

**M.Sc. Previous Year
Chemistry, MC-05(B)**

COMPUTERS FOR CHEMISTS



मध्यप्रदेश भोज (मुक्त) विश्वविद्यालय – भोपाल
MADHYA PRADESH BHOJ (OPEN) UNIVERSITY - BHOPAL

Reviewer Committee

1. Dr. Ila Jain
Professor
Govt. Dr. Shyama Prasad Mukharjee Science and
Commerce College, Bhopal (M.P.)
2. Dr. Neetupriya Lachoria
Assistant Professor
Govt. Dr. Shyama Prasad Mukharjee Science and
Commerce College, Bhopal (M.P.)
3. Dr. S.D. Dwivedi
Professor
Govt. Dr. Shyama Prasad Mukharjee Science and
Commerce College, Bhopal (M.P.)

Advisory Committee

1. Dr. Jayant Sonwalkar
Hon'ble Vice Chancellor
Madhya Pradesh Bhoj (Open) University,
Bhopal, (M.P.)
2. Dr. L.S. Solanki
Registrar
Madhya Pradesh Bhoj (Open) University,
Bhopal, (M.P.)
3. Dr. Kishor John
Director
Madhya Pradesh Bhoj (Open) University,
Bhopal, (M.P.)
4. Dr. Ila Jain
Professor
Govt. Dr. Shyama Prasad Mukharjee Science
and Commerce College, Bhopal (M.P.)
5. Dr. Neetupriya Lachoria
Assistant Professor
Govt. Dr. Shyama Prasad Mukharjee Science
and Commerce College, Bhopal (M.P.)
6. Dr. S.D. Dwivedi
Professor
Govt. Dr. Shyama Prasad Mukharjee Science
and Commerce College, Bhopal (M.P.)

COURSE WRITERS

Sanjay Saxena, Renowned Author of over 130 books on Information Technology
Unit (1.0-1.1, 1.2, 1.3-1.4, 1.5-1.7, 1.8-1.9, 1.10, 1.11-1.15)

Dr. Himanshu Pandiya, Professor, H.O.D. and Dean, F.O.S., Department of Chemistry, Faculty of Science, People's
University, Bhopal (M.P.)
Units (2.0-2.4, 2.9-2.18, 3, 4)

Prof. S. Mohan Naidu, Former Principal (I.C.), V.R.N. College of Computer Science and Management, Tirupati
Unit (2.5-2.6)

Dr. Subburaj Ramasami, Former Professor and Consultant, S.R.M. University, Chennai
Unit (2.7-2.8)

Copyright © Reserved, Madhya Pradesh Bhoj (Open) University, Bhopal

All rights reserved. No part of this publication which is material protected by this copyright notice may be reproduced or transmitted or utilized or stored in any form or by any means now known or hereinafter invented, electronic, digital or mechanical, including photocopying, scanning, recording or by any information storage or retrieval system, without prior written permission from the Registrar, Madhya Pradesh Bhoj (Open) University, Bhopal

Information contained in this book has been published by VIKAS® Publishing House Pvt. Ltd. and has been obtained by its Authors from sources believed to be reliable and are correct to the best of their knowledge. However, the Madhya Pradesh Bhoj (Open) University, Bhopal, Publisher and its Authors shall in no event be liable for any errors, omissions or damages arising out of use of this information and specifically disclaim any implied warranties or merchantability or fitness for any particular use.

Published by Registrar, MP Bhoj (open) University, Bhopal in 2020



VIKAS® is the registered trademark of Vikas® Publishing House Pvt. Ltd.

VIKAS® PUBLISHING HOUSE PVT. LTD.

E-28, Sector-8, Noida - 201301 (UP)

Phone: 0120-4078900 • Fax: 0120-4078999

Regd. Office: A-27, 2nd Floor, Mohan Co-operative Industrial Estate, New Delhi 1100 44

• Website: www.vikaspublishing.com • Email: helpline@vikaspublishing.com

SYLLABI-BOOK MAPPING TABLE

Computers for Chemists

Syllabi	Mapping in Book
<p>Unit I Introduction to Computer and Computing Basic structure and functioning of computers with a PC as an illustrative example. Memory, I/O devices. Secondary storage. Computer languages, Operating systems with DOS as an example. Introduction to UNIX and WINDOWS, Data Processing, principles of programming, Algorithms and flow charts.</p>	<p>Unit-1: Introduction to Computer and Computing (Pages 3-90)</p>
<p>Unit II Computer Programming in Fortran/C/Basic (The language features are listed herewith reference to FORTRAN. The instructor may choose another language such as BASIC or C and the features may be replaced appropriately). Elements of the computer language, Constants and variables, Operations and symbols, Expressions, Arithmetic assignment statement, Input and Output, Format statement, Termination statements, Branching statements such as IF or GO TO statements, LOGICAL variables, Double precision variables, Subscripted variables and DIMENSION, DO statement, FUNCTION and SUBROUTINE, COMMON and DATA statements. (Students learn the programming logic and these language features by hands on experience on a personal computer from the very beginning of this topic).</p>	<p>Unit-2: Computer Programming in FORTRAN/C/Basic (Pages 91-160)</p>
<p>Unit III Programming in Chemistry Development of small computer codes involving simple formulae in chemistry, such as van der Waals equation, pH titration, kinetics, radioactive decay. Evaluation of lattice energy and ionic radii from experimental data, Linear simultaneous equations to solve secular equations within the Huckel theory, Elementary structural features such as bond lengths, bond angles, dihedral angles etc. of molecules extracted from a databases such as Cambridge data base.</p>	<p>Unit-3: Programming in Chemistry (Pages 161-188)</p>
<p>Unit IV Use of Computer Programmes The students will learn how to operate a PC and how to run standard programmes and packages. Execution of linear regression, X- Y plot, numerical integration and differentiation as well as differential equation solution programmes, Monte Carlo and Molecular dynamics, Programmes with data preferably from physical chemistry laboratory. Further, the students will operate one or two or the packages such as MATLAB, EASYPLOT, LOTUS, FOXPRO and Word Processing software such as WORDSTAR/MSWORD.</p>	<p>Unit-4: Use of Computer Programmes (Pages 189-213)</p>



CONTENTS

INTRODUCTION	1
UNIT 1 INTRODUCTION TO COMPUTER AND COMPUTING	3-90
1.0 Introduction	
1.1 Objectives	
1.2 Basic Structure and Functioning of Computers	
1.3 I/O Devices	
1.4 Memory	
1.5 Computer Languages	
1.6 Operating System with DOS	
1.7 Introduction to Windows	
1.8 Introduction to UNIX	
1.9 Data Processing	
1.10 Principles of Programming: Algorithms and Flow Charts	
1.11 Answers to ‘Check Your Progress’	
1.12 Summary	
1.13 Key Terms	
1.14 Self-Assessment Questions and Exercises	
1.15 Further Reading	
UNIT 2 COMPUTER PROGRAMMING IN FORTRAN/C/BASIC	91-160
2.0 Introduction	
2.1 Objectives	
2.2 Introduction to C Language	
2.2.1 Importance of C	
2.2.2 Execution of a C Program	
2.3 Elements of the Computer Language	
2.3.1 Character Set	
2.3.2 Identifiers and Keywords	
2.3.3 Data Types	
2.4 Constants and Variables	
2.5 Operators and Expressions	
2.5.1 Operators	
2.5.2 Expressions	
2.6 Input/Output Operations	
2.7 Branching	
2.7.1 if Statement	
2.7.2 if...else Statement	
2.7.3 Nesting of the if...else Statements	
2.7.4 Logical Operators and Branching	
2.7.5 Conditional Operator and if...else	
2.8 Loops and Control Constructs	
2.8.1 Iteration using if	
2.8.2 for Statement	
2.8.3 Symbolic Constants and Looping	
2.8.4 Other Forms of the for Loop	
2.8.5 The while Loop	
2.8.6 do . . . while Loop	
2.8.7 switch Statement	
2.8.8 Break, Continue, Return and goto	

- 2.9 Logical Variables
- 2.10 Double Precision Variable
- 2.11 Subscripted and Dimension Variable
- 2.12 Function and Subroutine
- 2.13 Common and Data Commands
- 2.14 Answers to 'Check Your Progress'
- 2.15 Summary
- 2.16 Key Terms
- 2.17 Self-Assessment Questions and Exercises
- 2.18 Further Reading

UNIT 3 PROGRAMMING IN CHEMISTRY

161-188

- 3.0 Introduction
- 3.1 Objectives
- 3.2 Chemistry and Basic Programming
 - 3.2.1 Elements of Programming
- 3.3 Determination of Molecular Weight
- 3.4 Debye-Huckel Limiting Law Theory
- 3.5 Ionic Mobilities of Ions
- 3.6 Thermodynamic Parameters
- 3.7 Van der Waal's Equation
- 3.8 Radioactive Decay
- 3.9 Application of Chemical Kinetics
- 3.10 Determination of Lattice Energy
- 3.11 pH Titration
- 3.12 Program to Obtain the Value of Atomic Mass Unit in MeV
- 3.13 Determination of RMS, Average and Most Probable Velocities of Gases
- 3.14 Answers to 'Check Your Progress'
- 3.15 Summary
- 3.16 Key Terms
- 3.17 Self-Assessment Questions and Exercises
- 3.18 Further Reading

UNIT 4 USE OF COMPUTER PROGRAMMES

189-213

- 4.0 Introduction
- 4.1 Objectives
- 4.2 Standard Programmes and Packages
 - 4.2.1 Programming Process
- 4.3 Execution of Linear Regression
- 4.4 Numerical Integration and Differentiation
- 4.5 Monte Carlo and Molecular Dynamics
- 4.6 Introduction to Software Packages
 - 4.6.1 MATLAB
 - 4.6.2 EasyPlot
 - 4.6.3 FoxPro
- 4.7 Answers to 'Check Your Progress'
- 4.8 Summary
- 4.9 Key Terms
- 4.10 Self-Assessment Questions and Exercises
- 4.11 Further Reading

INTRODUCTION

Computers have today changed the face of the world of business and management. Today, it is essential for a successful manager to be comfortable using computers and also be aware of the great potential they offer in improving efficiency and productivity in all aspects of business. Students need to understand the functional components of a computer. They should also have thorough knowledge of the characteristics, performance and interactions of these components. Without knowledge of computer architecture, it will not be possible for students to structure a program and ensure that it runs efficiently.

C is a programming language and is substantially different from C++ and C#. Many operating systems are written using C, UNIX being the first. Later, Microsoft Windows, Mac OS X and GNU/Linux were written in C. Not only is C the language of operating systems, it is the precursor and inspiration for almost all the popular high-level languages available today. Perl, PHP, Python and Ruby are also written in C. In fact, one of the strengths of C is its universality and portability across various computer architectures. Therefore, C can be used for the development of different types of applications that include real-time systems and expert systems. C also provides flexibility to users for introducing new types of features in their programs, depending upon the requirement and definition of user-defined functions. The various features of C—algorithms, flow charts, decision-making statements, functions, arrays, structures and pointers—are useful for program developers.

This book, *Computers for Chemists*, follows the SIM format wherein each Unit begins with an Introduction to the topic followed by an outline of the 'Objectives'. The detailed content is then presented in a simple and an organized manner, interspersed with 'Check Your Progress' questions to test the understanding of the students. A 'Summary' along with a list of 'Key Terms' and a set of 'Self-Assessment Questions and Exercises' is also provided at the end of each unit for effective recapitulation.

NOTES



UNIT 1 INTRODUCTION TO COMPUTER AND COMPUTING

NOTES

Structure

- 1.0 Introduction
- 1.1 Objectives
- 1.2 Basic Structure and Functioning of Computers
- 1.3 I/O Devices
- 1.4 Memory
- 1.5 Computer Languages
- 1.6 Operating System with DOS
- 1.7 Introduction to Windows
- 1.8 Introduction to UNIX
- 1.9 Data Processing
- 1.10 Principles of Programming: Algorithms and Flow Charts
- 1.11 Answers to 'Check Your Progress'
- 1.12 Summary
- 1.13 Key Terms
- 1.14 Self-Assessment Questions and Exercises
- 1.15 Further Reading

1.0 INTRODUCTION

Computers can store huge amounts of data and are designed to cater to the end user's need for speed, accuracy, diligence, versatility and storage capacity. A computer can perform no function unless it is able to communicate with the outside world. Therefore, a system for receiving information from and communicating results to the external world of other computers and users is necessary for computers. An output device is an electromechanical device that accepts data from the computer and translates it into a form that can be understood by the outside world. The processed data, stored in the memory of the computer, is sent to an output unit, which then transforms the internal representation of data into a form that can be read by the users.

Computer language is the computer's native language that can only be understood by the computer itself. FORTRAN, COBOL, BASIC, PASCAL, ADA, LISP, C, C#, C++ and Java are the frequently used computer languages. All computer languages can be classified as machine language or first generation language, assembly language or second generation language or high-level language or third generation language. An assembly language is a low-level programming language which implements a symbolic representation of the machine codes and other constants needed to program a given CPU architecture.

NOTES

Interpreters and compilers are translation or conversion programs that produce the machine code from high-level languages. An interpreter translates the instructions of the program one statement at a time. A compiler takes an entire high-level language program to produce a machine code version for running as a single program.

Microsoft Disk Operating System (MS DOS) is a single user, single tasking operating system. DOS has a command-line, text-based/non-graphical user-interface commonly referred to as Character Based User Interface (CUI). When the computer is switched on, a small program checks all internal devices, electronic memory and peripherals. Once this process is completed, MS DOS is loaded. UNIX is a multi-user and multitasking operating system. In a multi-user environment, the computer can receive the commands from a number of end users to run programs, access files, and print documents simultaneously. In this unit, you will learn about the basic structure and functioning of computers, I/O devices, memory, computer languages, operating system with DOS, Windows and UNIX and principles of programming.

1.1 OBJECTIVES

After going through this unit, you will be able to:

- Understand the basic structure and functioning of computers
- Explain the different types of I/O devices
- Explain the concept memory
- Discuss the various categories computer languages
- Explain the basics of DOS, Windows and UNIX
- Understand the term data processing
- Describe the principles of programming

1.2 BASIC STRUCTURE AND FUNCTIONING OF COMPUTERS

You know that computers can store huge amounts of data and are designed to cater to the end user's need for speed, accuracy, diligence, versatility and storage capacity. Their characteristics are as follows:

- **Speed:** The internal processes of computers operate at the speed of light. This speed is checked only due to the programs controlling these processes and the amount of data being processed. A computer can perform in a minute what a human being may require a lifetime to perform. The speed of computers is not referred to in terms of seconds or milliseconds. It is referred to in terms of microseconds (10^{-6}), nanoseconds (10^{-9}) and picoseconds (10^{-12}).

NOTES

- **Accuracy:** A computer is extremely accurate. Although there are chances of errors, they occur mostly due to human error and not due to technological drawbacks. Errors originate due to imprecise thinking by the programmer or due to the input of erroneous data. They could also arise due to the poor design of systems. Garbage In Garbage Out (GIGO) is the term used to refer to computer errors resulting from incorrect data input or due to lack of reliability of programs.
- **Diligence:** Unlike human beings, computers are capable of working for long hours without breaks. A computer can perform a million calculations with accuracy and speed. The speed or level of accuracy will be consistent and will not deteriorate till the last calculation.
- **Versatility:** Computers can perform any task, as long as it can be broken down to a series of logical steps. For example, a task such as preparing a payroll can be reduced to a few logical tasks or operations performed in a logical sequence. This breaking down of a process into steps facilitates computerized processing.

A computer does have its limitations also. It can perform only four basic operations:

- (i) It can exchange information with the outside world via input/output (I/O) devices.
 - (ii) It can transfer data internally within the CPU.
 - (iii) It can perform basic arithmetic operations.
 - (iv) It can perform comparisons.
- **No intelligence:** A computer does not possess any intelligence of its own. It needs to be told what it has to do and in what sequence.
 - **Information explosion:** The speed with which computers can process information in huge volumes, has resulted in information explosion or generation of information on a large scale. Human beings have the ability to sift through data or knowledge and choose to retain only the important information and forget the irrelevant or unimportant stuff. There is clearly a difference in the way computers store information and the way human beings do. The secondary storage capacity of computers assists in storing and recalling any amount of information. Therefore, it becomes possible to retain information for as long as desired and recall it whenever needed.

Basic Anatomy of a Computer

The size, shape, cost and performance of computers have changed over the years. However, the basic logical structure remains the same (Figure 1.1). A computer system has three essential parts:

- Input device
- CPU (consisting of the main memory, the arithmetic logic unit and the control unit).
- Output device

In addition to these basic parts, computers also use secondary storage devices (also called auxiliary storage or backing storage), used for storing data and instructions on a long-term basis.

