed for a multitude of purposes.

Propagation and Behavior: Microwaves exhibit interesting behaviors, including *line-of-sight* propagation (Fig. 15.7). *Line-of-sight* means there must be an unobstructed view between transmitter and receiver for optimal communication. They can be focused and directed using specialized antennas, making them ideal for point-to-point communication, wireless backhaul, and radar systems. While some microwaves can penetrate certain obstacles, they are also absorbed by water molecules, leading to their remarkable use in microwave ovens for cooking.

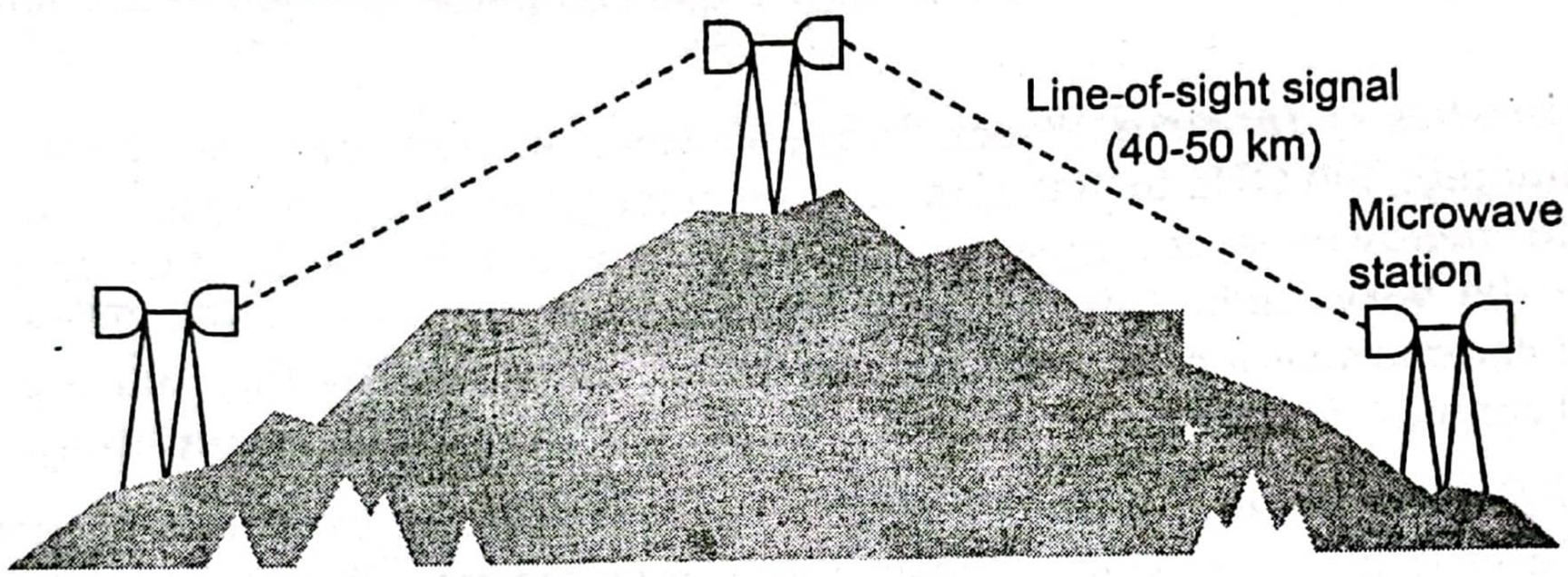


Fig. 15.7. Microwave transmission

Challenges and Future Prospects: While microwaves offer numerous advantages, they are susceptible to interference from atmospheric conditions and certain materials. As technology advances, overcoming these challenges becomes imperative. The ongoing evolution of microwave technology includes the development of advanced modulation schemes, beamforming techniques, and the exploration of higher frequency bands for improved communication and data transfer.

Applications of Microwaves

1. *Wireless Communication:* Microwaves underpin wireless communication networks, such as cellular networks and Wi-Fi systems. Their ability to carry large amounts of data quickly and efficiently has contributed to the rapid growth of digital communication.
2. *Satellite Communication:* Communication satellites use microwaves to transmit signals across vast distances, enabling global connectivity for television broadcasting, internet access, and long-distance communication.
3. *Radar Systems:* Microwaves are fundamental to radar technology, which detects and locates objects based on the reflection of microwave signals. Radars serve various applications, from weather forecasting and air traffic control to military defense and navigation.
4. *Microwave Ovens:* The absorption of microwaves by water molecules generates heat, making microwaves a cornerstone of microwave oven technology. This innovation has revolutionized food preparation and cooking times.
5. *Scientific Research:* In scientific research, microwaves are employed in microwave

CamScanner

spectroscopy, which provides insights into molecular structures and properties.

6. *Wireless Backhaul:* Microwaves are used for wireless backhaul, connecting cellular base stations and providing high-capacity data links between network elements.
7. *Remote Sensing:* Microwaves play a role in remote sensing, contributing to Earth observation, weather prediction, and environmental monitoring.

Infrared (IR)

Introduction: Infrared (IR) waves have wavelengths longer than visible light but shorter than microwaves. An infrared transmission technology is used for short-range wireless communication such as data transfer between two cell phones, TV remote controls, data transfer between a computer and cell phone resides in the same closed area.

Characteristics of Infrared Waves: Infrared waves exhibit frequencies ranging from approximately 300 GHz to 400 THz, with wavelengths falling between those of visible light and microwaves. It supports high bandwidth, and hence the data transmission rate will be very high. Infrared waves cannot penetrate the walls. Therefore, the infrared communication in one room cannot be interrupted by the nearby rooms. An infrared communication provides better security with minimum interference.

Thermal Emission and Sensing: One of the most significant aspects of infrared waves is their interaction with heat. All objects emit infrared radiation in proportion to their temperature. This phenomenon forms the basis for thermal imaging, a technology that captures the heat signatures of objects and environments. Infrared cameras convert these emissions into visible images, enabling applications in surveillance, building inspections, and even medical diagnostics.

Challenges and Considerations: While infrared waves offer numerous benefits, they also face challenges. Atmospheric absorption can limit the propagation of infrared signals over long distances. Hence, infrared communication is unreliable outside the building because the sun rays will interfere with the infrared waves. Additionally, the accuracy of certain applications, like medical diagnostics, hinges on calibration and proper interpretation of infrared data.

Future Prospects: As technology continues to advance, so does the potential of infrared waves. Improved sensor technologies, data processing techniques, and miniaturization are expanding the horizons of infrared applications. In fields such as medicine, environmental monitoring, and security, the continued evolution of infrared technology promises to deliver even greater accuracy and utility.

Applications of Infrared Waves

1. *Communication:* Infrared waves enable short-range wireless communication between devices, such as remote controls, data transmission between laptops and printers, and certain forms of indoor wireless networking.
2. *Remote Sensing:* Infrared sensors on satellites and aircraft facilitate remote sensing, contributing to weather forecasting, environmental monitoring, and resource management. Infrared imagery helps scientists and policymakers make informed decisions about climate, land use, and disaster response.

CamScanner

3. *Security Systems:* Infrared motion detectors play a pivotal role in security systems, detecting changes in infrared radiation caused by movement. These detectors are used in homes, businesses, and public spaces to trigger alarms and enhance security measures.
4. *Thermal Imaging:* Infrared cameras are employed for non-contact temperature measurement and thermal imaging. This technology finds applications in firefighting, search and rescue operations, industrial inspections, and medical diagnostics, enabling the identification of anomalies and potential issues.
5. *Medical Applications:* Infrared radiation is harnessed for various medical applications, including infrared thermography for identifying anomalies in the body's heat distribution, as well as laser-based therapies for pain management, tissue healing, and surgical procedures.

Various Types of Networks

There are many types of computer networks, but the two most common types of networks are:

1. LAN (Local Area Network),
2. WAN (Wide Area Network), and

LAN (Local Area Network)

A local area network is a privately owned communications network that serves users within a confined geographical area, connected by a length of wire, cable, or optical fiber, and run by special networking software. The range is usually within a kilometer—perhaps one office, one building, or a group of buildings close together, as a university campus. The LAN is generally owned by the company that is using it.

The major characteristics of LANs are:

1. *Limited geographic area:* LANs cover a relatively small area, such as a building, office, or campus.
2. *High data transfer rates:* LANs provide fast communication and data sharing between devices within the network.
3. *Shared resources:* LANs allow devices to share resources like printers, storage devices, and internet connections.
4. *Local administration and management:* LANs are managed by local administrators for maintenance, security, and user management.
5. *Ethernet, WiFi and TCP/IP:* Ethernet is the common technology used for wired LAN connections and WiFi is a wireless extension of a LAN. WiFi technology allows devices to connect to a network wirelessly using radio waves. TCP/IP is the standard protocol suite for LAN communication.
6. *Security measures:* LANs incorporate security measures like firewalls, access control, encryption, and VPNs to protect against unauthorized access and data breaches.

LANs are crucial in facilitating local communication and resource sharing, making them essential for various applications, including home networking, office environments, educational institutions, and small to medium-sized businesses.

CamScanner

Topology of LAN

The physical layout of a network is called a *topology*. The six basic topologies for LAN are *star, ring, bus, mesh, tree* and *hybrid*. In a network topology, a component is called a *node*, which is usually a computer on a network. (The term *node* is also used to refer to any device connected to a network, including the server, computers, and peripheral devices such as printer.)

Star Topology

In a star network topology, each device is connected to a central unit. Any communication between one device and another goes though the central unit. The Fig. 15.8 shows a typical configuration using central server and peripherals. Each device is directly connected to the central server, which is a computer containing the interface cards and software to manage all data communication in this network. If a user at workstation 2 wants to print out a page, workstation 2 sends that request to the central server. This central server notifies workstation 2 when that task is complete.

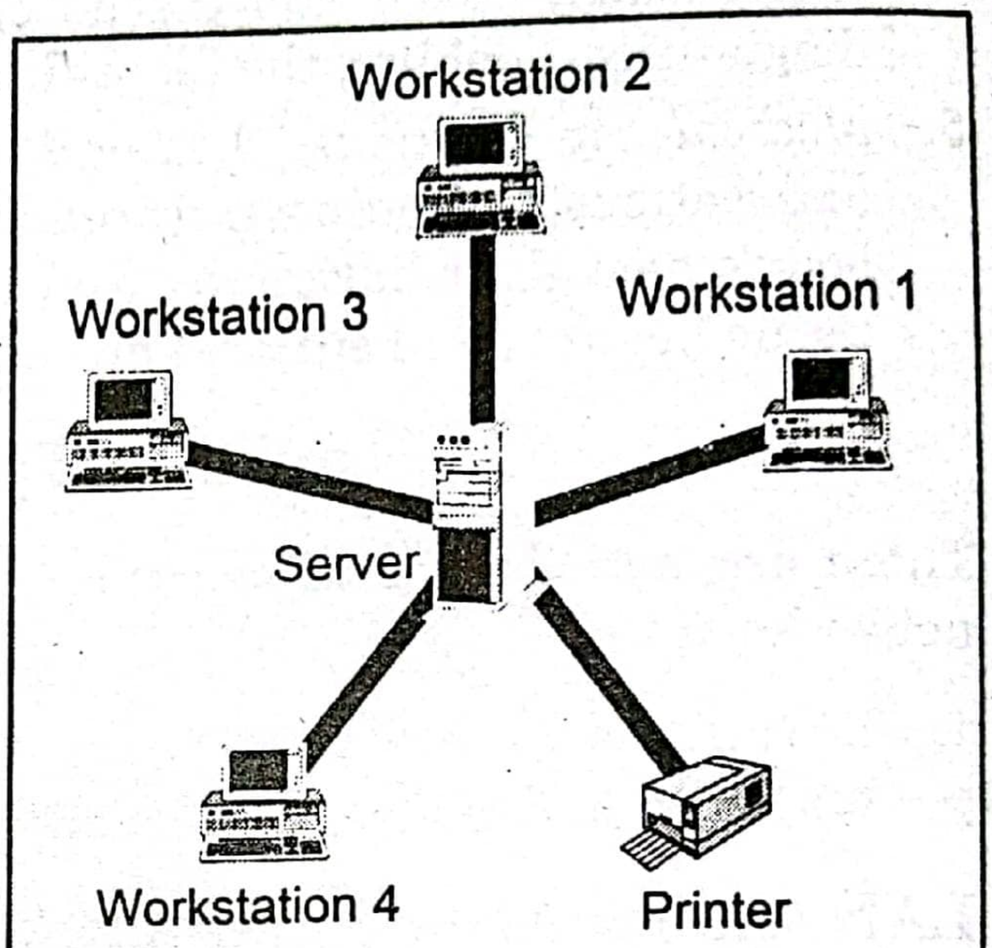


Fig. 15.8. In a star topology, all the computers in the network are attached to the central server. The central server can be a mainframe or a smaller computer.

Advantages of Star Topology

1. New terminals can be added to a network easily.
2. There is no chance of data collision because central server prevents collisions between messages.
3. If a connection is broken between any communications device and the central server, the rest of the devices on the network will continue operating.
4. Transmission delays between two terminals or between server and terminal do not increase by adding new nodes to the network because any two devices may be connected via two links only.

Disadvantages of Star Topology

1. The network crucially depends on the central server. If the central server fails, the entire network will stop.
2. Star topology is expensive because a separate cable is required for connecting each computer with the central server.

Ring Topology

In the ring network topology, multiple terminals and some peripherals are connected into a ring-like structure (see Fig. 15.9). All communication between terminals follows a clockwise or anticlockwise pattern. The message goes from terminal to terminal until the designated device is reached. There is no central server.

CamScanner

Advantages of Ring Topology

1. In a ring network messages flow in only one direction. Thus, there is no danger of collision.
2. It is more reliable than a star network because communication is not dependent on a single host computer.

Disadvantages of Ring Topology

1. If the ring is broken, the entire network stop working.
2. It is difficult to troubleshoot the ring network.
3. New computer or terminal cannot be added to network easily.
4. Communication delay is directly proportional to the number of computers in the network.

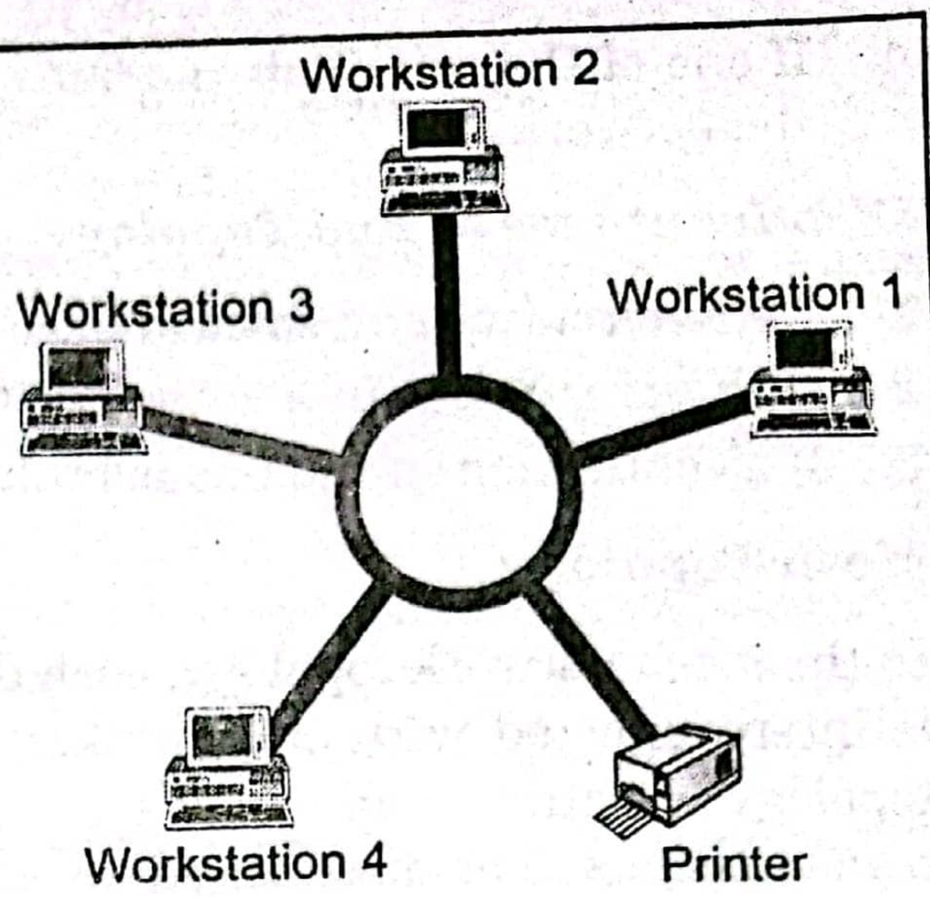


Fig. 15.9. In a ring topology, the computer nodes are attached in a circle without a central server.

Bus Topology

In a bus network topology, each device is connected to a common cable (see Fig. 15.10). Each component must have its own interface device, usually a circuit board or card, which plugs into one of the expansion slots. The card contains the hardware and software necessary to access the network. All communication takes place on the common cable or bus. The data are sent down the bus and are available to all devices. Each message must contain information identifying the destination device. If some messages collide, the device waits and tries to retransmit. There is no central server.

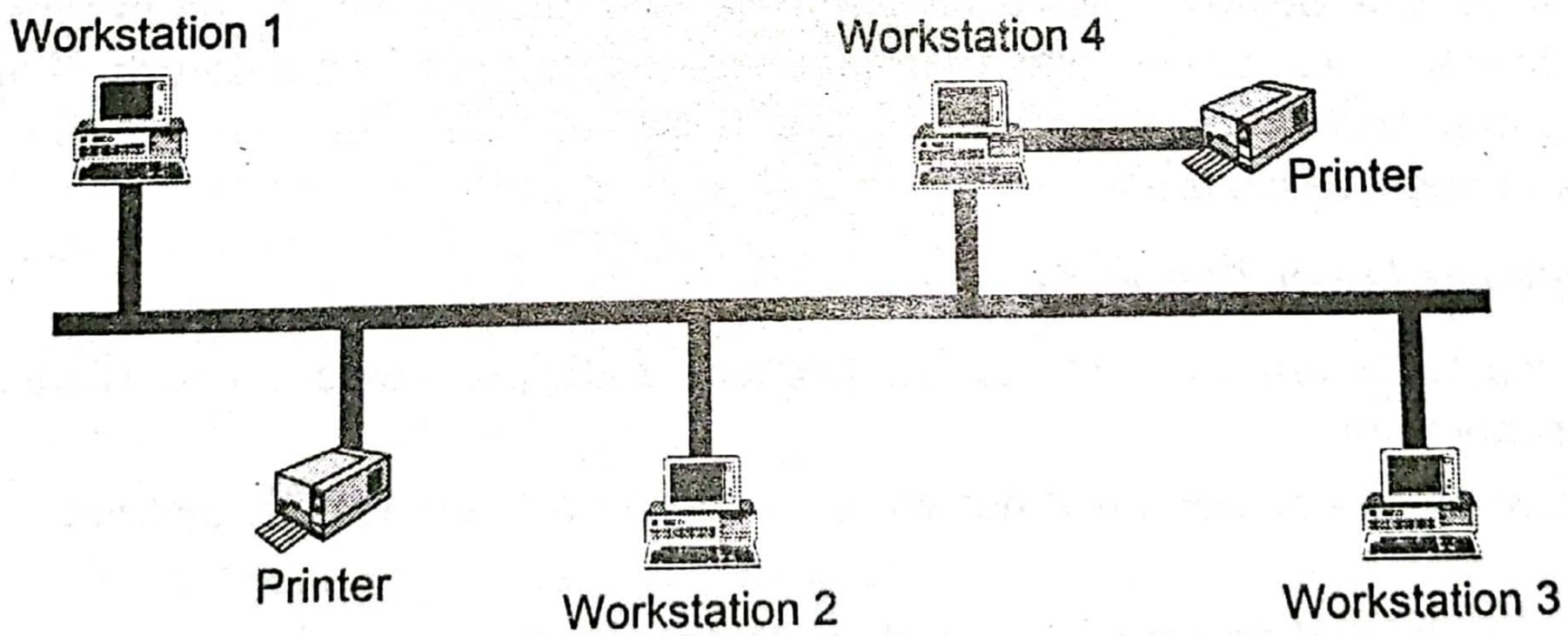


Fig. 15.10. In a bus topology, the failure of one network computer does not affect the other network computers.