NOTES

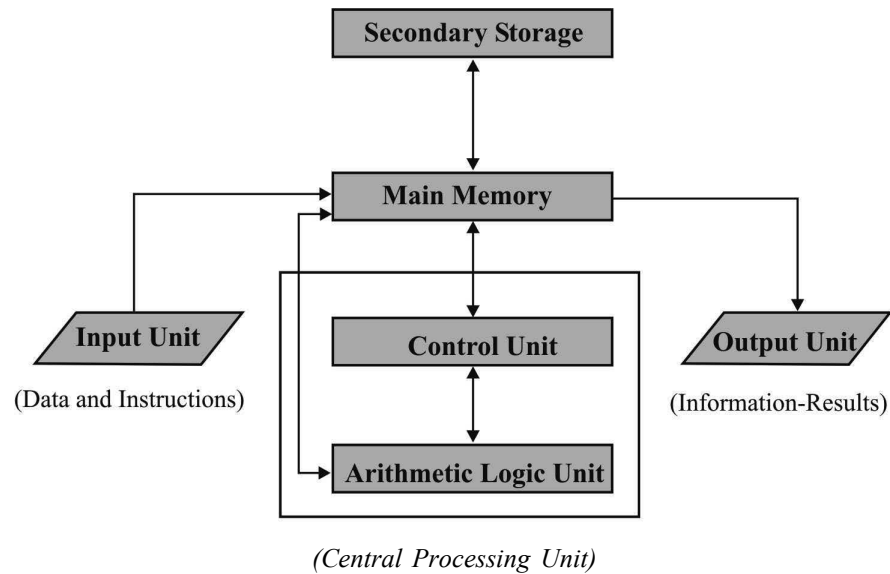


Fig. 1.1 Schematic Representation of a Computer System

Inputting, storing, processing, outputting and controlling are the basic operations that help convert raw data into relevant information.

1. Input units

Inputting is the process of entering data and instructions into the computer system. Both program and data need to be in the computer system before any kind of operation can be performed. Program refers to the set of instructions which the computer has to carry out, and data is the information on which these instructions are to operate. For example, if the task is to rearrange a list of telephone subscribers in alphabetical order, the sequence of instructions that guide the computer through this operation is the program, whilst the list of names to be sorted is the data.

The input unit performs the process of transferring data and instructions from the external environment into the computer system. Instructions and data enter the input unit depending upon the particular input device used (keyboard, scanner, card reader, etc). Regardless of the form in which the input unit receives data, it converts the data and instructions into a form that is acceptable by the computer (binary codes). It then supplies the converted data and instructions for further processing to the computer system.

Main memory (primary storage)

Storing is the process of saving instructions and data so as to make them available for future use, as and when required.

Instructions and data are stored in the primary storage before processing and are transferred when there is need, to the arithmetic logic unit (ALU) where the actual processing takes place. After completing the process, the final results

are again stored in the primary storage till they are released to an output device. Also, any intermediate results generated by the ALU are temporarily transferred back to the primary storage till there is a need for them. Thus, data and instructions may move back and forth many times between the primary storage and the ALU before the processing is completed. It may be worth remembering that no processing is done in the primary storage.

Arithmetic logic unit

Processing refers to the performing of arithmetic or logical operations on data, to convert them into useful information. Arithmetic operations include operations of add, subtract, multiply, divide and logical operations are operations of comparison like less than, equal to, greater than.

After the input unit transfers the information into the memory unit, the information can then be further transferred to the ALU where comparisons or calculations are done and results are sent back to the memory unit. Since all data and instructions are represented in numeric form (bit patterns), ALUs are designed to perform the following four basic arithmetic operations: multiply, divide, add, subtract, and logical operations like less than, equal to, greater than.

Secondary storage

A computer has limited storage capacity. It becomes necessary to store large volumes of data. Therefore, additional memory called secondary storage or auxiliary memory is used in most computer systems.

Storage other than the primary storage is called secondary storage and it enables permanent storage of programs and huge volumes of data belonging to the users. Examples of such storage are hardware devices like magnetic tapes and magnetic disks.

2. Output unit

Outputting is the process of providing the results to the user. These could be in the form of visual display and /or printed reports.

Since computers work with binary codes, the results produced are also in binary form. The basic function of the output unit, therefore, is to convert these results into human-readable form before providing the output through various output devices like terminals, printers, etc.

Control unit

Controlling refers to the process of directing the sequence and manner of performance of all these operations. It is the function of the control unit to ensure that according to the stored instructions, the right operation is done on the right data at the right time. It is the control unit that obtains instructions from the program stored in the main memory, interprets them and ensures that other units of the system execute them in the desired order. In effect, the control unit is comparable to the central nervous system in the human body.

NOTES

3. Central processing unit

The control unit and arithmetic logic unit are together known as the central processing unit. It is the brain of any computer system.

NOTES

A CPU is the most important component of a digital computer that interprets the instructions and processes the data contained in computer programs. The CPU works as the brain of the computer and performs most of the calculations. It is also referred to as the processor and is the most important component of a computer. For large computers, a CPU may require one or more printed circuit boards (PCBs) but in the case of PCs it comes in the form of a single chip called a microprocessor. PCB is a board that contains the circuitry used to connect the components of a PC.

Basic Design and Components of a Computer

Personal computers are microcomputers that are commonly used for commercial data processing, Desktop Publishing (DTP), engineering applications and so on. Figure 1.2 shows a personal computer.

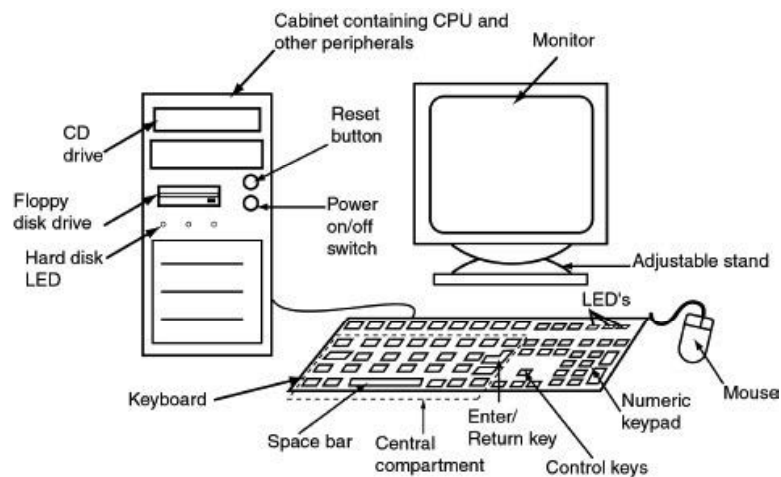


Fig. 1.2 Personal Computer

A personal computer comprises a Hard Disk Drive (HDD), Random Access Memory (RAM), processor, a keyboard, a Floppy Disk Drive (FDD), a mouse, a CD drive, a colour monitor and Read Only Memory (ROM). The RAM, ROM, microprocessor and other circuits are connected on the motherboard, which is a single board as shown in Figure 1.3.

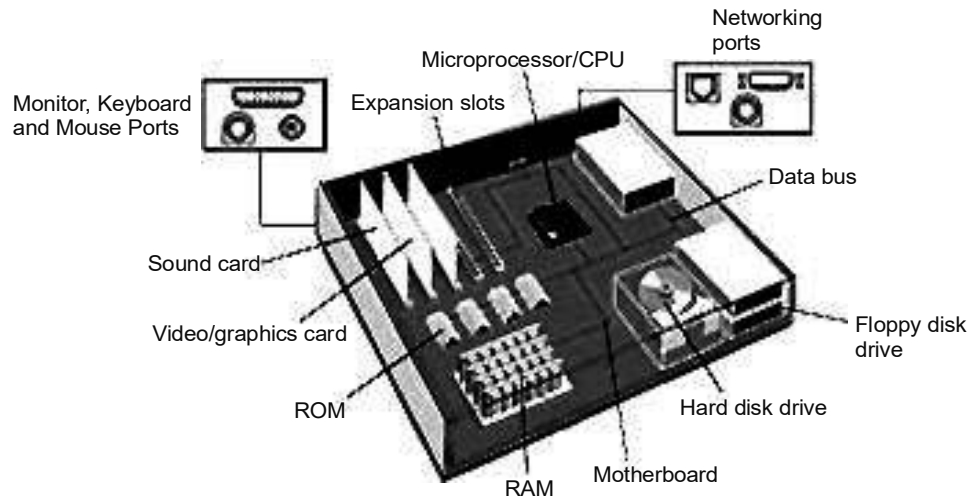


Fig. 1.3 Motherboard and CPU

Processor

The microprocessor control the control unit, memory unit (register) and arithmetic logic unit (Figure 1.4). The processing speed of a computer depends on the clockspeed of the system and is measured in megahertz (MHz).

The Intel Corporation's Pentium processors are used in most personal computers. Motorola, Cyrix and AMD (Advanced Micro Devices) are other makers of processors which are also used in personal computers.



Fig. 1.4 A Microprocessor

1.3 I/O DEVICES

A computer can perform no function unless it is able to communicate with the outside world. Therefore, a system for receiving information from and communicating results to the external world of other computers and users is necessary for computers. Computers have an input-output sub-system, referred to as I/O sub-system, which provides an efficient mode of communication between the central system and the outside world. Programs and data must be entered into the computer memory for processing, and results obtained from computations must be displayed or recorded for the user's benefit.

NOTES

NOTES

This can be explained with a very common scenario where the average marks of a student need to be calculated, based on the marks obtained in various subjects by the student. The marks would typically be available in the form of a document containing the student's name, roll number and marks scored in each subject. This data must first be stored in the computer's memory after converting it into machine – readable form. The data will then be processed (average marks calculated) and sent from the memory to the output unit, which will present the data in a form that can be read by users. (Figure 1.5)

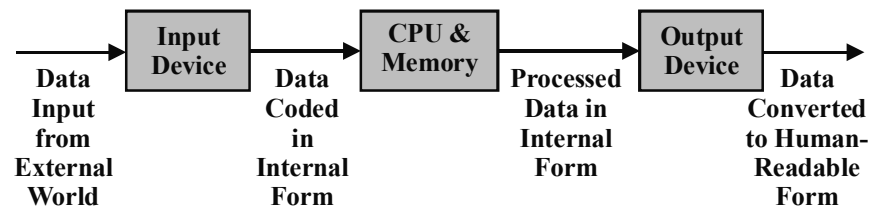


Fig. 1.5 Role of I/O Devices in a Computer System

The I/O devices that provide a means of communication between the computer and the outside world are known as peripheral devices. This is because they surround the CPU and the memory of a computer system. While input devices are used to enter data from the outside world into the primary storage, output devices are used to provide the processed results from primary storage to users.

As mentioned earlier in this unit, input devices transfer instructions and user data to the computer. The most commonly used input devices can be classified into the categories of:

- Keyboard devices (general and special purpose, key-to-diskette, key-to-disk, key-to-tape)
- Point-and-draw devices (mouse, trackball, joystick, light pen, touch screen)
- Scanning devices (optical mark recognition, magnetic ink character recognition, optical bar code reader, digitizer, electronic-card reader)
- Voice recognition devices
- Vision-input devices (webcam, video camera etc.)

Keyboard

Keyboard devices allow input into the computer system by pressing a set of keys, mounted on a board connected to the computer system. Keyboard devices are typically classified as general-purpose keyboards and special-purpose keyboards.

General-purpose keyboard

The most familiar means of entering information into a computer is through a typewriter-like keyboard that allows a person to enter alphanumeric information directly.

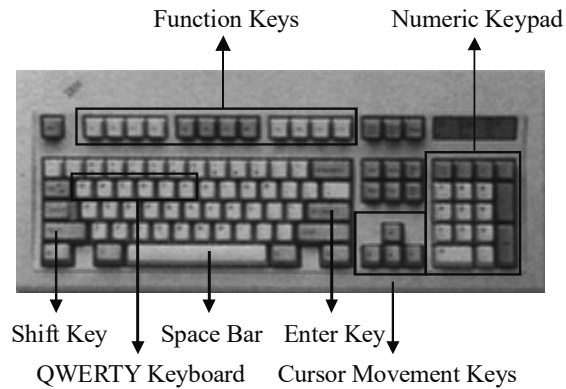


Fig. 1.6 QWERTY Keyboard Layout

The most popular keyboard used today is the 101-keys keyboard with a traditional QWERTY layout, having an alphanumeric keypad, twelve function keys, a variety of special-function keys, numeric keypad, and dedicated cursor-control keys. It is so called because of the arrangement of its alphanumeric keys in the upper left row (as refer the Fig. 1.6).

- **Alphanumeric Keypad:** Contains keys for the English alphabets, 0 to 9 numbers, special characters like * + - / [] etc.
- **Twelve Function Keys:** These are keys labeled F1, F2 ... F12 and are a set of user programmable function keys. The actual function assigned to a function key differs from one software package to another. These keys are also called soft keys since their functionality can be defined by the software.
- **Special-function Keys:** Have special functions assigned to each of them. For example, the enter key is used to send the keyed-in data into the memory. Other special keys include:
 - *Shift* (used to enter capital letters or special characters defined above the number keys)
 - *Spacebar* (used to enter a space at the cursor location)
 - *Ctrl* (used in conjunction with other keys to provide added functionality on the keyboard)
 - *Alt* (like Ctrl, it is used to expand the functionality of the keyboard)
 - *Tab* (used to move the cursor to the next tab position defined)
 - *Backspace* (used to move the cursor one position to the left and also delete the character in that position)
 - *Caps Lock* (to toggle between the capital letter lock feature – when ‘on’, it locks the keypad for capital letters input)
 - *Num Lock* (to toggle the number lock feature on and off – when ‘on’, it inputs numbers when you press the numbers on the numeric keypad)
 - *Insert* (used to toggle between the insert and overwrite mode during data entry – when ‘on’, entered text is inserted at the cursor location)

NOTES

NOTES

- *Delete* (used to delete the character at the cursor location)
- *Home* (used to move the cursor to the beginning of the work area which could be the line, screen or document depending on the software being used)
- *End* (used to move the cursor to the end of the work area)
- *Page Up* (used to display the previous page of the document being currently viewed on screen)
- *Page Down* (used to view the next page of the document being currently viewed on screen)
- *Escape* (usually used to negate the current command)
- *Print Screen* (used to print what is being currently displayed on the screen)
- **Numeric Keypad:** Consists of keys having numbers (0 to 9) and mathematical operators (+, -, *, /) defined on them. It is usually located on the right side of the keyboard and supports quick entry of numerical data.
- **Cursor-control Keys:** Defined by the arrow keys used to move the cursor in the direction indicated by the arrow (top, down, left, right)

The Dvorak system, designed for easier use and learning, is yet another popular key arrangement. In this keyboard, the vowels are located on one side and the most commonly used consonants are located on the other side of the home or middle row. This ensures that when a user types words, the keystrokes tend to alternate between the two hands. Though the Dvorak system has not gained wide popularity, nevertheless, some users do favour it over the QWERTY keyboard.

Special-purpose keyboard

These are standalone data entry systems used for computers deployed for specific applications. These typically have special purpose keyboards to enable faster data entry. A very typical example of such kind of keyboards can be seen at the Automatic Teller Machines or the ATMs where the keyboard is required for limited functionality (support for some financial transactions) by the customers. Point-of-Sale or POS terminals at fast food joints and air/railway reservation counters constitute other examples of special-purpose keyboards.

These keyboards are specifically designed for special types of applications only.

Mouse

A mouse is a small device that is pushed across a desk surface by a user. It serves the function of indicating a point on the display screen and selects one or more possible actions from that particular location. It gained popularity as a computer device when it was made a standard tool of the Apple Macintosh by Apple Computer. It now forms an integral component of the graphical user interface

(GUI) of all personal computers. It derives its name from being similar in colour and size to a toy mouse.

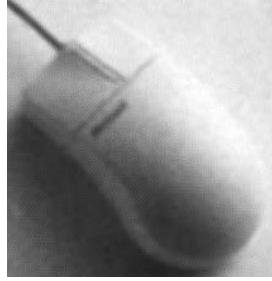


Fig. 1.7 Mouse

Traditional mice have two buttons on top, with the left one being used most often. In the windows operating system, clicking once on it sends a ‘Select’ signal that informs users that a specific position has been selected, and that it is ready for further action. Double-clicking (clicking the mouse twice in quick succession) a selected position initiates a specific action on the object that has been selected. For instance, in a Windows operating system, double-clicking an object launches the program associated with it. The button on the right offers certain infrequently used functions. For instance, while a Web page is being viewed, clicking on an image opens a pop-up menu that offers the option of saving the image on the hard disk. Some mice even come with a third button. Models for left-handed users are also available.

Variants of a mouse

Touchpad

A touchpad is a touch sensitive input device which takes user input to control the onscreen pointer and perform other functions similar to that of a mouse. Instead of having an external peripheral device such as a mouse, the touchpad enables the user to interact with the device through the use of a single or multiple fingers being dragged across relative positions on a sensitive pad. (Figure 1.8) They are mostly found in Notebooks and laptops where convenience, portability and space are the prime design concerns.

Touchpads are pressure and touch sensitive. They use finger drag movement and tapping combinations to perform multiple control operations. For instance, a user can also perform the scroll function of a mouse by sliding a finger in-between certain points, usually the extreme top and bottom corners on the right side of the pad. Their user interface is much more convenient to use and puts less strain on the wrist and hand, which is a very common side effect of using computers with a mouse over a long period of time. The functionality of a touchpad isn’t limited by the manufacturers’ user interface. They can be user programmed to recognize a combination of finger and tap movements to perform new actions as an input device.

Touchpads have shown a steady growth in market demand and user acceptability over all phases of their development cycles. Their growth is further

NOTES

expected to continue in the same way as the demand for touch sensitive portable devices with more functionality and better appearance is on the rise.

NOTES



Fig. 1.8(a) Touchpad



Fig. 1.8(b) Touchpad

Hand-Held Devices

Trackball

The trackball is a pointing device that is much like an inverted mouse. It consists of a ball inset in a small external box, or adjacent to—and in the same unit as—the keyboard of some portable computers. (Figure 1.9)



Fig. 1.9 Trackball

It is more convenient and requires much less space than the mouse since here the whole device is not moved (as in the case of a mouse). Trackball comes in various shapes but supports the same functionality. Typical shapes used are a ball, a square, and a button (typically seen in laptops).