Advantages of Bus Topology

1. This type of network is simple and easy to understand.
2. A bus network may be organized as *client / server* or *peer-to-peer* network.
3. New terminals can be added to a network easily.

CamScanner

4. If one of the terminals becomes defective it does not effect on other computers of the network.

Disadvantages of Bus Topology

1. Extra circuitry and software are needed to avoid collisions between data.
2. If an error arises in a network, it is not easy to detect.
3. If a connection in the bus is broken, the entire network may stop working.

Mesh Topology

In the mesh network topology, each device is interconnected with one another. This topology setup allows for most transmissions to be distributed even if one of the connections goes down. It is a topology commonly used for wireless networks. Fig. 15.11 is a visual example of a simple computer setup on a network using a mesh topology.

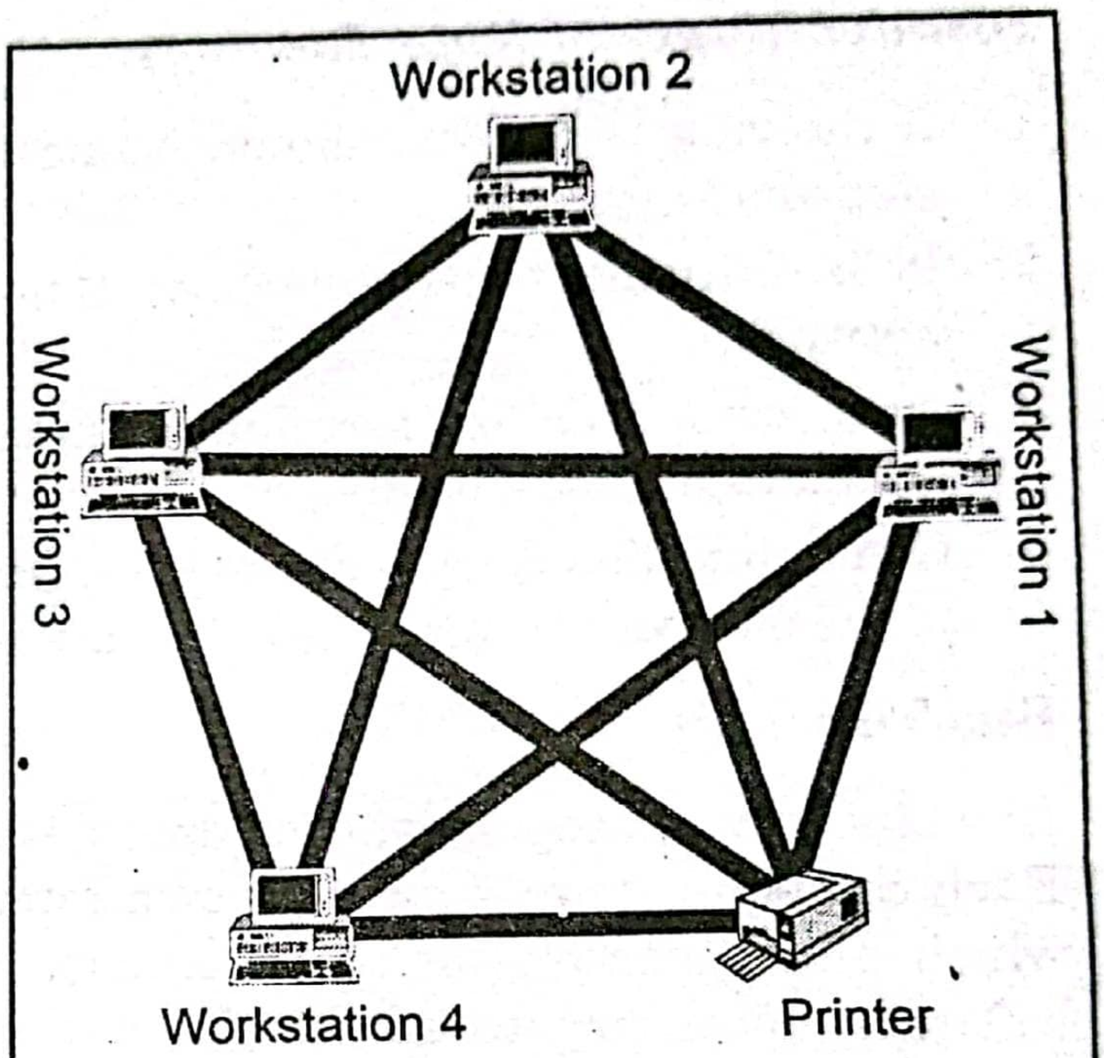


Fig. 15.11. In a mesh topology, each device is interconnected with one another.

There are two forms of this topology: full mesh and a partially-connected mesh. In a *full mesh topology*, every computer in the network has a connection to each of the other computers in that network. The number of connections in this network can be calculated using the formula $n(n-1)/2$, where n is the number of computers in the network. In a *partially-connected mesh topology*, at least two of the computers in the network have connections to multiple other computers in that network. It is an inexpensive way to implement redundancy in a network. If one of the primary computers or connections in the network fails, the rest of the network continues to operate normally.

Advantages of Mesh Topology

1. Manages high amounts of traffic, because multiple devices can transmit data simultaneously.
2. A failure of one device does not cause a break in the network or transmission of data.
3. Adding additional devices does not disrupt data transmission between other devices.

Disadvantages of Mesh Topology

1. The cost to implement is higher than other network topologies, making it a less desirable option.
2. Building and maintaining the topology is difficult and time consuming.
3. The chance of redundant connections is high, which adds to the high costs and potential for reduced efficiency.

CamScanner

Tree Topology

A *tree topology* is a special type of structure where many connected elements are arranged like the branches of a tree.

In computer networks, a tree topology is also known as a *star bus topology*. It combines characteristics of both a bus topology and a star topology. It consists of groups of star-configured workstations connected to a linear bus backbone cable. Fig. 15.12 is an example network diagram of a tree topology, where the central nodes of two star networks are connected to one another.

In Fig. 15.12, if the main cable (called *trunk*) between the two star topology networks fails, those networks would be unable to communicate with each other. However, computers on the same star topology would still be able to communicate.

Tree topologies allow for the expansion of an existing network, and enable schools and corporate to configure a network to meet their needs.

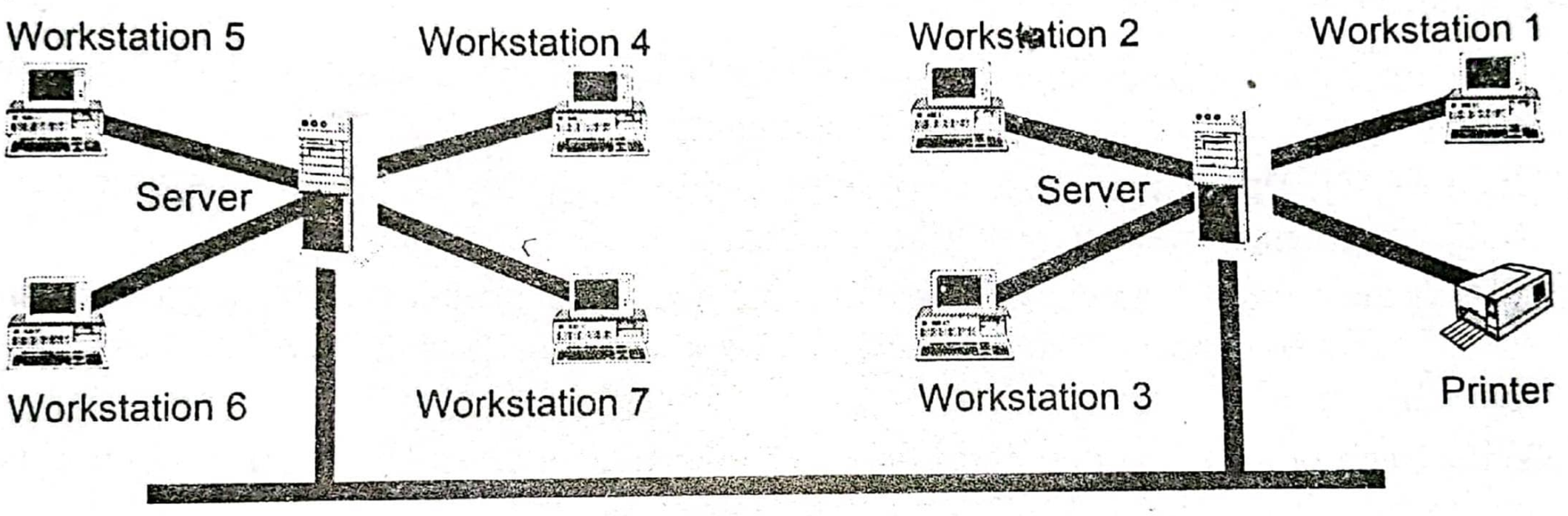


Fig. 15.12. A tree topology combines characteristics of both a bus topology and a star topology.

Advantages of Tree Topology

1. Failure of one node never affects the rest of the network.
2. Node expansion is fast and easy, this topology provides high scalability.
3. It provides easy maintenance and easy fault identification can be done.
4. It is easy to manage and maintain.

Disadvantages of Tree Topology

1. It is heavily cabled topology.
2. If the backbone line breaks, the entire segment goes down.
3. More difficult to configure and wire than other topologies.