Joystick

The joystick is a vertical stick that moves the graphic cursor in the direction the stick is moved. (As shown in Fig. 1.10)



Fig. 1.10 Joystick

It consists of a spherical ball, which moves within a socket, and has a stick mounted on it. The user moves the ball with the help of the stick that can be moved left or right, forward or backward, to move and position the cursor in the desired location. Joysticks typically have a button on top that is used to select the option pointed by the cursor.

Video games, training simulators and control panels of robots are some common uses of a joystick.

Light pen

A light pen is a pen-shaped tool that allows natural on-screen movement. (Fig. 1.11) It is made up of a light sensitive cell and a lens assembly designed in such a way that it focuses onto itself any light in its field of view. It has a light receptor or scanning beam that locates the position of the pen (X and Y coordinates on the screen). It gets activated when the pen is pressed against the display screen. Once the light pen locates a particular section on the display screen, the required action is initiated by the appropriate system software.

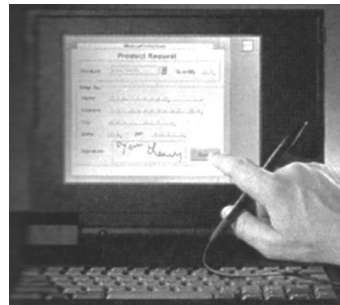


Fig. 1.11 Light Pen

Light pens are typically used in CAD (Computer Aided Design) applications to directly draw on screen.

Touch Screen

A touch screen is probably one of the simplest and most intuitive of all input devices. It uses optical sensors in, or near, the computer screen that can detect the touch of a finger on the screen. Once the user touches a particular screen position, sensors communicate the position to the computer. This is then interpreted by the computer to understand the user's choice for input.

The most common usage of touch screens is in information kiosks where users can receive information at the touch of a screen. These devices are becoming increasingly popular today.

Optical Input Devices

Scanning Devices

Scanning devices are input devices used for direct data entry from the source document into the computer system. Scanners capture and store information in a graphical format, and then display it on the graphical screen. A scanner consists of two parts, one that illuminates the page for capturing the optical image, and the

NOTES

other that converts and stores the graphical image in a digital format. Once this is done, the computer can then see and directly process this scanned graphical image.

NOTES



Fig. 1.12 Hand-held Scanner

There are two types of scanners, Contact and Laser. In both types, a beam of light is bounced off an image, and the value of the image is determined by measuring the reflected light. Hand-held contact scanners make contact as they are brushed over the printed matter to be read (Figure 1.12). Laser-based scanners are more versatile and can read data passed near the scanning area. (Figure 1.13).



Fig. 1.13 Flat-bed Scanner

Hand-held scanners are used where the information to be scanned or the volume of documents to be scanned is very low. They are much cheaper as compared to the flat-bed scanners.

Capturing information using scanners reduces the possibility of human error typically seen during large data entry. The reduction in human intervention improves the accuracy of data and provides for timeliness of the information processed.

The latest trend in inputting data is source data automation. Data is captured by the source data automation equipment as the by-product of a business activity. This wholly eliminates manual data input. A few common examples of these are described below.

Optical Mark Recognition (OMR)

Optical mark recognition devices sense the marks on computer readable paper. They are usually used by academic institutions for grading aptitude tests where the correct option is marked from multiple options given on special kinds of answer sheets (shown in Fig. 1.14). The OMR device directly reads this paper, and the computer can then use it for further processing.

The actual technique used by an OMR device once again involves focusing light on the page being scanned, thereby detecting the reflected light pattern for the marks. Pencil marks made by the user reflect the light determining which responses are marked.

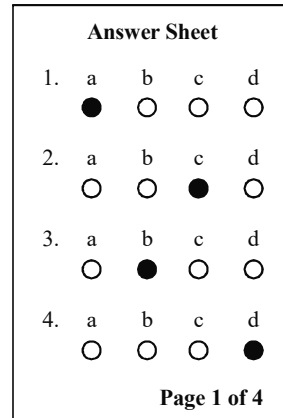


Fig. 1.14 An Example of a Pre-printed Answer Sheet that can be Read by an OMR Device

NOTES

Magnetic Ink Character Recognition (MICR)

MICR is similar to optical mark recognition and is used exclusively by the banking industry. The banking industry makes use of MICR devices for directly reading the account numbers written on cheques and for processing them further as required.

Banks using MICR technology print cheque books on special types of paper. (As shown in Figure 1.15) The necessary details of the bank (like the bank's identification code, relevant account number, and cheque number) are pre-printed on the cheques using an ink that contains iron oxide particles that can be magnetised.

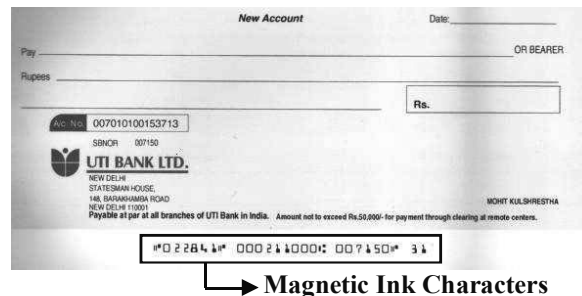


Fig. 1.15 A Bank Cheque using MICR Technology

Optical Barcode Reader (OBR)

Data coded in the form of small vertical lines forms the basis of barcoding. Alphanumeric data is represented using adjacent vertical lines called barcodes. These are of varying widths and the spacing between them is used to uniquely identify books, merchandise in stores, postal packages etc.

Below is an example of a bar code used on one of the books for its unique identification.

NOTES



Fig. 1.16(a) Barcode



Fig. 1.16(b) Barcode Reader

A barcode reader uses laser beam technology. The laser beam is moved across the pattern of bars in a barcode. These bars reflect the beam in different ways. The reflected beam is then sensed by a light-sensitive detector, which then converts the light patterns into electrical pulses, thereby transmitting them to logic circuits for further conversion to alphanumeric value.

Barcode devices are available as hand-held devices.

Digitizer

Digitizers are used to convert drawings or pictures and maps into a digital format for storage into the computer. (Figure 1.17)



Fig. 1.17 A Digitizer

A digitizer consists of a digitising or graphics tablet, which is a pressure sensitive tablet, and a pen with the same X and Y co-ordinates as on screen. Some digitising tablets also use a crosshair device instead of a pen. The movement of the pen or crosshair is reproduced simultaneously on the display screen. When the pen is moved on the tablet, the cursor on the computer's screen moves simultaneously to the corresponding position on screen (X and Y coordinates). This allows the user to draw sketches directly or input existing sketched drawings easily.

Digitizers see most common usage by architects and engineers as a tool for Computer Aided Designing (CAD).

Electronic-card reader

Card readers are devices that allow direct data input into a computer system. The electronic-card reader is connected to a computer system and reads the data encoded on an electronic card and transfers it to the computer system for further processing.

Electronic cards are plastic cards with data encoded on them and meant for a specific application. Typical examples of electronic cards are the plastic cards issued by banks to their customers for use in Automatic Teller Machines or ATMs. (Figure 1.18) Electronic cards are also used by many organizations for controlling access of various types of employees to physically secured areas.



Fig. 1.18 Access Card Security System

Depending on the manner in which the data is encoded, electronic cards may be either magnetic strip cards or smart cards. Magnetic strip cards have a magnetic strip on the back of the card. Data stored on magnetic strips cannot be read with the naked eye, a useful way to maintain confidential data.

Smart cards, going a stage further, have a built-in microprocessor chip where data can be permanently stored. They also possess some processing capability, making them suitable for a variety of applications. For example, to gain access, an employee inserts a card or badge in the reader. This device reads and checks the authorization code before permitting the individual to enter a secured area. Since smart cards can hold more information as compared to magnetic strip cards, they are gaining in popularity.

Output Devices

An output device is an electromechanical device that accepts data from the computer and translates it into a form that can be understood by the outside world. The processed data, stored in the memory of the computer, is sent to an output unit, which then transforms the internal representation of data into a form that can be read by the users.

There are two ways in which output can normally be produced – on paper, or on a display device/unit. Some applications also use other types of output such as mechanical output and speech output. Output produced on display units, or speech output that cannot be touched, is referred to as softcopy output while output produced on paper or material that can be touched is known as hardcopy output.

A wide range of output devices are available today and can be broadly classified under the categories of:

- Display devices (monitors, multimedia projectors, terminals—dumb, smart, intelligent, X terminals)

NOTES

NOTES

- Printers (dot matrix, inkjet, laser)
- Plotters (flatbed, drum)
- Computer Output Microfilms (COM)
- Voice response systems (voice reproduction system, speech synthesizer)

Monitors

Display devices

The display device is an important peripheral component of a computer system. Display terminals called alphanumeric terminals were used by conventional computers. These used a form of multi-dot (7×5 or 9×7) array to display characters. Their chief use lay in reading the text information displayed on the screen. The increasing demand for displaying graphs and pictures for presenting information in a visual form (more effectual in user interaction) led to the popularity of graphic display devices.

An image is produced by patterns created through a sequence of dots known as 'pixels' (picture elements) that comprise a graphic display. Every dot that forms a part of this pattern can be addressed uniquely and directly. Owing to the fact that each dot can be addressed as a separate unit, it provides greater flexibility for drawing pictures.

Display screen technology may belong to one of the three following categories:

Cathode Ray Tube (CRT): A display screen coated with phosphor, an electron gun and an electromagnetic field controlling the electron beam form the major constituents of a cathode ray terminal. The screen's phosphor-coating is organized into a grid of dots called pixels.

An electron beam is emitted by the electron gun, which the electromagnetic field directs towards the display screen coated with phosphor; and this is how an image is created.

CRT displays can be either of the two following types:

- **Raster scan display:** In a raster scan display, the electron beam is directed across all rows of pixels from the top to the bottom of the screen to project the image. Because the image is constantly refreshed, a high dynamic capability is provided. The raster screen display is gaining popularity now as it makes available full colour displays at reasonably low costs.
- **Vector CRT display:** In vector CRT displays, only the positions where the image is to be created have the electron beam directed at them.

The display device's resolution indicates the quality of the display. The number of horizontal and vertical pixels determine the resolution. Typical resolutions in graphic display range from (800×600) to (640×768) to (1024×1024) pixels.

Based on the resolution and the number of colours supported, several standards for colour monitors have evolved. The most popular of these include:

- Colour Graphics Adapter (CGA), which has a resolution of (320 × 200) and supports up to sixteen colours.
- Extended Graphics Adapter (EGA), which has a resolution of (640 × 350) and supports up to sixteen colours.
- Video Graphics Adapter (VGA), which has a resolution of (640 × 480) and supports up to 256 colours.
- Super VGA, having a resolution ranging from (800 × 600) to (1280 × 1024) and supporting up to 256 or more colours.

Note that each one of these is implemented by installing an add-on card in the computer, commonly known as graphics adapter or the video card. This card is then connected to the appropriate monitor.

Liquid Crystal Display (LCD): Though LCD technology is now being used for display terminals, it was first introduced in the 1970s in clocks and watches. In LCD screens, liquid crystals substitute cathode ray tubes to produce images. They do not have a superior colour capability and the quality of the images they produce is comparatively poor. The main advantage of LCD is its low energy consumption. It finds its most common usage in portable devices where low energy requirements and compactness are of prime importance.

Projection display: Projection display technology is characterized by replacing the personal size screen with large screens upon which the images are projected. Projection display systems are connected to the computer and an enlarged version of the object appearing on the computer display terminal is projected on a large screen. These are being used today for large group presentations.

Monitors

Monitors use a Cathode Ray Tube (CRT) to display information. It resembles a television screen and is similar to it in other respects.

The monitor is typically associated with a keyboard for manual input of characters. The screen displays information as it is keyed in, enabling a visual check of input before it is transferred to the computer. It is also used to display the output from the computer and hence serves as both an input and an output device.

The monitor along with the keyboard is called a Visual Display Unit (VDU).

This is the most commonly used input/output device today and is also known as a soft copy terminal. A printing device is usually required to provide a hard copy of the output.

Sound Systems

Voice response systems enable the computer to talk to its users. It consists of an audio-response device that produces the audio output.

Voice response systems are typically of two types: voice reproduction system and speech synthesiser.

NOTES

NOTES

Voice reproduction systems

Voice reproduction systems produce an audio output by selecting the appropriate response from a predefined set of responses. These responses may be in the form of speech (words or phrases spoken by human beings), musical sounds, alarms, or other sounds. The pre-recorded responses are first converted into digital data and stored in the computer. Once the appropriate response is selected, it is converted back to analog form to produce the audio output.

This is an appropriate method where standard replies to requests for information are all that is required. Potential applications of voice response systems include automatic answering machines, audio help on how to operate a system, enquiry on product availability, talking alarm clocks and the like.

Speech synthesizers

Text is converted into spoken words by speech access systems. A synthesizer forms the hardware component of the system that takes care of the speaking, and the screen access program forms the software component that gives directions to the synthesizer. Some speech access programs and synthesizers are sold together as a single package, while others are sold independently.

Users can access commercial software applications and convert the display or text into audio output with the help of the screen access software. It allows users to listen to their keystrokes spoken aloud and read the display when desired.

Phonemes are the smallest segments of sounds from which sound output is produced. The meaning of a word alters if even a single phoneme is changed. For instance, changing the last phoneme in the word 'bat' can modify the word to 'bad'. Although there are believed to be about forty-four phonemes in the English language, it is impossible to arrive at an accurate figure because of the subjective nature of the definition.

Synthetic speech has many applications, particularly for the disabled: it offers computer access to the visually challenged and supports augmentive communication. The latter uses computer-based technology for facilitating personal communication. This means that speech disabled people can indicate the utterances to be communicated through certain devices. Voice is substituted in such devices that use synthetic speech as a medium of communication. Cosmologist Stephen Hawking uses one of these devices.

Computer access to the visually challenged relies on the convertability of text into speech. A 'screen reader' assists in carrying out this function. A 'screen reader' refers to software that runs along with other programs and captures whatever is displayed on the screen by them. The greatest merit of a screen reader is its ability to work with standard application software; this ensures that there is no need to develop, say, talking spreadsheets or talking word processors.

Recently, along with the price of all computer technology, the price of speech synthesizers has also declined greatly. Currently, improving the quality is at the centre of R&D, and some progress has been made. Even now synthesizers are available that produce voices that are similar to recordings of human speech to a

large extent. But they still cannot carry out real time conversion of text to speech. Nevertheless, there is no doubt that in time, this too will be possible with the help of innovative technology.

Printers

Printers serve to produce output on paper. A large variety of commercially available printers exist today (estimated to be 1500 different types). These printers can be classified into categories based on:

- Printing technology
- Printing speed
- Printing quality

Printing technology: Printers can be classified as non-impact or impact printers, based on the technology they use for producing output.

Fairly similar to the standard printing mechanism used in a typewriter, impact printers work by striking a hammer on the paper through an inked ribbon. Such printers create images by touching the paper. Character printers and dot matrix printers fall under this category.

The mechanism of non-impact printers is such that the paper is not touched when the images are created. Instead, electrical, heat or chemical signals are used for etching symbols on paper. Specially treated or coated paper is required for many non-impact printers. Inkjet, laser and thermal printers fall under this category of printers.

Printing speed: This refers to the number of characters printed in a unit of time. Based on speed these may be classified as character printer (prints one character at a time), line printers (print one line at a time), and page printers (print the entire page at a time). Printer speeds are therefore measured in terms of characters-per-second or cps for a character printer, lines-per-minute or lpm for a line printer, and pages-per-minute or ppm for a page printer.

Printing quality: This is determined by the resolution of printing and is characterized by the number of dots that can be printed per linear inch, horizontally or vertically. It is measured in terms of dots-per-inch or dpi. Printers can be classified as near-letter-quality or NLQ, letter-quality or LQ, near-typeset-quality or NTQ, and typeset-quality or TQ based on their printing quality. NLQ printers of resolutions of about 300 dpi, LQ of about 600 dpi, NTQ of about 1200 dpi, and TQ of about 2000 dpi are available. NLQ and LQ printers are used for ordinary printing in day-to-day activities, while NTQ and TQ printers are used to produce top-quality printing, typically required in the publishing industry.

The section that follows explains the working of some commonly used printers.

Dot Matrix

Dot matrix printers were the most popular impact printers used in personal computing. These printers use a print head consisting of series of tiny pins that strike an ink-coated ribbon, and thus transfer the ink to the paper at the point of

NOTES

impact. Characters thus produced are in a matrix format. The shape of each character, i.e. the dot pattern, is obtained from information held electronically. (Fig. 1.19(a) and 1.19(b))

NOTES

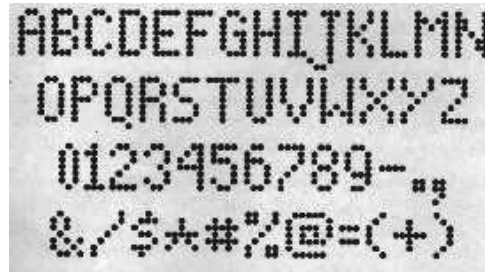


Fig. 1.19(a) Characters formed using Dots



Fig. 1.19(b) Dot Matrix Printer

The speed, versatility and ruggedness, combined with low cost, tend to make such printers particularly attractive in the personal computer market.

Typical printing speeds in case of dot matrix printers range between 40 – 1000 cps (characters-per-second). The one major shortcoming of dot matrix printers is that they do not offer a superior quality print.

Inkjet

In inkjet printers, ink is directly sprayed onto the paper by means of a series of nozzles. These therefore fall under the category of non-impact printers (Figure 1.20).



Fig. 1.20 Inkjet Printer

The print head of an inkjet printer consists of a number of tiny nozzles that can be selectively heated up in a few microseconds by an IC register. When this happens, the ink near it vapourizes and is ejected through the nozzle to make a dot on the paper placed in front of the print head. The character is printed by selectively heating the appropriate set of nozzles as the print head moves horizontally.

If you look at a paper printed through an inkjet printer, you will notice that:

- the size of the dots is exceedingly small (typically 50-60 microns in diameter); in fact, smaller than the diameter of a human hair (70 microns)
- the position of the dots is extremely precise, with upto a 1440×720 dpi resolution

- it is possible to combine different colours within the dots for creating photo-quality images

Inkjet printers are slower than dot-matrix printers (40 – 300 cps), cheaper to buy but are more expensive in running costs (the ink cartridge cost is considerably higher than that of the DMP ribbon) and are used by people/organisations where the speed of printing is not the most important factor.

Laser

Laser printers use heat, static electricity and dry ink (toner) to place and bond the ink onto the paper (Figure 1.21). They use a combination of laser and photocopier technology. In order to print an object, the laser beam is deflected onto a drum's photosensitive surface, after which, the toner is attracted to the image by the latent image. It is then electro-statistically transferred to the paper and set into a permanent image.



Fig. 1.21 Laser Printer

Laser printers are capable of converting computer output into print, page by page. Since characters are formed by very tiny ink particles, they can produce very high quality images (text and graphics); they generally offer a wide variety of character fonts, and are silent and fast in use.

Laser printers are faster in printing speed than the other printers discussed above. Their speeds range between about ten pages per minute and 200 pages a minute, depending upon the make/model.

Laser is high quality, high speed, high volume, and non-impact technology that works on pre-printed stationary or plain paper. Although the laser technology involves relatively more expenditure, it is still gaining popularity because of the speed, quality and noiseless nature of its operations.

Plotters

A plotter is used for producing graphical output on paper. It is a device capable of producing charts, drawings, graphics, maps etc. It is much like a printer but is designed to print graphs instead of alphanumeric characters.

NOTES

NOTES

Based on the technology used, plotters may be categorized as electrostatic plotters or pen plotters. While in the latter, an ink pen is attached for drawing images, the functioning of the former is similar to laser printers. The image is produced by charging the paper with a high voltage. This voltage attracts the toner, which is then melted on the paper with heat. Electrostatic plotters are fast, but the quality is generally considered to be poor when compared to pen plotters. This is why pen plotters are more extensively used as compared to electrostatic plotters.

Flatbed plotters and drum plotters constitute the most commonly used plotters.

Flatbed Plotters

Flatbed plotters have a flat base like a drawing board on which the paper is laid (as shown in Fig. 1.22(a)). One or more arms, each of them carrying an ink pen, moves across the paper to draw. The arm movement is controlled by a microprocessor (chip). The arm can move in two directions, one parallel to the plotter and the other perpendicular to it (called the X and Y directions). With this kind of movement, it can move very precisely to any point on the paper placed below.

The computer sends the commands to the plotter which are translated into X & Y movements. The arm moves in very small steps to produce continuous and smooth graphics.

The size of the plot in a flatbed plotter is limited only by the size of the plotter's bed.

The advantage of flatbed plotters is that the user can easily control the graphics. He can manually pick up the arm anytime during the production of graphics and place it in any position on the paper to alter the position of graphics to his choice.

The disadvantage here is that flatbed plotters occupy a large amount of space.

Drum Plotters

Drum Plotters use a drum revolver to move the paper during printing (as shown in Figure 1.22(b)). The arm carrying a pen moves only in one direction, perpendicular to the direction of the motion of the paper. Thus, in drum plotters, while one dimension or axis is provided by the pen moving in a single track or axis, the other is provided by the paper moving on a cylindrical drum. The combination of the pen and paper movement creates the graphics.

Thus, the graph's size is restrained only by the drum's width, without there being any restrictions on the length of the graph.

Drum plotters are very compact and lightweight as compared to flatbed plotters. This is one of the advantages of such plotters.

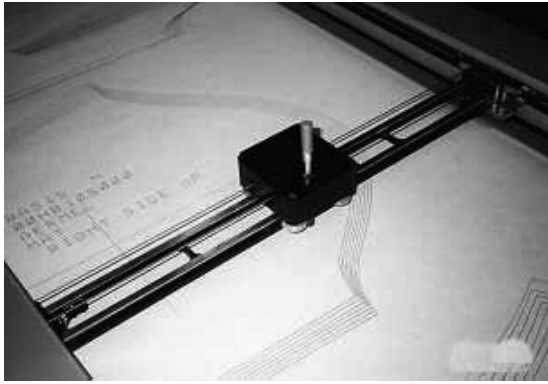


Fig. 1.22(a) Top View of a Flatbed Plotter



Fig. 1.22(b) Drum Plotter

NOTES

The disadvantage, however, is that the user cannot freely control the graphics when they are being created.

Plotters are more expensive than printers. Typical application areas for plotters include CAE (computer-aided engineering) applications such as CAM (computer-aided manufacturing) and CAD (computer-aided design), architectural drawing and map drawing.

1.4 MEMORY

The most common properties used for characterizing and evaluating the storage unit of the computer system are expressed below:

1. **Storage capacity:** Represents the size of the memory. It is the amount of data that can be stored in the storage unit. Primary storage units have less storage capacity as compared to secondary storage units. The capacity of internal memory and main memory can be expressed in terms of the number of words or bytes. The capacity of external or secondary storage, on the other hand, is measured in terms of bytes.
2. **Storage cost:** Cost is another key factor that is of prime concern in a memory system. It is usually expressed per bit. It is obvious that lower costs are desirable. It is worth noting that with the increase in the access time for memories, the cost decreases.
3. **Access time:** The time required to locate and retrieve the data from the storage unit. It is dependant on the physical characteristics and access mode used for that device.

Primary storage units have faster access time as compared to secondary storage units.

4. **Access mode:** Memory comprises various locations. Access mode is the mode in which information is accessed from the memory. The user can access memory devices in any of the following ways:

(a) *Random access memory (RAM):* This refers to the mode in which any memory location can be accessed in any order in the same amount

NOTES

of time. Ferrite and semiconductor memories, which usually constitute the primary storage or main memory, are of this nature.

- (b) *Sequential access*: Memories that can be accessed only in a pre-defined sequence are sequential access memories. Here, since sequencing through other locations precedes the arrival at a desired location, the access time varies according to the location. Information on a sequential device can be retrieved in the same sequence in which it was stored. Songs stored on a cassette, that can be accessed only one by one, are an example of sequential access. Typically, magnetic tapes are sequential access memory.
- (c) *Direct access*: Sometimes, the information is neither accessed randomly nor in sequence but something in between. In this type of access, a separate read/write head exists for each track, and on a track, you can access the information serially. This semi-random mode of access exists in magnetic disks.

5. **Permanence of storage**: If the storage unit can retain the data even after the power is turned off or interrupted, it is termed as non-volatile storage. And, if the data is lost once the power is turned off or interrupted, it is called volatile storage. It is obvious from these properties that the primary storage units of the computer systems are volatile, while the secondary storage units are non-volatile. A non-volatile storage is definitely more desirable and feasible for storage of large volumes of data.

Static and Dynamic RAM

The main memory is the central storage unit in a computer system. It is a relatively large and fast memory. It is used to store programs and data during computer operations. The principal technology used for the main memory is based on semiconductor-integrated circuits. There are two possible modes in which the integrated circuit RAM chips are available. These modes are: static and dynamic.

The static RAM (SRAM) stores binary information using clocked sequential circuits. The stored information remains valid only as long as power is applied to the unit. On the other hand, dynamic RAM (DRAM) stores binary information in the form of electric charges that are applied to capacitors inside the chip. The stored charge on the capacitors tends to discharge with time and so must be periodically recharged by refreshing the dynamic memory. The dynamic RAM offers larger storage capacity and reduced power consumption. Therefore, large memories use dynamic RAM, while static RAM is mainly used for specialized applications.

The different types of memory discussed above are both of the read/write type. What about a memory where only one of the operations is possible, e.g., if we allow only reading from the memory (cannot change the information in the memory)? The memory might have some major importance; like an important bit

of the computer's operating system which normally does not change can be stored in this kind of memory. Such a memory is called ROM (Read Only Memory).

Read-Only Memory (ROM)

Most of the memory in a general-purpose computer is made of RAM integrated circuit chips, but a portion of the memory may be constructed using ROM chips. Originally, RAM was used to refer to random-access memory, but now we use the term read/write memory to distinguish it from read-only memory (since ROM is also random access). RAM is used for storing the bulk of the programs and data that are subject to change, while ROM is used to store programs that are permanently resident in the computer and do not change once the production of the computer is completed.

Among other things, the ROM portion of the main memory is used for storing an initial program called the bootstrap loader, whose function is to get the computer software operating when power is turned on. Since RAM is volatile, its contents are destroyed when power is turned off. The contents of ROM remain unchanged even after the power is turned off and on again.

Read-only memories can be manufacturer-programmed or user-programmed. When the data is burnt into the circuitry of the computer by the manufacturer, it is called manufacturer-programmed ROM. For example, a personal computer manufacturer may store the boot program permanently in the ROM chip of the computers manufactured by it. Such chips are supplied by the manufacturer and are not modifiable by users. This is an inflexible process and requires mass production. Thus, a new kind of ROM, known as Programmable Read-only Memory (PROM), was designed. This is also non-volatile in nature. It can be written only once using some special equipment. The supplier or the customer can electrically perform the writing process in PROM.

In both ROM and PROM, you can perform write operations only once and you cannot change whatever you have written. But what about the cases where you mostly read but also write a few times? Another type of memory chip called EPROM (Erasable Programmable Read-only Memory) was developed to take care of such situations. EPROMs are typically used by R&D personnel who experiment by changing micro-programs on the computer system to test their efficiency.

Further, EPROM chips are of two types: EEPROMs (Electrically EPROM) in which high voltage electric pulses are used to erase stored information, and UVEPROM (Ultra Violet EPROM) in which stored information is erased by exposing the chip for a while to ultraviolet light.

NOTES

Figure 1.23 summarizes the various types of Random Access Memories.

NOTES

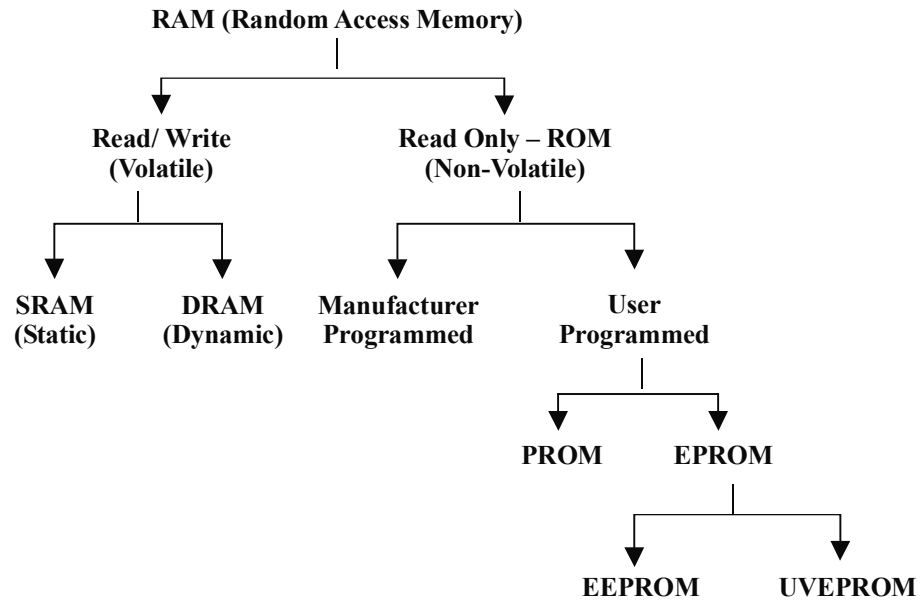


Fig. 1.23 Types of Random Access Memories

Cache Memory

Cache memories are small, fast memories placed between the CPU and the main memory. They are faster than the main memory with access times closer to the speed of the CPU. Caches are fast, but very expensive. So, they are used only in small quantities. As an example, caches of size 64K, 128K are normally used in PC-386 and PC-486, which can have 1 to 8 MB of RAM or even more. Cache memories provide fast speed memory retrieval without compromising the size of the memory.

If the memory is so small, would it be advantageous to increase the overall speed of memory? This can be answered with the help of a phenomenon known as locality of reference. Let us examine what this means.

Locality of reference: If we analyse a large number of typical programs, we would find that memory references at any given interval of time tend to be confined to a few localized areas in the memory. This phenomenon is called the property of locality of reference. This is true because most of the programs typically contain iterative loops (like 'for' or 'while' loops). During the execution of such programs, the same set of instructions (within the loop) are executed many times. The CPU repeatedly refers to the set of instructions in the memory that constitute the loop. Every time a specific subroutine is called, its set of instructions is fetched from the memory. Thus loops and subroutines tend to localize the references to memory for fetching instructions.

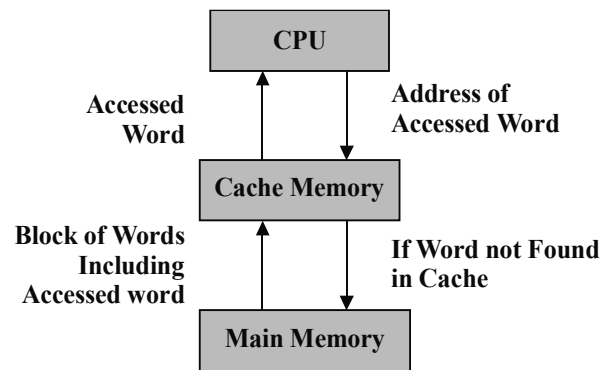


Fig. 1.24 Functioning of the Cache Memory

Figure 1.24 explains the functioning of the cache memory.

Based on the locality of reference, we understand that the cache has a copy of certain portions of main memory. First, the memory read or write operation is checked with the cache, and in case of availability of desired data in the cache, it is used directly by the CPU. Otherwise, a block of words is read from main memory to cache and the word is then used by the CPU from cache.

Secondary Storage Devices

As discussed earlier, RAM is a volatile memory with limited storage capacity. The cost of RAM is also relatively higher as compared to secondary memory. Logic dictated that a relatively cheaper media, showing some sort of permanence of storage, be used. As a result, additional memory called *external* or *auxiliary memory* or *secondary storage* is used in most computers.

The magnetic medium was found to be long lasting and fairly inexpensive, and therefore became an ideal choice for large storage requirements. The use of magnetic tapes and disks as storage media is very common. As optical technology is advancing, optical disks are turning out to be one of the major secondary storage devices.

Magnetic Storage Devices

Magnetic tapes are used for storing files of data that are sequentially accessed or not used very often and are stored off line. They are typically used as backup storage for archiving of data.

In case of magnetic tapes, a tape (plastic ribbon usually 1/2 inch or 1/4 inch wide and 50 to 2400 feet long) is wound on a spool and its other end is threaded manually on a take-up spool. The beginning of the tape (BOT) is indicated by a metal foil called a *marker*. When a write command is given, a block of data (records are usually grouped in blocks of two or more) is written on the tape. The next block is then written after a gap (called Inter Block Gap or IBG). A series of blocks are written in this manner. The end of tape (EOT) is indicated by an end-of-tape marker which is a metal foil stuck in the tape. After the data is written, the

NOTES

tape is rewound and kept ready for reading. Figure 1.25(a) shows the data organization on a magnetic tape. Figure 1.25(b) and 1.25(c) show magnetic tape reels and magnetic tape cartridges.

NOTES

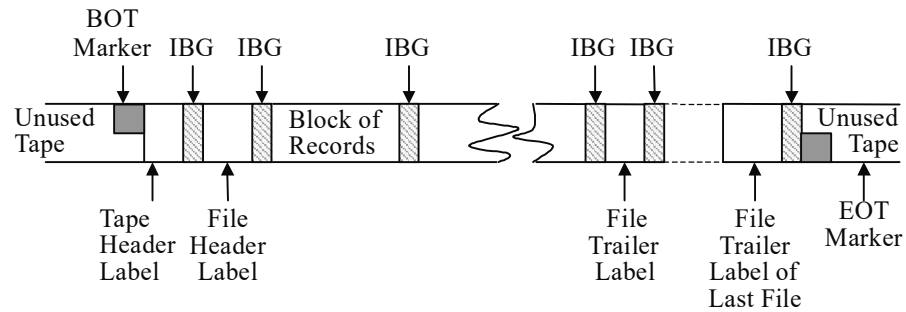


Fig. 1.25(a) Data Organization on a Magnetic Tape

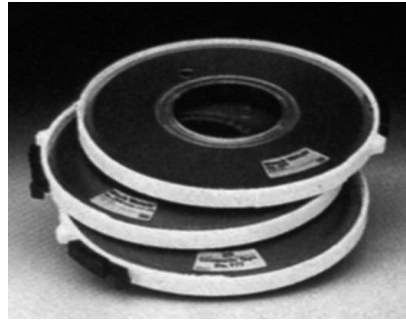


Fig. 1.25(b) Magnetic Tape Reel



Fig. 1.25(c) Magnetic Tape Cartridge

The tape is read sequentially, i.e., data can be read in the order in which the data has been written. This implies that if the desired record is at the end of the tape, all the earlier records have to be read before it is reached. A typical example of a tape can be seen in a music cassette, where, to listen to the fifth song one must listen to, or traverse, the earlier four songs. The access time of information stored on tape is therefore very high as compared to that stored on a disk.

The storage capacity of the tape depends on its data recording density and the length of the tape. The data recording density is the amount of data that can be stored or the number of bytes that can be stored per linear inch of tape. The data recording density is measured in BPI (Bytes per inch).