Hybrid Topology

The choice of topology for installing a computer network depends upon a combination of factors such as:

1. reliability of the entire system
2. expandability of the system
3. cost involved

CamScanner

4. availability of communication lines
5. delays involved in routing information from one terminal to another.

Different network configurations have their own advantages and limitations. Hence, a pure form of any of these three basic topologies is seldom found in practice. Most computer networks are *hybrids*—combination of these topologies. For example, a university campus might use a bus network to connect different departments and star and/or ring networks within certain departments.

WAN (Wide Area Network)

A wide area network is a communications network that covers a wide geographical area, such as a province or a country. These networks allow multiple users to access a variety of host computers simultaneously through the same physical medium, while separating each user's session so that no user is aware of another on the network. It also enable LANs to communicate with each other.

The major characteristics of WANs are:

1. The network operates beyond the local LAN's geographic scope. It uses the services of carriers provided by communication companies such as PTCL.
2. In a WAN, the transmission medium used are normally public systems such as telephone lines, microwaves and satellite links.
3. WANs rarely shut down, but because the enterprise must pay for services used, it might restrict access to connected workstations. Not all WAN services are available in all locations.
4. Wide area networks operate at speeds much higher than the limit of normal voice-grade telephone lines.

Table 15.1. LAN versus WAN comparison

	LAN	WAN
Stands For	Local Area Network	Wide Area Network
Covers	Local areas only (*e.g.*, homes, offices, schools)	Large geographic areas (*e.g.*, cities, states, nations)
Definition	LAN (Local Area Network) is a computer network covering a small geographic area, like a home, office, school, or group of buildings.	WAN (Wide Area Network) is a computer network that covers a broad area (*e.g.*, any network whose communications links cross metropolitan, regional, or national boundaries over a long distance).
Speed	High speed (1-100 Gbps)	Less speed (100 Mbps - 1 Gbps)
Data transfer rates	LANs have a high data transfer rate.	WANs have a lower data transfer rate compared to LANs.
Technology	Use certain connectivity technologies, primarily Ethernet and WiFi (Wireless LAN).	WANs use technologies like Leased Lines, MPLS (Multiprotocol Label Switching), ATM (Asynchronous Transfer Mode), and Frame Relay for connectivity over long distances.

CamScanner

Connection	One LAN can be connected to other LANs over any distance via telephone lines and radio waves.	Computers connected to a wide-area network are often connected through public networks, such as the telephone system. They can also be connected through leased lines or satellites.
Components	Use OSI Layer 2 devices like switches and bridges. Layer 1 devices like hubs and repeaters.	Use OSI Layers 3 devices Routers, Multi-layer Switches and Technology specific devices like ATM or Frame-relay Switches etc.
Fault Tolerance	LANs tend to have fewer problems associated with them, as there are smaller number of systems to deal with.	WANs tend to be less fault tolerant as they consist of large number of systems.
Data Transmission Error	Experiences fewer data transmission errors	Experiences more data transmission errors as compared to LAN
Ownership	Typically owned, controlled, and managed by a single person or organization.	WANs are not owned by any one organization but rather exist under collective or distributed ownership and management over long distances.
Set-up costs	If there is a need to set-up a couple of extra devices on the network, it is not very expensive to do that.	For WANs since networks in remote areas have to be connected the set-up costs are higher. However, WANs using public networks can be setup very cheaply using just software (VPN etc.).
Maintenance costs	Because it covers a relatively small geographical area, LAN is easier to maintain at relatively low costs.	Maintaining WAN is difficult because of its wider geographical coverage and higher maintenance costs.
Bandwidth	High bandwidth is available for transmission.	Low bandwidth is available for transmission.
Congestion	Less congestion	More congestion

Typical Networking Devices

Regardless of the type of media, each computer on the network still needs a hardware component to control the flow of data. *Networking hardware*, also known as *network equipment*, are physical devices which are required for communication and interaction between devices on a computer network. Specifically, they "securely transfer data as fast as they can" in a computer network. Here are some examples of typical networking equipment:

Network Interface Cards (NICs)

A network card, also known as a *network interface card* (NIC), is a computer hardware component that allows the transmission of data over a cable network, which connects

CamScanner

various computers and other devices such as printers.

The network card coordinates the transmission and receipt of data, instructions, and information to and from the computer or device containing the network card.

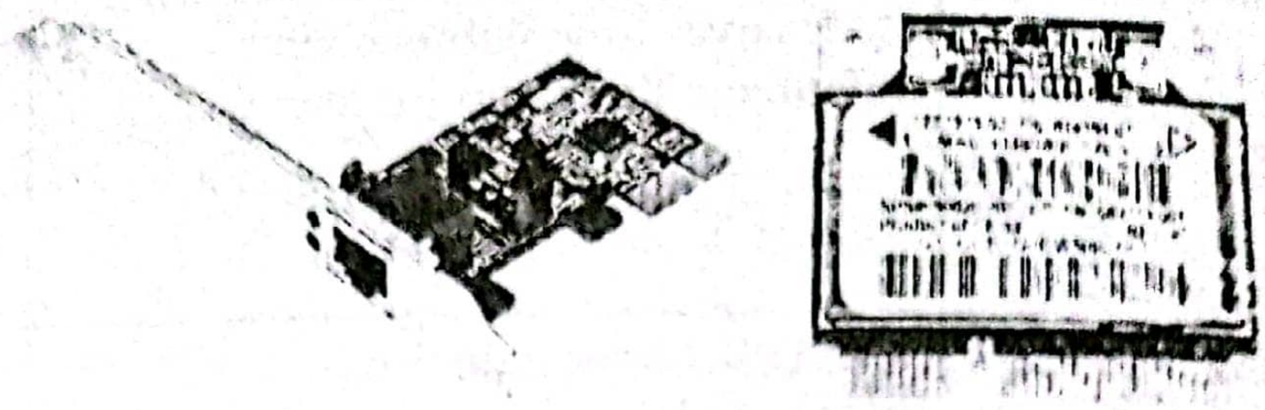


Fig. 15.13. Network cards are available for both desktop and notebook computers

Network cards are available in a variety of styles (Fig. 15.13). A network card for a desktop computer is an adapter card that has a port to which a cable connects. A network card for laptop is in the form of a mini-PCI card.

Network cards that provide wireless data transmission also are available. This type of card, called a *wireless network card*, often has an antenna. Sometimes the antenna is detachable, allowing the user to position it in a location with the best signal strength. Users also can install an amplifier to increase and/or stabilize the signal strength. Some network cards include support for both wired and wireless networks.

Modem

The simplest forms of data transmission involve the connection of terminals to a computer via a telephone line. A modem is a network hardware device that connects a computer to the Internet over a telephone line.

But the problem here is that the computer generates data in the form of *digital* signals whereas the telephone lines used for data communication in computer networks are usually meant for carrying *analogue* signals. When digital data are to be sent over an analogue facility, the digital signals must be converted to analogue form. The technique by which a digital signal is converted to its analogue form is known as *modulation*. The reverse process, that is the conversion of analogue signal to its digital form, at a destination device, is called *demodulation*.

The modulation can be done in different ways. Two of the most common methods used are amplitude modulation and frequency modulation. *Amplitude modulation (AM)* changes the amplitude of the signal while keeping the frequency fixed. *Frequency modulation (FM)* changes the frequency of the signal while keeping the amplitude fixed. (AM and FM are used to designate common radio bands.)

The process of modulation and demodulation, that is, the conversion of digital data to analogue form and vice-versa, is carried out by a special device called a modem (short for MOdulator-DEModulator). Hence, when an analogue facility is used for data communication between two digital devices (say a terminal and a computer), two modems are required, one near each digital device. As shown in Fig. 15.14, the digital signal generated at the terminal is converted to analogue form by the modulator of the modem placed near it. The analogue signal is transmitted through the telephone lines and then converted to digital form by the demodulator of the modem placed near the computer. This digital data is processed by the computer. The processed digital information is modulated to analogue form and returned via the telephone line to the

CamScanner

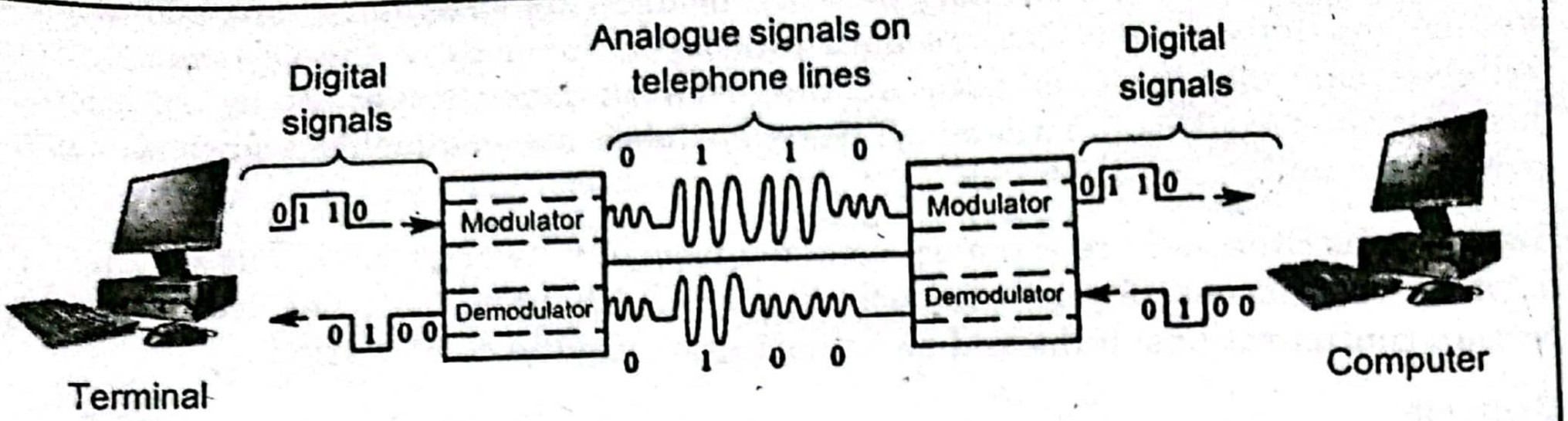


Fig. 15.14. Illustrating the use of modems in data communications

terminal where the analogue signals are demodulated to digital form for display on the terminal.

Hubs

A Hub, also called repeater hub, is a network hardware device for connecting multiple Ethernet devices together and making them act as a single network segment. It has multiple input/output (I/O) ports, in which a signal introduced at the input of any port appears at the output of every port except the original incoming. A repeater hub also participates in collision detection, forwarding a jam signal to all ports if it detects a collision. A hub is a half-duplex device, meaning it transmits data in both directions but only in one direction at a time. With hubs, bandwidth is shared by all components. A hub works at the physical layer (layer 1) of the OSI model.

In addition to standard RJ45 ports, some hubs may also come with a connector to allow connection to legacy network segments. Hubs are now largely obsolete, having been replaced by network switches except in very old installations or specialized applications.

Bridges

A network bridge is a computer networking device that creates a single aggregate network from multiple communication networks or network segments. This function is called *network bridging*. For instance, similar local area networks can be joined together to create larger area networks.

Bridging is distinct from routing. Routing allows multiple networks to communicate independently and yet remain separate, whereas bridging connects two separate networks as if they were a single network. In the OSI model, bridging is performed in the data link layer (layer 2). If one or more segments of the bridged network are wireless, the device is known as a *wireless bridge* and the function as *wireless bridging*.

Switches

A switch is a device that connects computers to a network. Unlike a hub, a network switch forwards data only to a computer that is the intended recipient, rather than broadcasting the same data out of each of its ports. A switch is a full-duplex device, meaning data is transmitted back and forth at the same time, which improves the performance of the network. Switches allow each component full use of the bandwidth.

CamScanner

A network switch is a multiport network bridge that uses hardware addresses to process and forward data at the data link layer (layer 2) of the OSI model. Some switches can also process data at the network layer (layer 3) by additionally incorporating routing functionality. Such switches are commonly known as layer-3 switches or multilayer switches.

Switches for Ethernet are the most common form of network switch, but switches also exist for other types of networks including Fiber Channel. Switches are used only in certain configurations; hubs and switches can be used in combination.

Routers

A router is a networking device that connects multiple computers or other routers together and transmits data packets to its correct destination on a network. A router can be used on any size of network. Routers perform the *traffic directing* functions on the Internet. On the largest scale, routers along the Internet backbone forward data packets to their destination using the fastest available path. Data sent through the internet, such as a web page or email, is in the form of data *packets*. A packet is typically forwarded from one router to another router through the networks that constitute an internetwork until it reaches its destination node. It works on OSI layer 3.

A router is connected to two or more data lines from different networks. When a data packet comes in on one of the lines, the router reads the network address information in the packet to determine the ultimate destination. Then, using information in its *routing table*, it directs the packet to the next network on its journey.

The most familiar type of routers are *home and small office routers* that allow multiple computers to share a single high-speed Internet connection such as through a cable modem or DSL modem (see Fig. 15.15). These routers connect from 2 to 250 computers. To prevent unauthorized users from accessing files and computers, many routers are

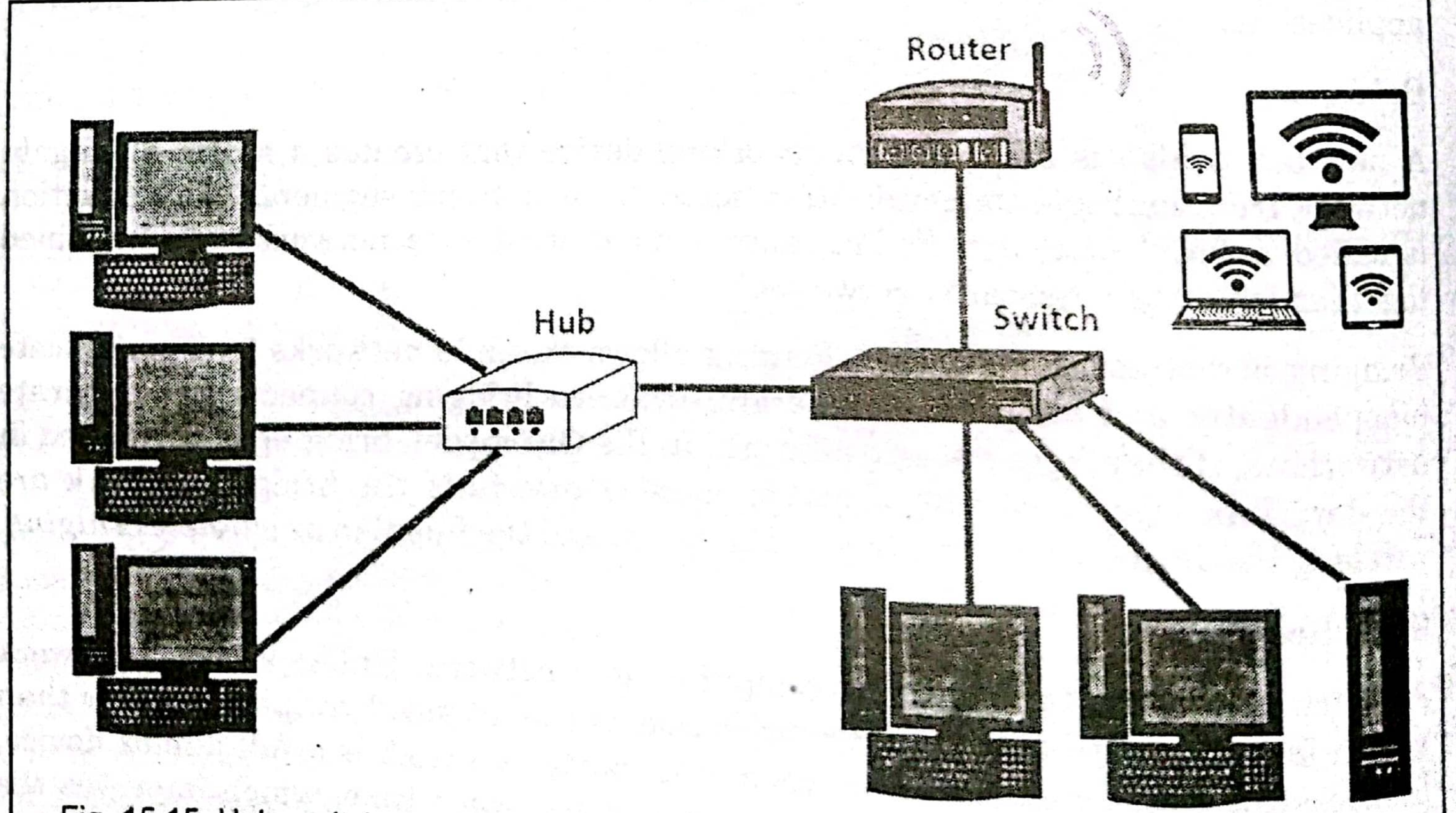


Fig. 15.15. Hub, switch or router is a central point that connects several devices in a network

CamScanner

protected by a built-in firewall, called a *hardware firewall*. Some also have built-in antivirus protection. Routers also support wireless communications, eliminating the need for a separate wireless access point in a wireless network. Some routers also include additional functionality such as including a built-in print server. Today's routers or combination wireless access point/routers are easy to configure and secure against unauthorized access. More sophisticated routers, such as enterprise routers, connect large business or ISP networks up to the powerful core routers that forward data at high speed along the optical fiber lines of the Internet backbone.

Gateways

A gateway is the piece of networking hardware that permits communication between dissimilar networks —for instance, between a LAN and a WAN or between two LANs based on different network operating systems or different layouts. Till host is configured with default gateway address, every packet should have default gateway address. A default gateway address is the address of gateway device. If packet does not find its destination address in local network, then it would take the help of gateway device to find the destination address in remote network. A gateway device knows the path of remote destination address. If require, it also change the encapsulation of packet so it can travel in other network to get its destination address. Gateways are distinct from routers or switches in that they communicate using more than one protocol and can operate at any layer of the 7 layer OSI model of networking.

Gateways can be hardware, software, or a combination of the two. Therefore, the term gateway can also loosely refer to a computer or computer program configured to perform the tasks of a gateway, such as in a default gateway.

Network Communications Standards

Standards are rules acknowledged by participating members of an organization and the industry they represent. One of the challenges of any standards organization is to agree on what a standard should be. In the case of computer networks there are hundreds of ways by which any computer could be connected and communicate with any other computer, any of which could serve as a standard.

Because manufacturers have long produced devices that use a variety of transmission techniques, the telecommunications industry has adopted *standards* called *protocols* to rectify the problem of conflicting procedures.

Communications protocols describe how data are transmitted in a computer network. Protocols specify how devices will physically connect to a network, how data will be packaged for transmission, how receiver devices will acknowledge signals from sender devices, how errors will be handled, and so on. Just as people need an agreed-upon set of rules to communicate effectively, machines also need a common set of rules to help them get along with one another.

To solve the problems of incompatibility and ensure that hardware and software components can be integrated into any network, various organizations such as IEEE, ISO, ASCII and ITU develop network standards. A network standard defines guidelines that specify the way computers access the medium to which they are attached, the type(s) of medium used, the speeds used on different types of networks

CamScanner

and the type(s) of physical cable and/or the wireless technology used.

Internet communication protocols are published by the Internet Engineering Task Force (IETF). The IEEE handles wired and wireless networking, and the International Organization for Standardization (ISO) handles a family of information exchange standards. The ITU-T handles telecommunication protocols and formats for the public switched telephone network (PSTN). As the PSTN and Internet converge, the standards are also being driven towards convergence.