Thus,

$$\text{Storage capacity of a tape} = \text{Data recording density} \times \text{Length of tape}$$

It is worth noting that the actual storage capacity for storing user data, is much less owing to the file header labels, file trailer labels, BOT and EOT markers, and the use of IBGs.

Some commonly used magnetic tapes are:

- 1/2 inch tape reel
- 1/2 inch tape cartridge

- 1/4 inch streamer tape
- 4 mm DAT (Digital Audio Tape) – typical capacity of 4GB to 14 GB

Magnetic Disks

Magnetic disks are direct-access medium, and so are the most popular online secondary storage devices. Direct-access devices are also called random-access devices because information is literally available at random or in any order. There is direct access to any location on the device. Thus, approximately equal access time is required for each location. An example of this is a music CD where if you wish to listen to the fifth song, you can directly select the fifth track without having to fast forward the previous four.

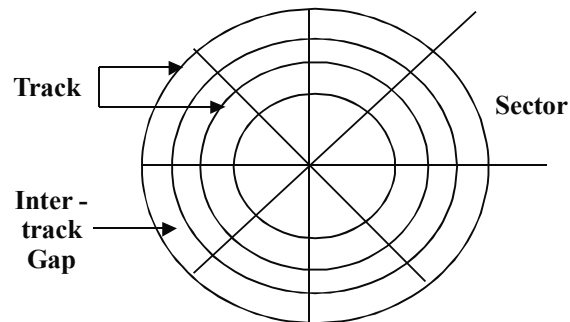


Fig. 1.26 Logical Layout of a Magnetic Disk

A magnetic disk refers to a circular plate made of metal or plastic and coated with magnetized material (as shown in Figure 1.26). Often both sides of the disk are used. Data is recorded on the disk in the form of magnetized and non-magnetized spots (not visible to the naked eye) representing 1s and 0s.

Disk Devices

A **disk drive** is a peripheral device used to store and collect information. It can be removable or fixed, high capacity or low capacity, fast or slow speed, and magnetic or optical.

Structurally, a drive is the object inside which a disk(s) is either permanently or temporarily stored. While a disk contains the media on which the data is stored, a drive contains the machinery and circuitry required for implementing the read / write operations on the disk.

The disk looks literally like a flat circular disk. The computer writes information onto the disk, where it is stored in the same form as it is stored on a cassette tape. Disks, as such, are just magnetically coated rolls or circular disks which are divided into sectors and tracks. The data is accordingly stored and numbered with respect to the track and sector number on the disk. Only the structure of the media is different from one to another. Examples of removable disk drives are DVD, CD-ROM, floppy disk drive, etc. The examples of non-removable disk drives include hard disk.

The method of accessing data could be sequential access (Magnetic Tape Drives) or random access (HDD, DVD), where the read/write head can directly go to any location on the disk and perform action.

NOTES

NOTES

Diskettes

Based on the size and packaging of the disks, they can be classified into two types – floppy disks and hard disks. Further, disks that are permanently attached to the unit assembly and cannot be removed by the occasional user are called hard disks. A drive using removable disks is called a floppy disk drive.

Floppy disks

The disks used with a floppy disk drive are small removable disks made of plastic, and coated with magnetic recording material. There are two sizes commonly used, with diameters of 5.25 and 3.5 inches.

- The 5.25 inch disk is a 5.25 inch diameter floppy disk. Earlier, such disks recorded data only on one side and were called single-sided (SS) disks. Today both the surfaces are used for recording and are called double-sided (DS) disks. These are available in two capacities – double density (DD), and high density (HD), where density refers to the number of bits that can be stored per square inch area.

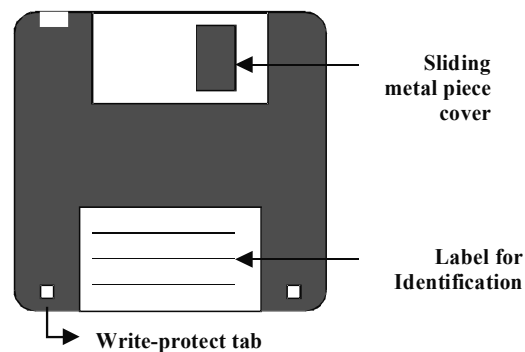


Fig. 1.27 A 3.5 Inch Floppy Disk

- The 3.5 inch disk is a disk of 3½ inch diameter. These record data on both sides and are therefore double sided disks. These disks come in three different capacities – double density, high density and very high density. These are smaller and can store more data than can the 5.25 inch disks.

The storage capacity for any disk can be calculated as:

$$\text{Storage capacity} = \text{Number of recording surfaces} \times \text{Number of tracks per surface} \times \text{Number of sectors per track} \times \text{Number of bytes per sector}$$

Thus, for a 3.5 inch high density disk which has 80 tracks, 18 sectors/ track, and 512 bytes/sector, the disk storage capacity can be calculated as follows:

$$2 \times 80 \times 18 \times 512 = 14,74,560 \text{ bytes or } 1.4 \text{ MB (approximately)}$$

Table 1.1 provides the necessary details and associated storage capacities of various kinds of floppy disks.

Floppy disks were extensively used in personal computers as a medium for distributing software to computer users. Nowadays, CDs or DVDs are used for that purpose.

Table 1.1 Storage capacities of floppy disks

Size (diameter in inches)	No. of Recording Surfaces	No. of Tracks	No. of Sectors/Tracks	No. of Bytes/Sector	Storage Capacity (approx)
5.25	2	40	9	512	3,68,640 bytes or 360kB
5.25	2	80	15	512	12,28,800 bytes or 1.2 MB
3.5	2	40	18	512	7,37,280 bytes or 720 kB
3.5	2	80	18	512	14,74,560 bytes or 1.4 MB
3.5	2	80	36	512	29,49,120 or 2.8 MB

NOTES

Hard Disks

Unlike floppy disks, hard disks are made up of rigid metal. The sizes for the disk platters range between 1 to 14 inches in diameter. Depending on the way they are packaged, hard disks can be categorised as disk packs or Winchester disks.

- **Disk packs** consist of two or more hard disks mounted on a single central shaft. Because of this, all disks in a disk pack rotate at the same speed. It consists of separate read/write heads for each surface (excluding the upper surface of the topmost disk platter and the lower surface of the bottommost disk platter, as mentioned earlier). Disk packs are removable in the sense that they can be removed and kept offline when not in use (typically stored away in plastic cases). They have to be mounted on the disk drive before they can be used. Thus different disk packs can be mounted on the same disk drive at different instances, thereby providing virtually unlimited (modular) storage capacity.

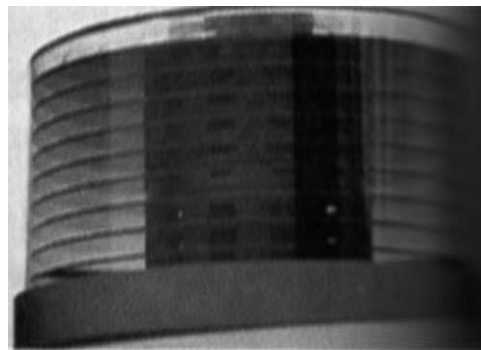


Fig. 1.28 A Disk Pack

- **Winchester disks** also consist of two or more hard disk platters mounted on a single central shaft but are of the fixed type. The disk platters are sealed in a contamination-free container. Due to this fact, all the disk platters, including the upper surface of the topmost disk platter and the lower surface of the bottommost platter, are used for storing data. So, even though Winchester

disks have limited storage capacity as opposed to disk packs, they can store larger amounts of data as compared to the same number of disk platters.

NOTES

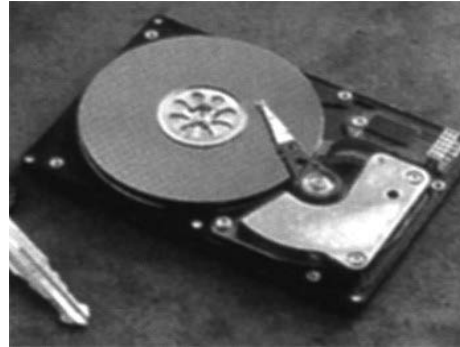


Fig. 1.29 A Winchester Disk

Another kind of disk called the zip disk is very common today. This consists of a single hard disk platter encased in a plastic cartridge. Such a disk typically has a capacity of about 100 MB. The zip drive can further be fixed or portable. The fixed zip drive is permanently connected to the computer system while the portable ones can be carried around and connected to any computer system for the duration of its use. In both cases, however, the zip cartridge (the actual storage medium) is portable just like a floppy, albeit with a nearly 100 times larger storage capacity.



Fig. 1.30 Zip Disks and Zip Drive

Optical disks

Optical disks are storage devices with huge storage capacities. They are a relatively new storage medium and use laser beam technology for writing and reading of data.

Optical disks consist of one large track that starts from the outer edge and spirals inward towards the centre (this is unlike the magnetic disk in which tracks are concentric circles on the disk platter). An optical disk is also split into sectors but these are of the same length regardless of their location on the track. Data is therefore packed at maximum density over the disk.

The storage capacity of an optical disk is determined as follows:

$$\text{Storage Capacity} = \text{Number of sectors} \times \text{Number of bytes per sector}$$

(Note that we do not consider the number of tracks since there is only one track in this case)

Thus, a 5.25 inch optical disk having 3,30,000 sectors and storing 2,352 bytes per sector, will have a storage capacity

$$3,30,000 \times 2352 = 77,61,60,000 \text{ bytes or } 740 \text{ MB (approx.)}$$

The technology used in optical disks uses laser beams to write and read data as opposed to the read/write head used in magnetic disks. Data is recorded by etching microscopic pits (burnt surface) on the disk surface. A high-intensity laser beam is used to etch the pits, while for retrieving data, a low-intensity laser beam is used.

NOTES



Fig. 1.31 An optical disk and disk drive

There are three optical memory devices that are becoming increasingly popular in various computer applications: CD-ROM, WORM, and Erasable Optical disks. We shall discuss these below.

CD-ROM: The compact disk read-only memory (CD-ROM) is a direct extension of the audio CD. It is generally made from a resin called polycarbonate that is coated with aluminium to form a highly reflective surface. The information on a CD-ROM is stored as a series of microscopic pits on the reflective surface (using a high-intensity laser beam). The process of recording information on these disks is known as 'mastering'. This is so-called because this master disk is then used to make a 'die' that is used to make copies.

The information is retrieved from a CD-ROM using a low-powered laser that is generated in an optical disk drive unit. The disk is rotated and the laser beam is aimed at the disk. The intensity of the laser beam changes when it encounters a pit. A photo-sensor detects the change in intensity, hence recognizing the digital signals that are recorded on the surface of the CD-ROM and converts them into electronic signals of 1s and 0s.

As the name suggests, information stored in a CD-ROM can be read only. It cannot be modified in any way. It is therefore useful for applications in which there is a database of information that is useful as it is and does not need changing in any way, for example, a directory such as Yellow Pages. CD-ROMs are very handy in the distribution of large amounts of information to a large number of users. The strengths of CD-ROMs lie in the fact that they provide for:

- large storage capacity for information/data.
- fast and inexpensive mass replication.
- suitable for archival storage since they are removable disks.

NOTES

The weaknesses of CD-ROMs are:

- they are read-only and cannot be updated.
- access time in them is greater than that of a magnetic disk.

WORM: The drawbacks of CD-ROM were partially resolved by the introduction of WORM ('write-once, read many').

In some applications, only a limited copies of compact disks are required to be made. This makes the CD-ROM production economically not viable. This is because manufacturers duplicate CD-ROMs by using expensive duplication equipment. To overcome such a problem, write-once read-many CDs have been developed.

WORM disks allow users to create their own CDs by using a CD-Recordable (CD-R) drive. This can be attached as a peripheral device to the computer system. WORM disks recorded like this can be read by any CD-ROM drive.

Erasable optical disk: The erasable optical disk is the latest development in optical disks. Like magnetic disks, the data in the erasable optical disk can be changed repeatedly. Erasable optical disks are therefore also known as rewritable optical disks.

These disks integrate the magnetic and optical disk technologies to enable rewritable storage with the laser-beam technology and so are also called magneto-optical disks. In such systems, a laser beam is used along with a magnetic field to read or write information on a disk that is coated with a magnetic material.

To write, the laser beam is used to heat a specific spot on the magnetic coated material. At this increased temperature, a magnetic field is applied so as to change the polarization of that spot, thereby recording the data that is required. It may be noted here that this process does not cause any physical changes in the disk. Hence, it can be repeated many times. The degree of rotation of the polarized laser beam reflected from the surface is detected to perform reading function. This implies that as the disk spins, the polarized spots pass under the laser beam, and depending on their orientation or alignment, some of them reflect the light while others scatter it. This produces patterns of 'on' and 'off' that are converted into electronic signals of binary 1s and 0s.

The capacity of an erasable disk is very high in comparison to a magnetic disk; for example, a 5.25 inch optical disk can store around 650 MB of data while Winchester disks normally can store a maximum capacity of 320 MB. This is why magneto-optical disks are ideal for multimedia applications that require large storage capacities.

DVD: Known as the Digital Versatile Disk or the Digital Video Disk, it has the same physical dimensions as that of a CD-ROM, but it can hold up to 4.4GB data on a Single layer disk and up to 8.47GB on a Dual layer disk with the maximum data transfer of 27MB/s at 20x speeds. The laser used to read/write data on a DVD is much more precise and has a wavelength of 650nm, which is one reason why a DVD can hold more data.

HD-DVD: This is a high density, mostly single sided, double layered optical disc that can hold up to 15GB on a single layer and 30GB on a dual layer disc. The read/write speed on an HD-DVD varies between 36 MBPS to 72 MBPS. These were primarily designed for the storage of high definition videos and large volumes

of data. The basic look and feel of an HD-DVD drive and disk is the same as that of a CD-ROM and DVD except that it uses a laser of a different wavelength and the microscopic structure of storage on a disk is different.

Blu-Ray: Another high density optical storage media format is gaining increasing popularity these days. Its main uses are high-definition video and data storage. A dual layer Blu-ray Disc can store 50 GB, almost six times the capacity of a double-layer DVD (or more than 10 times if single-layer). The data can be read from 36 MBPS to 432 MBPS which is much higher than HD-DVD. When used for HD video playback, the video is encoded on it in MPEG-2, AVC, VC-1 (H.264) format, which is the same as HD-DVD.

With the wide variety of storage choices available and constant increase in their number due to changes in technology, it is impossible to say that a single medium of storage can suit the needs of all. Therefore, it is necessary to make the choice of right media based upon factors like capacity, speed, durability, life span and cost of media.

Tape Drives

Computer Output Microfilm (COM)

COM refers to a process characterized by copying/printing data from media located on personal computers, mini, or mainframe computers onto a microfilm. It comprises a high-speed recorder, which transfers digital data onto a microfilm applying laser technology, and a processor which develops the microfilm once exposed to the light source.

A computer output microfilm device translates information normally held on magnetic tape into miniature images on a microfilm (also called microfiche—'fiche' pronounced as 'fish'). The device displays information as characters on a CRT screen and then using photographic methods, records the display onto the film. Drawings and images can also be displayed along with narrative text.

A special reader/printer can be subsequently used to view the processed film. The reader operates on a 'back projection' principle displaying one frame at a time on a translucent screen, typically about A4 size. The printer can then be used selectively to produce a hard copy of what is presented on screen.

Figure 1.32 identifies the various steps in COM production.

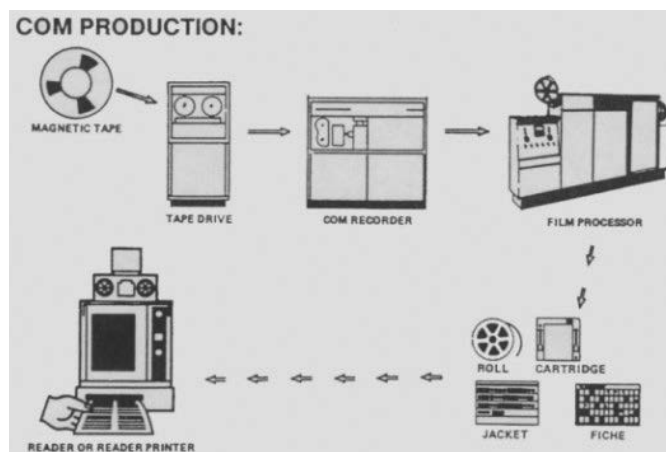


Fig. 1.32 Steps in a COM Production

NOTES

NOTES

A COM system provides an easy and compact way of recording and storing information, and subsequently retrieving desired pieces of information. It offers various advantages like reduction of paper, reduction in cost (since it is cheaper than most electronic media), improved quality (COM technology provides superior image quality), and electronic record retention/archiving.

COM is best suited for data requiring long-term storage. This is because microfilm is less volatile than magnetic media like disks and tapes. COM stores data in a very compact format. It is to be noted that up to 270 pages can be contained in a single 4 × 6 inch fiche.

Conversion of magnetic tapes to microfilms is cost-effective for closed files. However, in case of highly active data or data requiring regular updating, using microfilms may not be as efficient as retaining the information online. It is therefore useful for data that must be archived for long periods of time and referenced only occasionally, e.g., information that must be archived to comply with legal regulations, information maintained by insurance companies, banks, government agencies, and various other organizations of this type.

Measuring Drive Performance

The performance of a disk is measured in terms of how fast it can read or write data. Over the years there have been changes in disk drive interface, rotation speeds, number of heads and cylinders and storage format, all of which have led to a decrease in data access time.