ASCII, in full *American Standard Code for Information Interchange*, is a standard data-encoding format for electronic communication between computers. ASCII assigns standard numeric values to letters, numerals, punctuation marks, and other characters used in computers. The ASCII Transmission Control Protocol, or ASCII TCP is a query-response/question and answer communication protocol in which a host PC uses ASCII characters to send commands to a device, and receive responses back from the device. ASCII TCP is a connection-based protocol, in which the client establishes a connection with the server and the server waits for an incoming connection from the client. Once a connection is established, the ASCII command set is used to configure devices, send data to devices and to read data and status information back from devices. This occurs until the client closes the connection.

Network Architectures

Network architectures provide a structured framework for understanding and designing computer networks. Three prominent network architectures are *the OSI Model, the TCP/IP Model,* and *the Internet Model*. Each of these models defines a set of protocols and layers that guide the process of data communication and exchange in a networked environment.

The OSI Model

One communications standard created by the ISO is called the *Open Systems Interconnection* (OSI) model that simulates the communications process using seven layers, each with its own set of protocols. The purpose of the OSI model is to enable any vendor's computer system to share data with any other vendor's system in an open networking environment. This also prevents protocol standards with overlapping functionality and allows clear definition of the responsibilities of a protocol at the different levels (layers). Fig. 15.16 shows the structure of the OSI model. A description of each of the seven layers, from bottom to top, are follows:

Layer 1. Physical Layer: The physical layer controls the electrical, mechanical, and functional transmission of bits over the data circuits.

Layer 2. Data Link Layer: The data link layer detects and compensates for transmission errors and ensures that information sent by high-speed transmitters is properly received by slow receivers.

Layer 3. Network Layer: The network layer determines how information is routed between computers and within and between individual networks. It also handles software interfaces between networks, including networks with different protocols.

Layer 4. Transport Layer: The transport layer specifies the rules for information

CamScanner

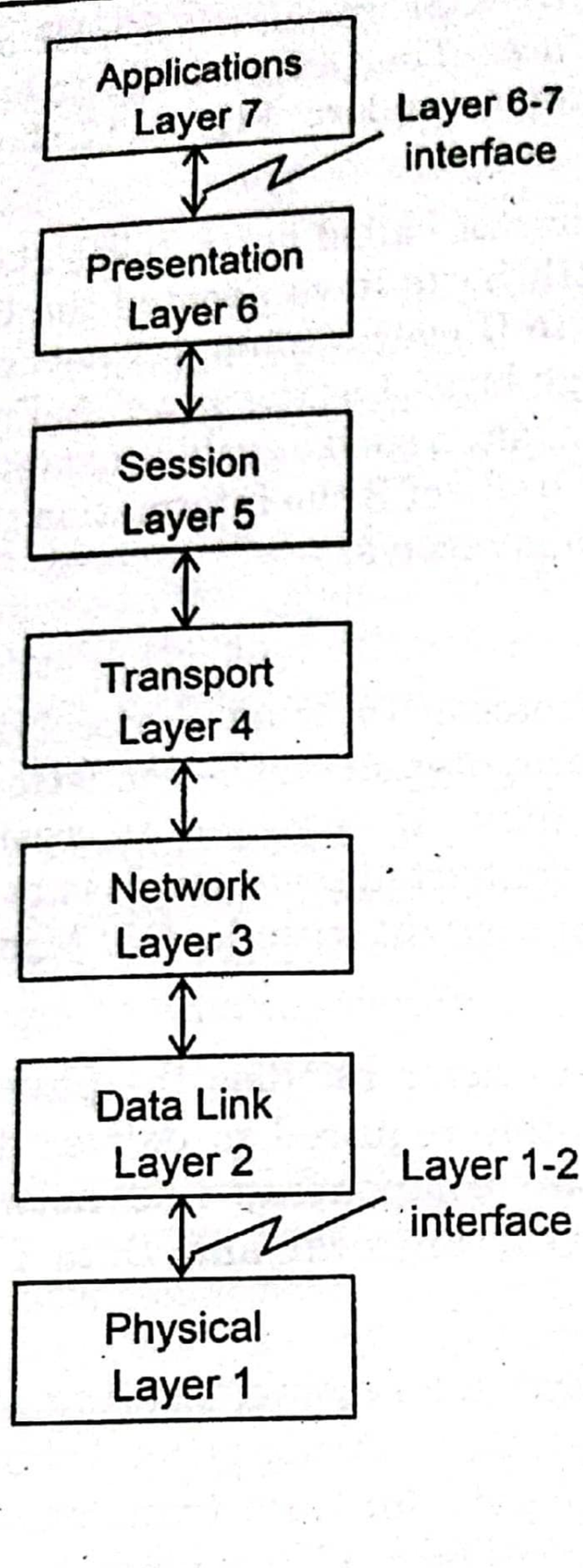


Fig. 15.16. OSI model with interfaces

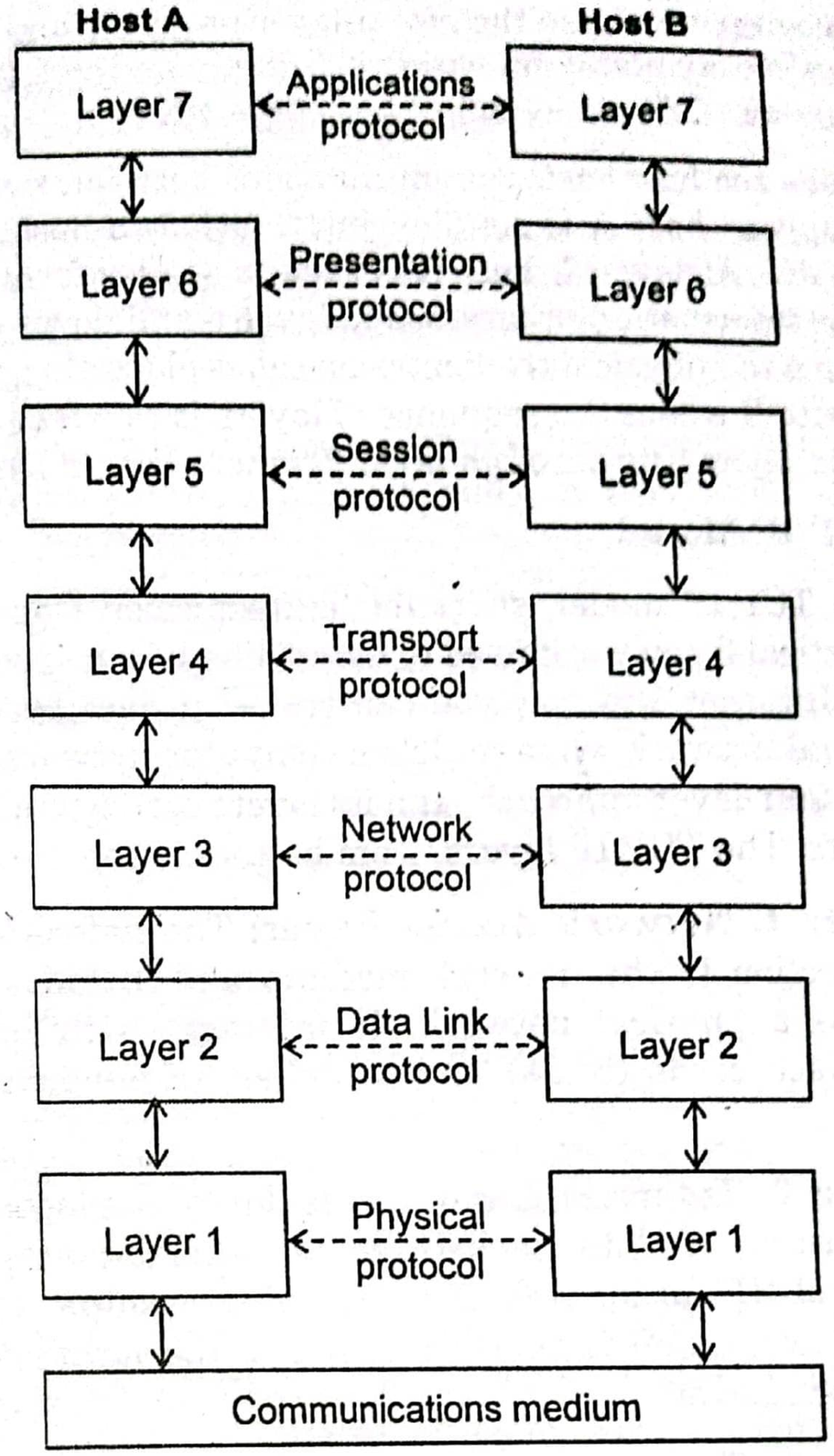


Fig. 15.17. OSI model with protocols

exchange and manages end-to-end delivery of information within and between networks, including error recovery. It also controls information flow—for example, multiple data streams on a single channel.

Layer 5. Session Layer: The session layer controls the dialog between two computers, managing file transfers and putting checkpoints into a data stream to allow portions of files to be retransmitted as needed.

Layer 6. Presentation Layer: The presentation layer supplies transparent communications by masking the differences in unlike data formats such as the ASCII and EBCDIC character codes, and performs data compression and encryption.

Layer 7. Applications Layer: The applications layer supplies functions for particular applications such as file transfer, remote access, and virtual terminals.

CamScanner

When sending or receiving information according to the OSI model, the sender and receiver must use the seven-layer protocols and interfaces. The OSI model is an open system available on communications systems from many vendors. Fig. 15.17 shows how the OSI model might be represented.

Let's see how basic communications between two machines, called hosts, takes place. Suppose host A is sending information to host B. Both hosts have adopted the OSI model. At layer 7, host A begins to send information to B over a computer network. The information progresses to layer 6 and down through layer 1 at the site of A. From site A the physical transmission takes place along the medium such as cable or satellite to site B where the sequence of layers is reversed. At the site of B the information goes from layer 1 up through layer 7, where the information is receive.