The various types of disks currently available in the market are:

1. IDE – Integrated Drive Electronics
2. EIDE – Enhanced Integrated Drive Electronics
3. SCSI – Small Computer System Interface
4. SATA I & II – Serial Advanced Technology Attachment

There are two standard methods for accessing and writing data on a disk – sequential access and random access.

Sequential Access is when you read or write to the disk blocks in sequential or continuous order, that is, one block after another. Examples of where Sequential Access is used in computing or data storage would be the backing up of data onto tape drives or the process of writing data onto CDs and DVDs. Any storage medium based on magnetic tape, VHS, audio cassette etc, are read and written by Sequential Access.

Random Access, as the name suggests, is performed when the hard drive head needs to read/write data from/at various locations on the disk. In this case, the disk heads move rapidly from one place to another and the seek time to access data is increased because it involves mechanical operations. Most of the disk operations performed during routine computer work are random access. This is also the reason why random access time is more important while measuring disk performance than sequential access time. While data is written onto optical media sequentially, data on CDs and DVDs can be read randomly.

For Random Access, the *average seek time* and *average latency time* are added to come up with the total time it takes for the disk to read and write data on it.

The average seek time is the time it takes to move the head arm from one position to another, and average latency time is the time it takes for the required data block to come under the head for the read/write operations. The average latency time depends on the rotations per minute (RPM) of the disk, which is the speed at which the magnetic or optical disk rotates.

NOTES

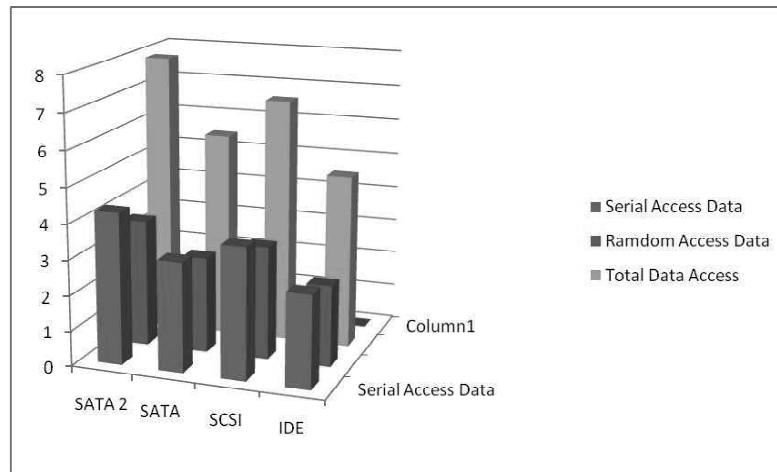


Fig. 1.33 Data accessed by Different Disk Drives

The above Bar Chart shows the relative amount of Data accessed by each type of Disk Drive.

1.5 COMPUTER LANGUAGES

A computer language is a language that can be understood by the computer. It is the computer's native language. A computer language consists of lexicon and syntax, i.e., characters, symbols and rules of usage that allow the user to communicate with the computer. Each and every problem to be solved by the computer needs to be broken down into discrete logical steps before the computer can execute it. The process of writing such instructions in a computer or programming language is called programming or coding.

All computer languages can, however, be classified under the following categories:

- Machine language (First-Generation Language or 1 GL)
- Assembly language (Second-Generation Language or 2 GL)
- High-Level language (Third-Generation Language or 3 GL)

Low-level Language

Low-level languages are basically machine codes or assembly languages that can easily be converted to run on a computer with a separate CPU. However, for this, it is translated into a machine code by an assembler program. Basically, assembly languages and machine languages are known as low-level languages. They do not hide the details of machines and bit patterns. They are often difficult to write the program. When mnemonic based low-level language replaces the instructions, these instructions are very difficult to read, write and remind. Writing

NOTES

codes in low-level languages is very difficult. There are an infinite number of such programs available in each language. There is no limit on the size of such programs and the average length of each program is generally infinite. This is the main reason that low-level language is not successful.

Machine Language (1 GL)

The computer understands only a binary-based language. As already explained, this is a combination of 0s and 1s. Instructions written using sequences of 0s and 1s are known as a machine language. First-generation computers used programs written in the machine language.

The machine language is very cumbersome to use and is tedious and time consuming for the programmer. It requires thousands of machine language instructions to perform even simple jobs like keeping track of a few addresses for mailing lists. Every instruction in the machine language is composed of two parts—the command itself, also known as the ‘operation code’ or opcode (such as add, multiply, move, etc.), and the operand, which is the address of the data that has to be acted upon.

Assembly Language (2 GL)

Assembly language was the first step in the evolution of programming languages. It used mnemonics (symbolic codes) to represent operation codes and strings of characters to represent addresses. Assembly language was designed to replace each machine code by an understandable mnemonic and each address with a simple alphanumeric string. A program written in the assembly language needs to be translated into the machine language before the computer can execute it. This is done by a special program called ‘assembler’ which takes every assembly language program and translates it into its equivalent machine code. The assembly language program is known as the source program while the equivalent machine language program is known as the object program. It may be useful to know that the assembler is a system program supplied by the computer manufacturer. Second-generation computers used assembly language.

High-level Languages (3 GL)

The lack of portability of programs (written using machine or assembly languages) among various computer systems led to the development of high-level languages. Since they allowed a programmer to overlook a lot of low-level particulars of the hardware of the computer system, they were called high-level language programs. They contained commands that were particularly suited to one type of application, for example, a number of languages were designed to process scientific or mathematical problems. Others emphasized on commercial applications. These languages varied very little between different computer systems, unlike machine or symbolic languages. However, a compiler or an interpreter program was required to translate these machine codes. The high-level program is called the source code while its equivalent machine language program is referred to as the object code. The following are the various types of high level languages.

FORTRAN: FORMula TRANslation (FORTAN) was the first high-level language developed by John Backus at IBM in 1956. FORTRAN has a number of versions with FORTRAN IV being one of the earlier popular versions. In 1977, the American National Standards Institute (ANSI) published standards for FORTRAN with a view to standardizing the form of the language used by manufacturers. This standardized version is called FORTRAN 77.

COBOL: COmmon Business Oriented Language (COBOL), the first language used for commercial applications, was developed under the leadership of Grace Hopper, a US Navy programmer, with a group of computer manufacturers and users in 1959. COBOL was standardized by ANSI in 1968 and in 1974. COBOL became the most widely used programming language for business and data processing applications.

BASIC: Beginner's All-purpose Symbolic Instruction Code (BASIC) was developed as a teaching tool for undergraduate students in 1966 by John Kemeny and Thomas Kurtz, two professors at Dartmouth College. Eventually BASIC was used as the main language among the personal computer users. One of the newer versions of BASIC, commonly known as Visual Basic, has also evolved from the original BASIC language. It contains various statements and functions that can be used to create applications for a Windows or a GUI environment.

PASCAL: PASCAL was designed by Niklaus Wirth, a Swiss professor, in 1971. It was developed as a better structured language used for teaching which Wirth named after the French mathematician Blaise Pascal, who also designed the first successful mechanical calculator. In addition to manipulation of numbers, PASCAL supports manipulation of vectors, matrices, strings of characters, records, files and lists, thereby supporting non-numeric programming. Hence, it has proved to be an attractive language for professional computer scientists. PASCAL has been standardized by International Standards Organization (ISO) and ANSI.

PL/1: Programming Language 1 or PL/1 was developed by IBM in the 1960s and was the first language that was attempted to be used for a variety of applications rather than one particular area, such as business or science or artificial intelligence.

LISP: LIST Processing (LISP) was developed in the early 1950s but was implemented in the 1959 by John McCarthy at the Massachusetts Institute of Technology. It became a standard language with the artificial intelligence community and was a program that could easily handle recursive.

C: C language was developed by Dennis Ritchie of Bell Laboratories in order to implement the operating system 'UNIX'.

C++: C++ was developed by Bjarne Stroustrup of Bell Laboratories by enhancing C. C++ is also used to write procedural programs, such as C but the reason for its increased popularity is perhaps because of its capability to handle rigours of object oriented programming. C and C++ are the most extensively used general-purpose languages among programming experts.

NOTES

NOTES

Java: Java is again an object oriented language like the C++ but is a simplified version with extra features. It is prone to less programming errors. It was developed for writing programs that could be safely and easily executed through the Internet. It is free from any kind of common virus threats. It is basically a network-oriented language that can develop Website pages with enhanced multimedia features using small Java programs known as Java applets, Java is a 'secure to use over' the Internet and is a platform independent language.

Fourth-Generation Language (4 GL)

The Fourth-Generation Languages are non-procedural. This means that they signify what needs to be accomplished but do not specify how to do it. Fourth-generation languages are characterized by simple English-like instructions and a quick and easy learning process. They are so user-friendly that with little training and practice an individual can write his/her own programs and generate the desired reports. The fourth-generation languages refer to software packages that are mostly written in one of the languages (FORMula TRANslation or FORTRAN, C, and so on) for any specific application. This language is also called command line language. Some of the commonly used 4 GL packages are dBase, FoxPro, Oracle, SQL (database management); WordStar, Microsoft Word, PageMaker (desktop publishing); Lotus 123, Microsoft Excel (electronic spreadsheets); AutoCAD (computer-aided design and drafting); IDEAS, PRO/E, Unigraphics, Solidworks (computer-aided design and solid modelling); ANSYS, NASTRAN, and ADINA (finite element analysis for engineering components).

Fifth-Generation Language (5 GL)

Fifth-Generation Languages are an outgrowth of research in the area of artificial intelligence. They are, however, still in their infancy. PROLOG (PROgramming LOGic) was designed in the early 1970s by Alain Colmerauer, French computer scientist, and Philippe Roussel, a logician. Logical processes can be program and deductions can be made automatically by using PROLOG. A number of other 5 GL have been developed for meeting specialized needs. Some of the more popular ones include Programmed Instruction Learning or Testing (PILOT) used to write instructional software; LOGO, a version of LISP, developed in the 1960s to help educate children about computers; String-Oriented Symbolic Language (SNOBOL), designed for list processing and pattern matching; and General Purpose System Simulator (GPSS), used for modelling environmental and physical events.

Interpreters and Compilers

Programs written in high-level languages need to be converted into machine language before the computer can execute them. 'Interpreters' and 'compilers' are translation or conversion programs that produce the machine code from high-level languages. The interpreter and compiler perform the same function but in fundamentally different ways. An interpreter translates the instructions of the program one statement at a time. This translated code is first executed before the interpreter begins work on

the next line (refer Figure 1.34). Thus, instructions are translated and executed simultaneously. The object code is not stored in the computer's memory for future use. The next time the instruction is to be used, it needs to be freshly translated by the interpreter. During repetitive processing of instructions in a loop, every instruction in the loop will need to be translated every time the loop is executed.

NOTES

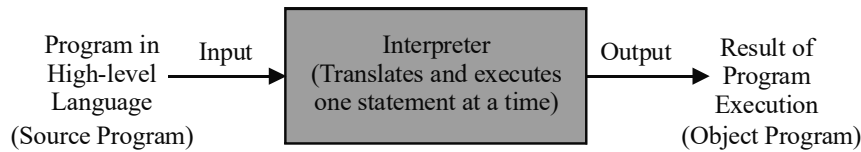


Fig. 1.34 Translation Process using an Interpreter

A compiler, on the other hand, takes an entire high-level language program and produces a machine code version out of it. This version is then run as a single program. Source statements are created, which are then compiled by running the appropriate language. These statements are converted into their corresponding machine code, which can then be executed by the computer. The object code can be stored in the computer's memory for executing in future. A compiler does not need to translate the source program every time it needs to be executed, thereby saving execution time (refer Figure 1.35).



Fig. 1.35 Translation Process using a Compiler

Check Your Progress

1. List any two basic operations of a computer.
2. What are the three essential parts of computer?
3. Name the various hand-held input devices.
4. What are electronic-card readers?
5. What are the different types of printers?
6. How is the storage unit of a computer system evaluated?
7. Sate about the disk drive.
8. Define the term of computer language.

1.6 OPERATING SYSTEM WITH DOS

Microsoft Disk Operating System (MS DOS) is a single user, single tasking operating system. DOS has a command-line, text-based/non-graphical user-interface commonly referred to as Character Based User Interface (CUI). When the computer is switched on, a small program checks all internal devices, electronic memory and peripherals. Once this process is completed, MS DOS is loaded.

NOTES

DOS Prompt: The DOS prompt known as the command prompt looks like C:\> or D:\> where 'C', 'D' represent the hard drives of the computer system. All commands are typed at the DOS prompt. ENTER key is pressed to view the output of the typed command. If the command is correctly typed desired output would be displayed. Otherwise an error message (bad command or filename/Invalid parameter) is displayed on the screen.

Limitations of MS DOS

1. It has a text based user interface where the commands have to be typed for each operation that the user wants to perform. The user is expected to remember the commands as well as their syntax.
2. It is a single-user, single-task operating system and the working is limited to one megabyte of memory. 640 kilobytes of the memory is used for the application program.
3. It does not allow using long file names. The user is restricted to eight-character file names with three-character extensions.

Below is the list of commonly used DOS commands. All DOS commands are case-insensitive (i.e., there is no difference whether you type COPY, copy or coPY). The same is true for the attributes, parameters and filenames. To view the complete description of each command type HELP followed by the command name, for example HELP COPY. Alternatively help can also be displayed using <command>/? (COPY/?).

Directory Related Commands

Command	Syntax	Explanation	Example	Notes
DIR	DIR Drive/ Directory - Name Name>	Displays all the sub-directories and files of the specified drive/directory. It also shows the size of the files and the date and time they were last modified.	C:\>DIR D:	Displays all the contents (files and directories) of the D: drive.
DIR/P	DIR Drive/ Directory -Name>/P	Displays the contents of directory one screen at a time and pauses until any other key is pressed to continue the display.	C:\>DIR DATA/P	Displays the contents of the 'DATA' directory by pausing the screen.
DIR/W	DIR <Drive/ Directory>/W	Displays the contents of the directory width-wise. It omits file size, date and time of creation of file so that more files can be displayed at one time on the screen.	C:\>DIR DATA/W	Displays the contents of the 'DATA' directory width-wise.
DIR/W/P	DIR <Drive/ Directory>/W/P	The Wide and Pause display option can be combined.	C:\>DIR DATA/W/P	Displays the contents of the 'DATA' directory width-wise and by pausing the screen.
CD	CD<Directory- Name> CD\ -Directly takes to the root directory.	Displays the name of the current directory if no parameter is specified with the command. Changes the current directory to the specified directory.	C:\>CD DATA\ SUBDATA	Changes the current directory to 'DATA\SUBDATA'.
MD	MD <Drive/ Directory-Name>	Creates a new directory in the specified location.	C:\>MD 'HELLO'	Creates a directory named 'HELLO' in the C: Drive.
RD	RD <Directory- Name>	Removes the specified directory.	C:\>RD HELLO	To remove a directory first you should come to one level above the current directory and then remove command should be given. This command will delete the 'HELLO' directory from the C: drive.
DELTREE	DELTREE <Directory-Name>	Deletes a directory and all the sub-directories and files in it.	C:\>DELTREE TEMP	Prompts the user for confirmation. If user selects 'Y' (Yes) then the directory 'TEMP' and all its sub-directories will be deleted.

Simple DOS Commands

Command	Syntax	Explanation	Example	Notes
COMMAND	COMMAND	Starts a new Window for the DOS command interpreter.	C:\>COMMAND	Starts a new Window for the DOS command interpreter.
EXIT	EXIT	Quits the <u>COMMAND.COM</u> program (command interpreter).	C:\>EXIT	Quits the command interpreter.
CLS	CLS	Clears the screen display completely leaving only the DOS prompt.	C:\>CLS	Clears the screen and displays C: Prompt at the top of the screen.
DATE	DATE	Displays the system's current date and prompts to enter the new date.	C:\>DATE	Current date is Fri 05-09-2003 Enter new date <mm-dd-yy>:
TIME	TIME	Displays the current time and prompts the user to enter the new time.	C:\>TIME	Current time is 12:55:25.34p Enter new time.
VER	VER	Displays the Windows version.	C:\>VER	Displays the Windows version installed on your computer.
HELP	HELP <Command Name> or Command Name/?	Provides complete, information about queried MS-DOS commands.	C:\>HELP/?	Help is used to access the information and help file from MS DOS prompt.
DOSKEY	DOSKEY	Edits lines of command, recalls commands of MS-DOS, and creates macros.	C:\>DOSKEY	Once the Doskey is installed then Up and Down arrow keys can be used in the subsequent commands to recall the previous commands.
PROMPT	PROMPT [Text]	Changes the MS-DOS command prompt to the specified text. If the command is typed without any parameters then the default prompt	D:\>PROMPT	Changes the prompt to the default setting.
PRINT	PRINT <Filename>	Prints a text file while other MS-DOS commands are being used.	C:\DATA>PRINT TEMP.TXT	Prints 'TEMP.TXT' stored in the 'DATA' folder of the C: drive.
LABEL	LABEL	Makes, changes, or deletes the label of volume of a disk.	C:\>LABEL	Displays the current volume label and volume serial number. Also prompts to enter a new label.
MEM	MEM	Displays the amount of and free and used memory in your system.	C:\>MEM	Displays the total amount of memory, amount of used and free memory in the system.
MORE	MORE <Filename>	Displays output on the screen at a time for the text files. Useful in cases where the content of text file does not fit in a single screen.	C:\DATA> MORE TEMP.TXT	Breaks the contents of 'TEMP.TXT' in multiple screens. Subsequent screens can be viewed by pressing the 'ENTER' key.
ECHO	ECHO	Displays messages, or turns on or off the echoing command.	C:\>ECHO	Displays the current echo setting. ('OFF' or 'ON').
EDIT	EDIT	Starts MS-DOS editor, which produces and changes ASCII files.	C:\>EDIT	Opens the MS-DOS editor.
SET	SET	Displays, sets, or removes MS-DOS environment variables.	C:\>SET	Displays the settings for the current environment variables.
CHKDSK	CHKDSK <Drive-Name>	Checks a disk and gives the information like how many bytes have been used and how many are free and if any bad sectors are there on the disk.	C:\>CHKDSK A:	Checks A: drive and gives information about the disk.
SCANDSK	SCANDISK <Drive-Name>	Finds errors from a drive and fixes any problem it encounters.	SCANDISK A:	Scans the A: drive and repairs the disk if any problem is there, like damaged area or virus, etc.
DISKCOPY	DISKCOPY	Copies the content of one floppy disk to another.	DISKCOPY A:	Copies the entire content of one floppy to another.
FORMAT	FORMAT <Drive-Name>	Formats the specified drive.	C:\>Format A:	Floppy inserted in the A: drive will be formatted. Any information in the floppy A: will be erased.