TCP/IP Model

The TCP/IP model, short for Transmission Control Protocol/ Internet Protocol, is a practical framework used to determine how a specific computer should be connected to the Internet and how you can transmit data between them. It helps you to create a virtual network when multiple computer networks are connected together. It is based on a four-layer approach, and its layers correspond to some extent with the OSI Model's layers. The TCP/IP layers, from bottom to top, are:

Layer 1. Network Access Layer: The network access layer handles the physical connection to the network medium and includes protocols required to deliver data across a physical network. It interacts with hardware components like network interface cards (NICs) and switches. (Equivalent to OSI Physical and Data Link Layers)

Layer 2. Internet Layer: This layer manages logical addressing, routing, and forwarding of data packets across interconnected networks. It uses the Internet Protocol (IP) to identify devices and determine the best path for data transmission. This information is then conveyed to the network access layer. (Equivalent to OSI Network Layer)

Layer 3. Transport Layer: The transport layer is responsible for end-to-end communication, data segmentation, error detection, and flow control. It includes protocols like Transmission Control Protocol (TCP) and User Datagram Protocol (UDP). (Equivalent to OSI Transport Layer)

Layer 4. Application Layer: This layer represents the application or user interface that interacts with the network. It includes protocols for various services such as email (SMTP), file transfer (FTP), and web browsing (HTTP). (Equivalent to OSI Session, Presentation, and Application Layers)

Both the TCP/IP and OSI models provide logical ways of networking, as well as the processing of information using a layered system. In both systems, each layer has a specific function. This makes it easier to pinpoint where issues are occurring in the event of a failure.

For example, in both models, you can determine if the data is not being transmitted properly to a hardware device by isolating potential problems in the data link layer

CamScanner

(OSI) or the network access layer (TCP/IP).

The biggest difference between the two models is that the OSI model segments multiple functions that the TCP/IP model groups into single layers. This is true of both the application and network access layers of the TCP/IP model, which contain multiple layers outlined within the OSI model.

This is a significant difference because it can make it more difficult to troubleshoot issues or enhance performance when you are using the TCP/IP model. With the OSI model, for example, you can focus specifically on the application layer, presentation layer, or session layer to figure out why data is not coming out the way you expect.

With the TCP/IP model, on the other hand, the functions of these three layers are all combined in the application layer. In other words, while someone may say, "There is a problem in the application layer," someone thinking in terms of the OSI model may get confused, wondering, "OK, but which part of the application layer?"

TCP/IP is a practical model that addresses specific communication challenges and relies on standardized protocols. In contrast, OSI serves as a comprehensive, protocol-independent framework designed to encompass various network communication methods. It is known for its efficiency and simplicity, making it a popular choice for network communication in both local and global contexts.

Internet Model

The Internet Model, also known as the *Simplified TCP/IP Model* or the *DoD Model* (Department of Defense Model), is a framework that forms the foundation of modern computer networking, enabling the communication and data exchange that underpins the global Internet. Developed alongside the creation of the Internet in the 1970s, the Internet Model provides a systematic approach to understanding and implementing network communication protocols.

Similar to other networking models, the Internet Model employs a layered structure that divides network functions into distinct layers. Each layer is responsible for specific tasks and interacts with adjacent layers to ensure efficient and reliable data transmission. While the Internet Model is often simplified into three layers, it is important to note that these layers align with and expand upon the more comprehensive OSI Model.

Layer 1. Network Access Layer: Equivalent to the TCP/IP model's "Network Access Layer," this layer handles the physical connection to the network medium and includes data link control and addressing.

Layer 2. Internet Layer: Equivalent to the TCP/IP model's "Internet Layer," this layer manages logical addressing, routing, and forwarding of data packets using the Internet Protocol (IP).

Layer 3. Transport/Application Layer: This combined layer corresponds to the TCP/IP model's "Transport Layer" and "Application Layer." It handles end-to-end communication, data segmentation, error detection, and flow control, as well as user applications and services

CamScanner

A notable feature of the Internet Model is its close alignment with the TCP/IP protocol suite. TCP/IP, a collection of protocols that govern internet communication, mirrors the layer structure of the Internet Model, emphasizing its practicality and real-world applicability.

The Internet Model's modular design and flexibility have contributed to its widespread adoption and adaptability. Its layers promote interoperability, allowing different systems and technologies to communicate seamlessly. This interoperability, in turn, has fueled the rapid expansion and globalization of the internet, enabling the interconnectedness of people, devices, and information across the world.

CamScanner

Questions

15.1. Identify the basic elements of a communication system and the purpose of each.
15.2. Differentiate between simplex, half-duplex, and full-duplex modes of data transmission.
15.3. Which mode of data transmission is suitable for communication between a terminal and a computer?
15.4. "A full-duplex line is faster since it avoids the delay that occurs in a half-duplex circuit". Explain this statement.
15.5. What are half-duplex channels commonly used for?
15.6. What are full-duplex channels commonly used for?
15.7. Explain the term bandwidth.
15.8. What is a twisted-pair cable? In what situations are they suitable for use in data communications?
15.9. What is a coaxial cable? Give some of its practical uses.
15.10. What is a fiber optics? How is it used for data communications?
15.11. What carries data over fiber optic cables and how is it produced?
15.12. What are the advantages of fiber optics?
15.13. Explain how microwave systems can be used for communication between two distant stations?
15.14. Why are microwave relay stations placed on top of tall buildings, towers, and mountains about 40-50 kilometres apart?
15.15. Identify several communications media that can be used over very long distances.
15.16. List and describe briefly the benefits that networks provide.
15.17. What is a LAN? What are its main objectives?
15.18. List and describe briefly the six common LAN topologies.
15.19. What is a hybrid network?
15.20. What are some common components used in LAN?
15.21. What are modems? What purpose do they serve in data communications?
15.22. What do you understand by modulation and demodulation?
15.23. What is a WAN? What are its main objectives?
15.24. Differentiate between a LAN and a WAN. Given one example of each.
15.25. Explain the need to set standards in communications networks.
15.26. What is a communication protocol? What are the normal functions performed b these protocols?
15.27. Describe the data link layer in the OSI model.
15.28. Briefly explain what TCP/IP model.

CamScanner

Objectives and MCQs

Each of the following questions has four alternative answers, of which one or more may be correct. Pick up the correct alternative/alternatives.

15.1. A mode of communications that allows information to travel in two directions but not simultaneously is called
- (a) full-duplex
- (b) half-duplex
- (c) half-simplex
- (d) simplex

15.2. Transmission permitting data to move only one way at a time is called
- (a) half-duplex
- (b) full-duplex
- (c) simplex
- (d) start/stop

15.3. Distortion in a signal is called
- (a) phase
- (b) noise
- (c) IP switch
- (d) amplitude

15.4. Microwave transmission, coaxial cables, and fiber optics are examples of
- (a) modems
- (b) routers
- (c) communication media
- (d) ring networks links

15.5. In a fiber optic cable, the core is surrounded by a second layer of glass called
- (a) yoke
- (b) cladding
- (c) outer-pipe
- (d) cap

15.6. A network that spans a large city is known as a(n)
- (a) ISDN
- (b) LAN
- (c) WAN
- (d) NIC

15.7. A network of geographically distant computers and terminals is called a(n)
- (a) bus
- (b) WAN
- (c) Internet
- (d) LAN

15.8. The physical layout of a LAN is called the

CamScanner

(a) topology
(b) contention
(c) link
(d) switch

15.9. Select the simplest topology from the following
(a) bus topology
(b) star topology
(c) token ring
(d) hybrid topology

15.10. A network that places all nodes on a single cable:
(a) star
(b) ring
(c) switched
(d) bus

15.11. Which of the following network configurations always uses a central server?
(a) bus
(b) star
(c) ring
(d) hybrid

15.12. Internet protocol is governed by
(a) EFT
(b) Internet
(c) TCP/IP
(d) EDI

15.13. A term used to describe the data-handling capacity of a given media is
(a) buffer
(b) routing
(c) bandwidth
(d) cache

15.14. The most important de facto organization involved in establishing communication standards and protocols is
(a) CCITT
(b) IEEE
(c) ISO
(d) ANSI

CamScanner

15.1. Full-duplex transmission allows transmission in both directions at once.
15.2. For a sinusoidal pattern, the height from the mean position is called the amplitude.
15.3. Networks allow users simultaneous access to shared programs and data.
15.4. The physical layout of a LAN is known as the topology.
15.5. LAN topologies can be a bus, star, ring, or hybrid.
15.6. Hybrid network is the type of LAN that combines star, ring and bus networks.
15.7. Before a personal computer in a LAN can send and receive messages, a network interface card must be inserted into an expansion slot in a computer.
15.8. LANs connect computers that are relatively close together, whereas WANs join multiple LANs that are spread over a large geographical area.
15.9. A wide area network is a communications network that covers a wide geographical area, such as a province or a country.
15.10. Computers on the Internet must use the same protocol.
15.11. Interactions among networked computers must use a protocol.
15.12. In the OSI model, the transmission of data in bits takes place at the physical layer.
15.13. A modem can be used for either modulation or demodulation.
15.14. The modem converts computer-to-terminal electrical analogue signals to digital signals so that the data can be transmitted over telephone lines.