NOTES

Simple File Operations

NOTES

Command	Syntax	Explanation	Example	Notes
MOVE	MOVE <Source> <Destination>	Moves the file from one source to the specified destination. File will exist only at the specified destination.	C:\DATA>MOVE TEST.TXT LETTER	Moves the file 'TEXT.TXT' from 'DATA' folder in the C: drive to the 'LETTER' folder in the C: drive.
COPY	COPY <Source> <Destination>	Creates a copy of the specified file and places it in the specified location, file will exist at the specified location as well as the source location.	C:\DATA> COPY HELLO.TXT LETTER	Creates a copy of HELLO.TXT in the LETTER folder of the C: drive.
REN	REN <Path> <Oldfile> <Newfile>	Renames the old file name by the specified new file name.	C:\DATA>REN HELLO.TXT Hi.TXT	Renames 'HELLO.TXT' as 'Hi.TXT'
DEL/ERASE	DEL <Path><Filename>	Deletes the specified file present in the Specified path/location from the hard disk.	C:\DATA>DEL Hi.TXT	Deletes the file 'Hi.TXT' located in the 'DATA' folder of the C: drive.
TYPE	TYPE <Filename>	Displays the contents of a text file.	C:\DATA>TYPE TMP.TXT	Displays the contents of TMP.TXT.
ATTRIB	ATTRIB [+A -A] [+R -R][+H -H] [+S -S] <filename> + Sets an attribute + Clears an attribute A-Archive attribute R-Read only attribute H-Hiddenfile attribute S-System file attribute	Displays or changes file attributes.	C:\>ATTRIB+H +R FIRST.TXT	Sets the attributes of 'FIRST.TXT' as Read only and hidden.
FC	FC <File1> <File2>	Compares two files or sets of files, and displays the dissimilarities between them. Generally used to compare files with same names located in different directories.	C:\FC C:\DATA\ ONE.TXT C:\INFO\ NE.TXT	Compares the file 'ONE.TXT' located in DATA folder with 'ONE.TXT' located in INFO folder and displays the differences between them.
XCOPY	XCOPY <Source> <destination>	Copies files and subdirectories to the specified location.	C:\>XCOPY C:\DATA C:\INFO	Copies the entire contents of the 'DATA' folder to 'INFO' folder. If the 'DATA' folder contains 'ami' subdirectories then they will also be copied to the 'INFO' folder.
BACKUP	BACKUP <Source> <Destination>	Copies the files onto diskettes or to other source.	C:\>BACKUP C:\DATA\TEMP. TXT C:\INFO	Copies 'TEMP.TXT' file in the 'DATA' folder into the 'INFO' folder.
RESTORE	RESTORE<Source> <Destination>	Reinstates files that were backed up by using the BACK UP command.	C:\>RESTORE C:\DATA\TEMP. TXT C:\INFO	Restores 'TEMP.TXT' file in the 'DATA' folder into the 'INFO' folder.

Wild Cards

Wild card characters can be used in specifying filenames in DOS. There are two types of wild cards (?,*):

? : It is used to represent any single character in the file name.

SYNTAX: C:\ DIR BA?.TXT

Displays all the text files in the C: drive starting with 'BA' and ending with any single character.

Examples: BAT.TXT, BAG.TXT, BAR.TXT, BAD.TXT, etc.

* : It is used to represent one or more characters in a file name.

SYNTAX: C:\ DIR CON*.TXT

Displays all the text files in the C: drive starting with 'CON'.

Examples: CONCEPT.TXT, CONCATENATE.TXT, CONTEMPT.TXT, CONSOLE.TXT, etc.

Path Command

MS DOS uses PATH= statement which resides in the autoexec.bat file. PATH command is used to display or set a search path for executable files.

```
PATH [[drive:]path[;...]]
```

```
PATH ;
```

Concept of File

A file is created with the help of another application program. A user, for instance, may create a text document in Notepad. So the application program in this case in Notepad and the file is a Notepad file.

It is a portion of a software program that is used to store data, information, settings, and/or commands used with that program. Examples of files are Word document files, Excel files, PowerPoint presentation files, database files, and so on.

Apart from the normal application program files such as MS Word, MS Excel, MS PPT, a computer system includes other file types also. These are as follows:

- **Batch File:** A batch file allows the users of MS Windows and MS DOS to generate a list of commands for running in sequence after the execution of the batch file. A batch file can be used for running the commands that are frequently run, deleting files, moving files, etc. No special programming skills are required for a simple batch file and can be prepared by users who have a basic knowledge of MS DOS commands.

An instance of a batch file is a file that is somewhat similar to an icon on the Mac OS or a shortcut in Windows. Similar to a shortcut, batch files can be used for running one or more commands and/or programs through the command line. Another popular instance of a batch file is a simple boot file called autoexec.bat, which is loaded every time the OS is loaded. It works on computers with early Windows. This batch file had all the required commands and programs used for running MS DOS and Windows every time the computer started.

- **Executable File:** It is a file that performs different functions or operations on a computer. Since an executable file is compiled, unlike a data file, it is generally not readable. On an IBM-compatible computer, the common executable files are .COM, .BAT, .BIN and .EXE. Other types of executable files can also be there depending on the operating system and its setup.
- **System File:** A file that is being used by an operating system and cannot be deleted or changed without the stopping of the operating system is known as a system file. Since these files are in use by the operating system, generally they cannot be deleted.

A system file is also an attribute that can be added to any file in Microsoft operating systems that allows the OS to know the file is an important system file. Files that are marked as system files will also be hidden files.

NOTES

Naming Files

A filename is a special type of string which is used for identifying a file stored on the file system of a computer.

NOTES

A filename is divided in two parts: basename (the primary filename) and the extension (which indicates the type of file related to a certain format of file).

A filename includes one or more of the following components:

- Protocol (or scheme) — Method of accessing (e.g., ftp, http, file, etc.)
- Host (or network ID) — Host name, IP address, domain name, or the network name of LAN (e.g., wikipedia.org, 207.142.131.206, \\MYCOMPUTER, SYS:, etc.)
- Device (or node) — Port, socket, drive, root mountpoint, disc, volume (e.g., C:, /, SYSLIB, etc.)
- Directory (or path) — Directory tree (e.g., /usr/bin, \TEMP, [USR.LIB.SRC], etc.)
- File – Base name of the file
- Type (format or extension) — Indicates the nature / type of the content of the file (e.g., .txt, .exe, .dir, etc.)
- Version — Revision number of the file

An extension name of a computer file is usually a three- character addition after the file's name. This extension assists IBM-compatible computers, such as those running Microsoft Windows, in identifying the program to relate the file with and correctly open the file.

Given below are the characters that cannot be used in filenames or directory names in most operating systems.

\, /, :, *, ?, ", <, > and |

Creating Directories

A hierarchical file system in the operating system, such as UNIX, Linux, DOS, OS/2, etc., contain directories. To refer to a directory, a user usually indicates it by name. Some of the important directories are: root directory, home directory and current directory.

Therefore, in Windows, there is no concept of a directory unless we create a folder and then create sub-folders and sub-sub folders within it.

The following is an example of a directory path in MS DOS.

C:\Windows\System32>

In the above example, C: denotes the drive, and System32 – which is in the directory of the Windows – is the present directory one would see.

The following is an example of a directory in a Linux/UNIX variant.

/usr/bin

In the above example, you are located in the bin directory which is a subdirectory of the usr directory.

For changing a directory in Linux, MS DOS, UNIX and other command line operating systems, the 'cd' command is used.

Given below are the special characters that cannot be used file or directory names in most operating systems:

\, /, :, *, ?, ", <, > and |

The AUTOEXEC.BAT File

AUTOEXEC.BAT is a system file found originally on DOS type operating systems. It is a plain text batch file that is located in the root directory of the boot device. The following is a basic DOS AUTOEXEC.BAT configuration consisting only of essential commands:

```
@echo off
prompt $P$G
PATH=C:\DOS;C:\WINDOWS
set TEMP=C:\TEMP
set BLASTER=A220 I7 D1 T2
lh smartdrv.exe
lh doskey
lh mouse.com /Y
win
```

The CONFIG.SYS File

The CONFIG.SYS file is found in the root directory of the C: drive or other boot drive of every PC. The CONFIG.SYS tells DOS which peripherals and devices are installed on your computer. Following statement is set as default values in CONFIG.SYS file:

```
DOS=HIGH
HIMEM.SYS
SETVER.EXE
FILES=60
BUFFERS=30
LASTDRIVE=Z
STACKS=9,256
FCBS=4
```

NOTES

1.7 INTRODUCTION TO WINDOWS

Windows XP is a line of proprietary operating systems which was developed by Microsoft and is meant to be used for general-purpose computers, such as home computers and business desktops, notebook computers and various types of media centres. Windows XP succeeds Windows 2000 and Windows ME, and is the first consumer-oriented operating system produced by Microsoft to be built on the Windows NT kernel and architecture. The most popular operating systems versions are Windows XP Home Edition which is primarily meant for home users, and

NOTES

Windows XP Professional, which boasts additional features, such as support for Windows Server domains and dual processors, and is meant for professionals and other experts. Windows XP Media Center Edition has additional multimedia features. Windows XP has an edge over the earlier versions of Microsoft Windows because of enhanced efficiency and better stability.

Windows XP helps you access and manage your files on the PC using a Graphical-User Interface GUI. All Programs and files stored on the PC are represented as pictures are called icons. These icons are stored on the desktop.

Components of Windows XP

The Windows XP user interface consists of various components and concepts that help make Windows XP user-friendly and intuitive. We will discuss these components and their use in Windows XP in this unit. Some of the components that we will discuss are as follows:

- Desktop
- Start menu
- Taskbar
- Icons
- Recycle Bin
- Windows
- Applications
- Folders
- Files
- Control Panel

We will explain what these components are and how to make use of them to make the most of your operating system.

Features of Windows XP

Following are the features of Windows XP:

1. All PCs give you the liberty to personalize your system according to your liking. XP makes it easy to personalize your computer by adding color, patterns, pictures, and even sound to improve your screen's appearance.
2. The User Accounts feature in Windows XP enables storing personalized settings and preferences by numerous users. Every time a new user logs in and request data, the computer retrieves it as though it has the information related to that user alone. User Accounts also help by securing your PC.
3. A Web browser called Internet Explorer comes as part of Windows XP. It gives you access to a rich knowledge base by accessing the Web.
Another interesting feature of Microsoft Windows XP is the Internet Connection Sharing (ICS). The computer using ICS, called the ICS host, can share its Internet connection with the other computers on the network.
4. Outlook Express is another widely used application in Microsoft Windows XP. With Outlook Express, you can send and receive e-mail messages. It also lets you to manage these messages by organizing them into different folders.
5. Windows Messenger provides the facility of instant text, chat voice, and video communication with friends and family all around the world.

6. Windows XP's Security Zone feature lets you to assign security options for Website access. It restricts you from exploring different sites by selecting the desired security level and asks for your approval before opening a file or running a program from the Internet.
7. With Microsoft Windows XP, you can download music from the Internet, arrange tunes in the order of your preference and compile them into a CD of your own. You can also store music on a portable MP3 player to enjoy your favourite tunes while you are away from your computer.
8. With its new Photo Album feature you need not worry about getting your photographs developed in the photo studio. You can categorize your photographs and keep them in a digital album on Windows XP. You can even edit these pictures to make them look more exciting and interesting. Moreover, Windows XP has some additional features which help you to view photos, as well as print, store and share them.
9. It also enables to play games on separate computers or with each other, as well as on the Internet. With its Internet gaming feature users can play multiplayer games with networked computers.
10. Windows XP has the option of linking computers to create a network, thereby sharing expensive resources, peripherals and even Internet connections. You can share scanners, printers, hard disks, files, folders and play multicomputer games.

NOTES

Working with Windows

Whenever you open a program, file or folder, it appears on your screen in a box or frame called a *window*.

Parts of a Window

Most windows have the following basic parts:

- **Title Bar:** It displays the name of the document and program or the folder name, if you are working in a folder.
- **Minimize, Maximize and Close Buttons.** These buttons hide the window, enlarge it to fill the whole screen and close it, respectively.
- **Menu Bar:** It contains items that you can click to make choices in a program.
- **Scroll Bar:** It lets you to scroll the contents of the window to see information that is currently out of view.
- **Borders and Corners:** You can drag these with your mouse pointer to change the size of the window.

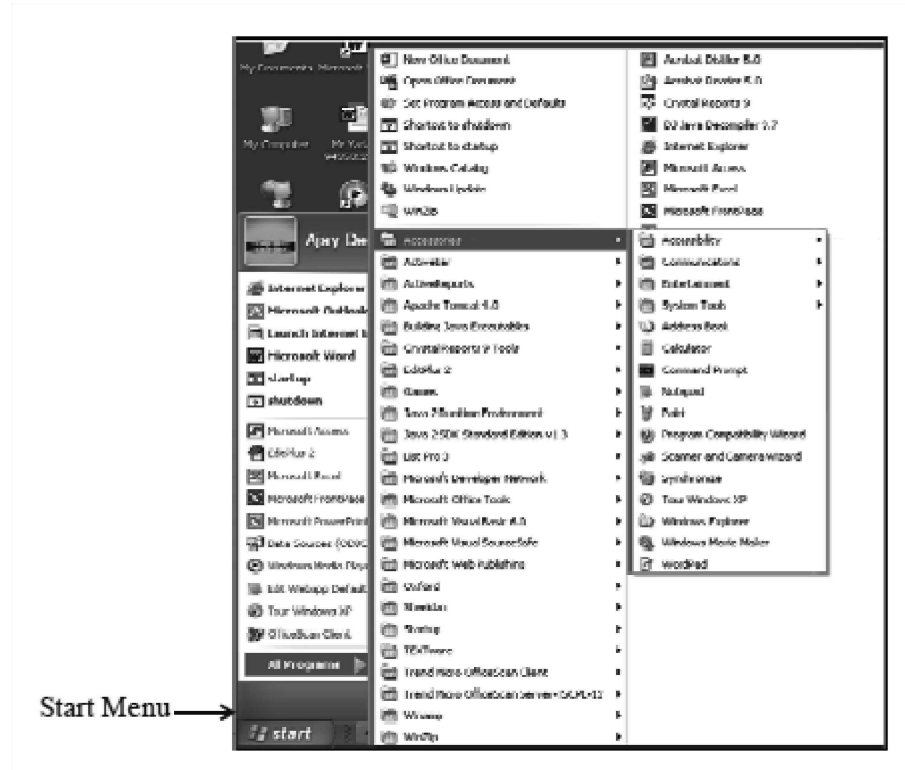
Start Menu and the Taskbar

Menus are very useful parts of the Windows system. They consist of various commands and subcommands for default and other applications and programs stored on your computer.

NOTES

Start Menu

Windows XP also has a new **Start menu** for enabling faster access to commonly used programs and common system areas, such as My Computer, Control Panel and Search.



The Start menu and the Start button are user interface elements in Microsoft Windows, and serve as the central launching point for all applications. By default, the Start button is always visible in the lower left-hand corner of the screen. It exhibits the Windows logo and the word ‘start’.

For choosing a program from the Start menu click on the Start menu option on the Taskbar. Once the Start Menu is expanded you can drag the mouse to the right and switch over to the submenus.

Switching between Programs

When you have more than one program running you can switch between these to make any one of them current. You can switch between two programs by simply clicking on the desired program button on the **Taskbar** shown below. You will notice that the active program window button will be darker in color.



Alternatively, you can perform the same task by using a combination of keys on your keyboard, by pressing the **Tab** key while holding the **Alt** key. On pressing this combination you will see a box around the icons for the running

programs. Each time you press the **Tab** key, the box moves to the next program. As soon as you release the **Tab** key the highlighted icon window will open. In case of the figures shown the window for 'My Computer' will open.

Quitting Windows

It is always a good idea to shut down all applications before switching off your computer. Directly switching off the power can corrupt your data.