CamScanner

APPENDIX A

Syllabus of Computer Science for Class XI

UNIT 1: OVERVIEW OF COMPUTER SYSTEM

Contents & Scope	Learning Outcomes/Skills
1.1 Introduction to Computer System	• Describe term computer system ○ Computer Hardware ○ Computer Software ○ Basic Operations of Computer ○ Computer Memory • Demonstrate the block diagram of computer system consisting hardware, software, basic operations and memory
1.2 Computer Hardware and their examples	• Classify the hardware (input, output, storage, processing and communication) devices as per their functionality. • Input Devices ○ Keyboard ○ Mouse ○ Joy Stick / Joy Pad ○ Scanning Devices ○ Data Tablets / Stylus ○ Voice Recognition Devices ○ Touch Sensitive Screens ○ Webcam ○ Bar Code Scanner ○ Microphones • Output Devices ○ Monitors (LED, LCD, Plasma, CRT) ○ Printers (All Types) ○ Plotters ○ Multimedia Projectors ○ Speakers / Headphones ○ SGD (Speech Generating Device) • Storage Devices and Media ○ Hard Disk ○ Flash Drive/USB/Pen Drive ○ CD / DVD / CD-R/ CD-RW ○ Blu-ray Drive ○ SD Card and Micro SD Card

CamScanner

	• Communication Devices ○ Network Interface Card ○ Modem ○ Data Terminals Equipments ○ Fax Machines ○ Switch/ Hub ○ Gateway ○ Routers
1.3 Computer Software	• Explain system software and its types ○ Operating Systems ○ Device Drivers ○ Language Translator ○ Antimalware (Antivirus, Trojan Horses) • Discuss Application software ○ General Purpose Software ○ Special Purpose Software • Describe Utility software ○ Compression Software ○ Data Recovery Software ○ Disk Manager
1.4 Computer Memory	• Explain Primary and Secondary Memory • Compare primary and secondary memory as per their functions.
1.5 Cutting Edge Technology	• Describe how computers are affecting the technologies • Explain the Artificial Intelligence (AI) • Discuss the AI application in different fields • Discuss Cloud Computing and its advantages

UNIT 2: COMPUTER MEMORY

Contents & Scope	Learning Outcomes/Skills
2.1 Memory Measurement Units	• Understand the memory measurement units ○ Bit ○ Nibble ○ Byte ○ Kilo Bytes KB ○ Mega Bytes MB ○ Giga Bytes GB ○ Tera Bytes TB
2.2 Primary/ Main Memory	• Differentiate between volatile and non-volatile memory • Discuss the following types of computer memory:

CamScanner

	○ RAM - Static RAM - Dynamic RAM ○ ROM - PROM - EPROM - EEPROM
2.3 Secondary Memory	• Explain secondary storage devices ○ Hard Drive • Describe other Peripheral Storage Devices ○ Flash Drive (USB)/ Data Traveler ○ External Hard drive ○ SD Card/ Micro SD Card ○ Combo drives

UNIT 3: INSIDE CENTRAL PROCESSING UNIT (SYSTEM UNIT)

Contents & Scope	Learning Outcomes/Skills
3.1 System Unit	• Describe the characteristics of different CPU casings • Discuss the main components of system unit ○ Power supply ○ Motherboard/Main Board ○ Hard drive ○ Combo drive ○ Ports ○ Cooling System
3.2 Motherboard/ Mainboard	• Recognize the various components on motherboard ○ Microprocessors ○ BIOS (ROM) ○ RAM Slot ○ CMOS Battery ○ Jumpers ○ Ports ○ Expansion Slots ○ System Bus • Outline the function of each component
3.3 Microprocessor	• Explain the role of internal architecture of microprocessor ○ Clock ○ Registers ○ Arithmetic and Logic Unit (ALU) ○ Control Unit (CU)

CamScanner

	○ Cache • Recognize the External Architecture of Microprocessor (Based on pins) ○ Address pins ○ Data pins ○ Control pins ○ Interrupt Pins ○ Power Pins • Understand the microprocessors specifications in terms of Technology, Generation, Clock speed, Bits, Bus width, Cache
3.3 Buses	• Define buses ○ Address Bus ○ Data Bus ○ Control Bus • Differentiate the types of buses on the basis of data flow direction (Uni-directional and Bi-directional).
3.4 Ports	• Discuss the types of ports ○ Serial Ports ○ Parallel Ports ○ USB ○ HDMI
3.5 Expansion Slots	• Explain the functions of different types of slots for expanding the computer system ○ PCI ○ EISA ○ VESA

UNIT 4: OPERATING SYSTEM

Contents & Scope	Learning Outcomes/Skills
4.1 Introduction to Operating System	• Define Operating System and its objectives • Discuss the most used operating system for computers (Windows, Mac, DOS and UNIX) • Explain the following features of Operating System ○ Batch Processing ○ Multi-Tasking ○ Time Sharing ○ Multi-Processing ○ Parallel Processing ○ Distributed ○ Embedded

CamScanner

4.2 Functions of Operating System	Describe the following main functions of operating system: ○ Booting ○ Process Management ○ Memory Management ○ File Management ○ I/O System Management ○ Secondary Storage Management ○ Network Management ○ Protection System ○ Command-Interpreter
4.3 Process Management	• Define a process • Describe the following states of processing ○ New ○ Running ○ Waiting/ blocked ○ Ready ○ Terminated • Differentiate between ○ Thread and process ○ Multi-threading & multi-tasking
4.4 Working with GUI OS	• Familiarize with environment of the Windows Operating System • Apply data management techniques (files/ folders) ○ Creation ○ Deletion ○ Copying ○ Renaming • Understand the concept drives and paths • Apply files and folders searching options • Understand the Properties and Attributes of files and folders • Demonstrate the management of the hardware software resources through control panel • Use DirectX (dxdiag) command to analyze the computer resources • Show and Change System Configurations by using msconfig command to show
4.5 Advance Operating Systems	• Describe OS other than Windows • Explain the Open-Source OS and their advantages • Discuss the Operating System for Mobile • Devices (Android and its versions)

CamScanner

UNIT 5: PROGRAMMING CONCEPT USING C++

Contents & Scope	Learning Outcomes/Skills
5.1 Overview of Programming Concepts	• Explain data types & Operators used in C++ • Discuss input output streams • Use cascading technique in input output streams • Build different programs by using operators
5.2 Review of Control Structure	• Use Selection / Decision Making Structure to develop programs ○ if Statement ○ if-else statement ○ elseif statement ○ switch- default statement ○ Nested if- else • Use Iteration Statement/Loops in programs ○ for ○ while ○ do-while ○ Nested loops • Differentiate Jump Statements ○ Break ○ continue ○ goto
5.3 Function	• Discuss type of functions and their advantages ○ Pre-defined ○ User Defined • Explain the signature of function (Name, Arguments, Return type) • Apply the following terms related to functions through programming ○ Function declaration (prototype) ○ Function definition (declarator & body) ○ Function call • Explain Scope & Visibility ○ local, global, and static variables • Differentiate/Compare between formal and actual parameters • Define inline functions in a program • Define the statement 'exit function'
5.4 Passing arguments and	• Apply passing arguments to user defined functions ○ Constants

CamScanner

returning values	○ By value ○ By reference • Use default argument • Use "return" statement to return values from function
5.5 Function overloading	• Explain function overloading and its advantages • Write a code for function overloading by using ○ Number of arguments ○ Data types of arguments ○ Return types

UNIT 6: ARRAYS, STRINGS AND STRUCTURES

Contents & Scope	Learning Outcomes/Skills
6.1 Introduction to arrays	• Discuss the concept of an array and its elements • Explain the following ○ Array Name ○ Array Size ○ Array Index • Describe defining and initializing an array of different data types (with syntax) • Explain how to access an element in an array • Develop a program to traverse an array (using loop) • Use the sizeof( ) function to find the size of an array
6.2 Two dimensional Arrays	• Explain two-dimensional array and its initialization with syntax • Write a program to access a two-dimensional array
6.3 Strings	• Explain strings and how to define a string • Understand the appropriate use of string terminator ('\0') • Write a statement to initialize a string • Apply the following string functions ○ strlen() ○ strcat() ○ strcpy() ○ strcmp() ○ substr()
6.4 Structures	• Explain Structure and its declaration • Explain how to define structure Variable • Write code for Initializing and accessing structure members.

CamScanner

UNIT 7: COMPUTER COMMUNICATION AND NETWORKS

Contents & Scope	Learning Outcomes/Skills
7.1 Basic of Networks	• Describe the Networks and its advantages • Discuss Network components: o Sender o Receiver o Medium o Message o Protocol
7.2 Data Flow Direction (Communication Modes)	• Define following Data flow directions (Communication Modes) o Simplex o Half Duplex o Full Duplex
7.3 Transmission Medium	• Describe Transmission medium and their types: o Guided Medium (Twisted Pair, Coaxial & Fiber Optic) o Unguided medium (Radio waves, Microwave, Infrared)
7.4 Network Types and Topologies:	• Discuss Types of networks: o Local Area Network (LAN) o Wide Area Network (WAN) Describe following topologies o Ring o Star o Bus o Mesh o Hybrid o Tree
7.5 Standard Organizations Network Architectures	• Explain Standard organizations o ISO o IEEE o ITU o ASCII • Describe following network architectures o ISOs OSI Model o TCP/IP Model o Internet Model

CamScanner

Activities for Grade XI

1. Demonstrate the physical component of the CPU and motherboard
2. Assembling, dissembling and connecting different part of computer like RAM, Hard Disk, Cables etc.
3. Demonstrate the BIOS setting and selecting booting options
4. Demonstrate the components of Desktop and Windows Explorer
5. Manage files and folder using cut, copy, move, delete and rename operations through drag and drop and menu selection
6. Demonstrate the management of the hardware and software resources through control panel
7. Use files and folders searching options
8. Demonstrate the Properties and Attributes of files and folders, also change the attributes
9. Write program of Simple Calculator using Arithmetic Operators
10. Write of Program of simple Marksheet by using Selection Structures.
11. Input a number and find out whether it is a prime or composite
12. Input a 4 (four) digit number and display each digit in separate line
13. Calculate factors of a given numbers
14. Write a function to draw of stars for a given size
15. Write a function that returns larger number from 2 arguments
16. Write a program to sort list of items in array (numeric / string)
17. Search a Number/Character from inputted list of items (numeric /string)
18. Calculate average of all array members
19. Finding addition and multiplication of a matrices (Maximum 3 x 3)
20. Write a program to display your inputted name in reverse order using strlen().
21. Writer a program to create structure to display employee data with structure members as name, designation and salary.
22. Demonstrate different types of network cables and practically implement the Cross-wired cable and straight through cable using clamping tool.
23. Show usage of network devices
24. Set of network IP setting.
25. Connect the computers in Local Area Network.
26. Make centralized and decentralized Network.

CamScanner