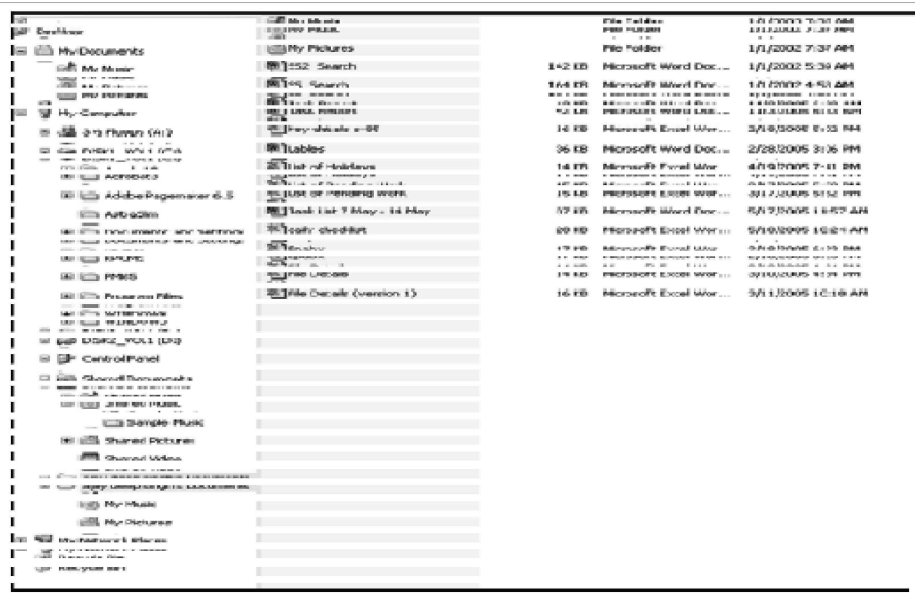
NOTES



In order to shut down your PC go to the Start menu on the Taskbar and select the **Turn off computer** option. On doing so a screen (below, right) is displayed. This gives you options to **Log Off**, **Restart** or **Turn Off** the PC. Select the Turn Off option for it to close all applications before powering off.

Windows Explorer — Managing Files and Folders

Windows is responsible for maintaining all your files and subfolders on your disk. You are aware by now that you store information in individual files. Groups of files on a common subject can be kept in a specially earmarked place called a folder.



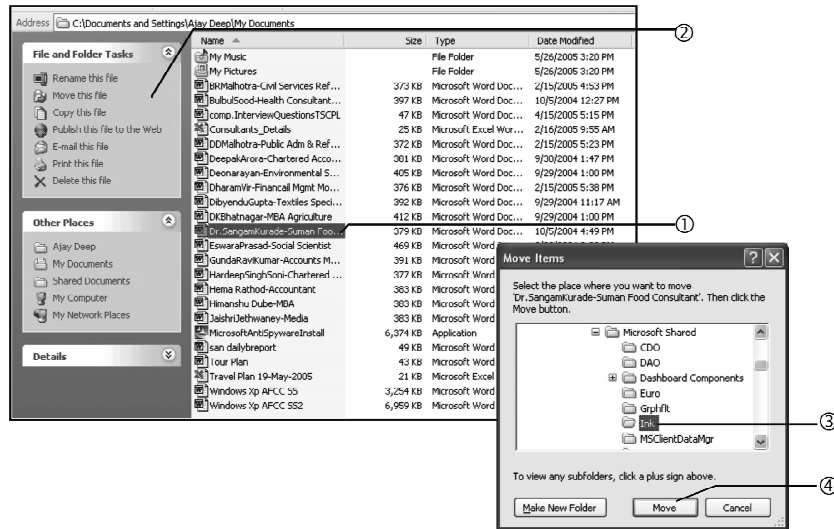
Starting Windows Explorer

Click on the **Windows Explorer** command from **All Programs** option of the **Start** menu. (Some versions of Windows call it **Programs** instead of **All Programs**.)

NOTES

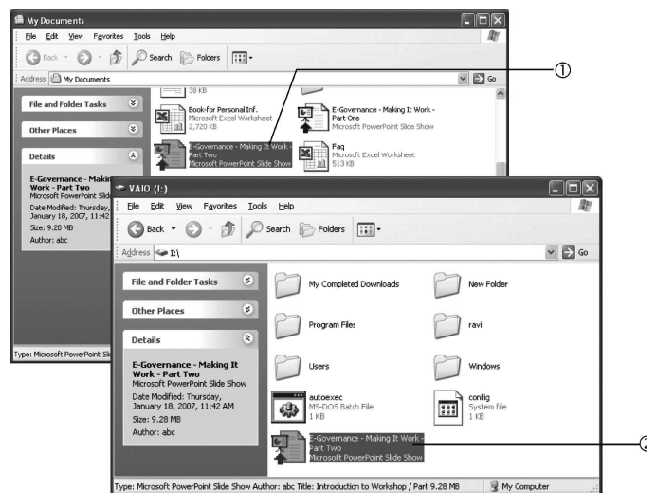
Moving Files

- 1 Click on the file which you wish to move.
- 2 Click on the **Move this file** icon from the **Standard buttons** toolbar. The **Move Items** dialog box will be displayed.
- 3 Click on the location/folder you wish to place the file. In our example, we have taken **Ink** as the folder we want to move the file to.
- 4 Click on the **Move** button to move the file.



Copying Files by Click and Drag Method

- 1 Click on the file that you want to copy to another folder.



- 2 Keeping the left mouse button pressed on the highlighted file, drag it to the folder which you want to copy. Make sure that the folder you want to drag

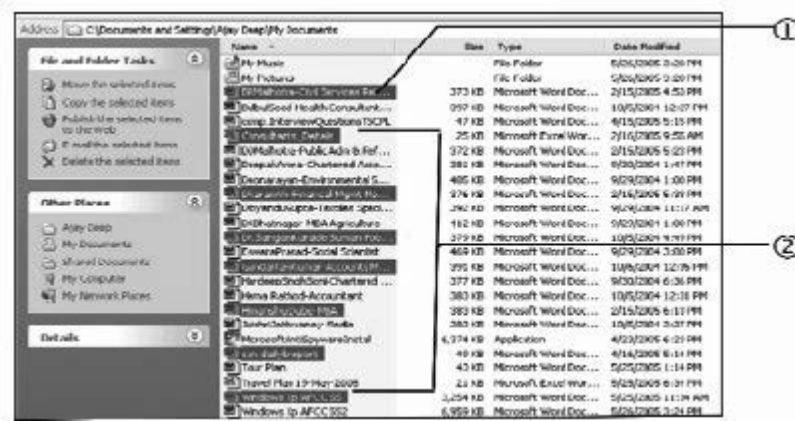
the file to is visible. Once you have dragged the selected file to the desired folder, you can release the mouse button.

You will now see the file displayed in the required folder. Note that when you drag a file to a folder on the same disk, it will be moved, but if you drag it to a folder on another disk, it will be copied.

NOTES

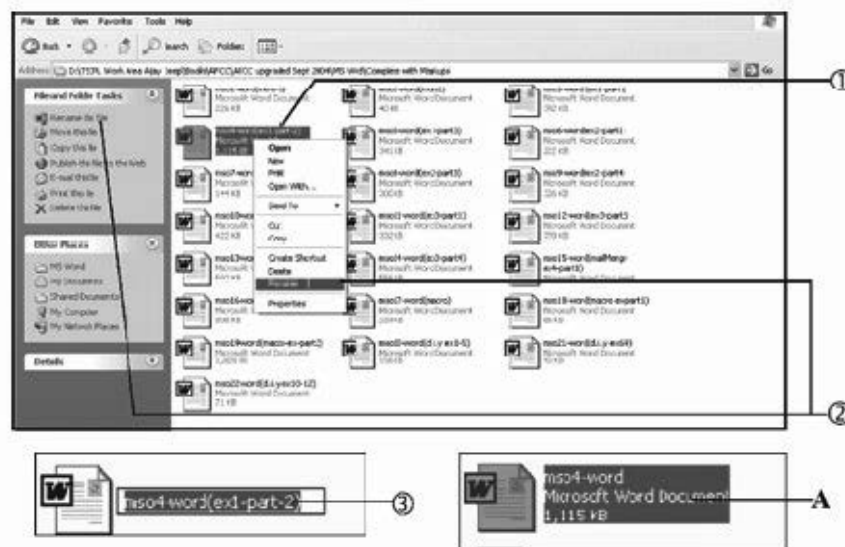
Copying Multiple Files

- ① Click on a file that you want to copy.
- ② Press the **Ctrl** key and while keeping it pressed, click on another file. Repeat the same process for as many files as you wish to select. Now, follow the steps that you take to copy a file to another folder. This option of copying files is easier in the sense that you do not have to copy the required files one by one to another folder.



Renaming a File

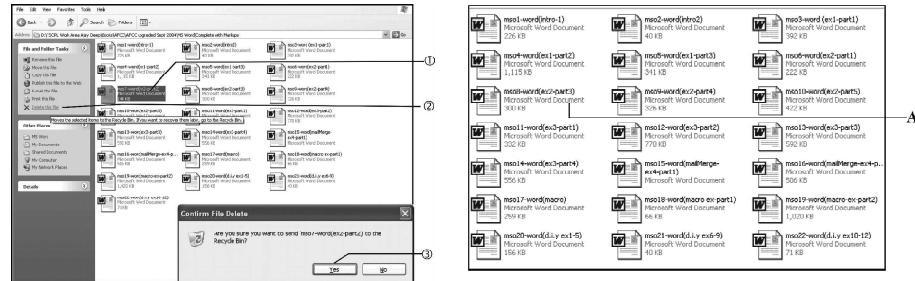
- ① Highlight the file you want to rename.
 - ② Click on the **Rename this file** option from the left panel or press the right mouse button once and select the **Rename** option from the pop-up menu.
 - ③ You can type in the new name for the file here.
- A The file name has been changed.



NOTES

Deleting Files

- ① Highlight the file you want to delete.
- ② Click on the **Delete this file** option from the left panel or press the **Delete** key from your keyboard.
- ③ Click on the **Yes** button once.

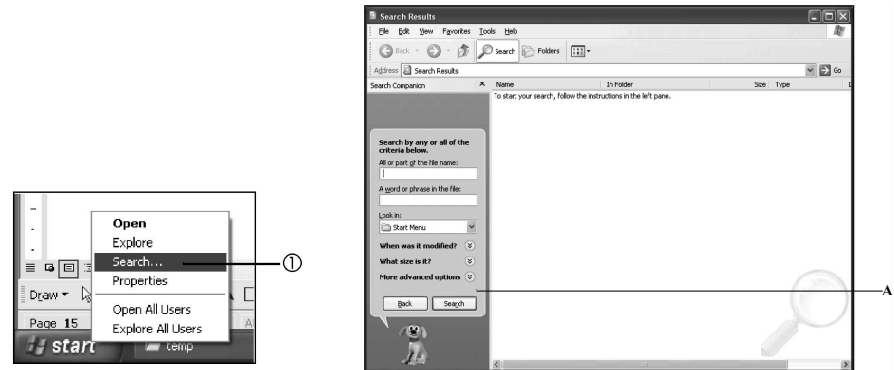


A You will see that the desired file has been deleted.

Searching a File or Folder

- ① Place your mouse pointer on the **Start** button. Press the right mouse button once and select the **Search** option.

A You can search for the files or folders as desired based on the criteria you select.

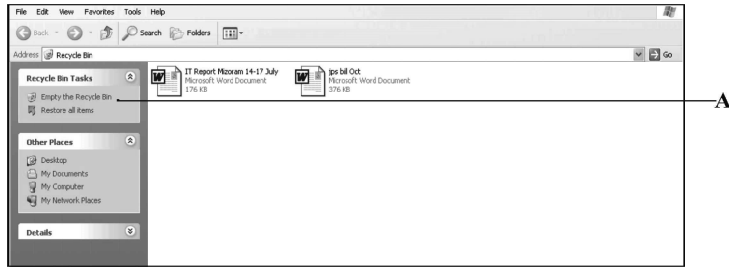


The Recycle Bin

- ① Click on the **Recycle Bin** icon on the Desktop.

A You can select from the options available. These include deleting the files from the Recycle Bin permanently or restoring it to the position where they had been deleted from.





NOTES

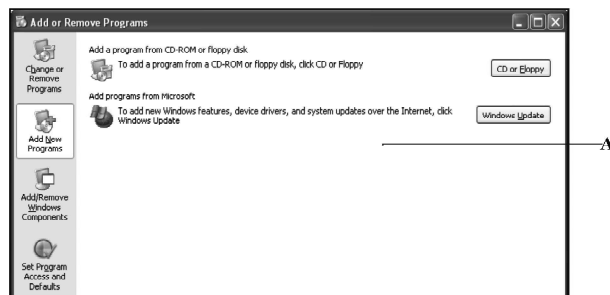
The Control Panel

Adding or Removing programs (Adding All Programs and changing Windows Components).

- ① Click on the **Start** menu and go to **Control Panel**.
- ② Select the **Add or Remove Programs** option by double-clicking on it.



A The following screen will be displayed.



You will see icons in the left panel of the screen that say **Change or Remove Programs**, **Add New Programs**, and **Add/Remove Windows Components**.

Change or Remove Programs

NOTES

This option allows you to change or remove existing installed programs. To remove a program

- ① Select the program by clicking on it.
- ② Click once the **Remove** button shown against the program



Add New Programs

To install and/or uninstall additional software programs click on the **Add New Programs** buttons.

A The following window would be displayed. By clicking once on the **CD or Floppy** button you can install a new program from either a CD or a floppy.

B You can choose **Windows Update** to add new Windows features or system updates over the Internet.



Date and Time

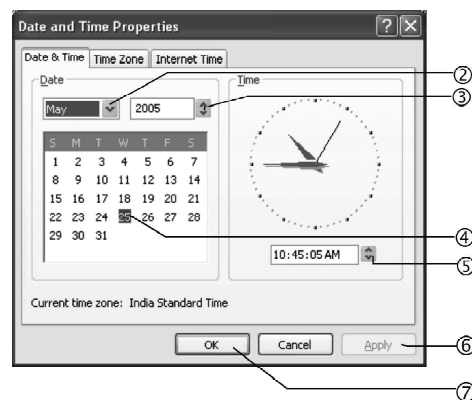
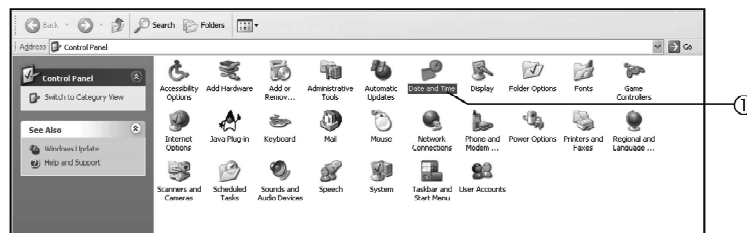
There is a small component inside your computer called Real Time Clock (RTC). This RTC and CMOS setup are powered by a battery cell to ensure that the basic settings of your computer are not lost when you switch off your system. RTC stores the current date and time in your computer and passes on this information to put a date and time stamp on all file operations. The date and time option gives you the liberty of adjusting the system's date and time to correct settings. It is important that your computer works on the correct clock and calendar timings, so that it uses current date and time information to create and update your files.

Windows allows you to set the date, time, and time zone for your computer using the **Date and Time** option of the **Control Panel**.

NOTES

Changing the Date and Time

- 1 Click once on the **Date and Time** icon.
- 2 Click on the month drop-down list to change the month.
- 3 Click on this button to change the year.
- 4 Click on any of the dates to change the date.
- 5 Click on this draw to change the time.
- 6 Click on **Apply** button to apply the changes you have made.
- 7 Click on **OK** button to continue.



Setting Time Zone

Following steps are required to set time zone in Windows XP system:

- Open Date and Time in Control Panel.
- On the **Date & Time** tab, select the item you want to change.

NOTES

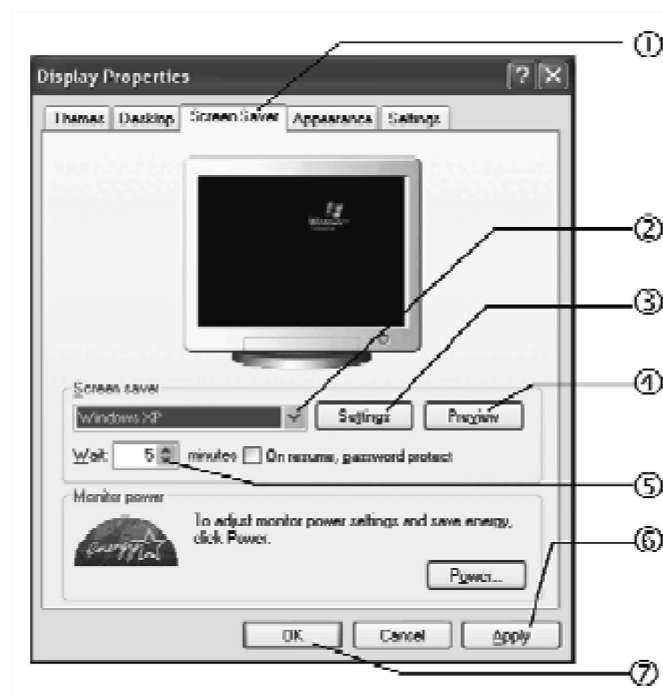
- To change the hour, double click the hour and then click the arrows to increase or decrease the value. To change the minutes, double-click the minutes and then click the arrows to increase or decrease the value. To change the seconds, double-click the seconds and then click the arrows to increase or decrease the value. To change the AM/PM indicator, select it and then click the arrows.
- To change your time zone, click the Time Zone tab and then click the drop-down arrow to set the current time zone.

Screen Saver

A screen saver is a moving image or pattern that appears on your monitor when you have left your computer idle for a specified time.

This makes sure that nobody peeps and messes with your computer files while you are away from your desk.

- ① Click on the **Screen Saver** tab.
- ② The drop-down list lets you choose the screen saver that you want to apply.
- ③ Click on the **Settings** button to change other settings.
- ④ You can preview the selected screen saver before applying it. To do so click on the **Preview** button once.
- ⑤ You can increase or decrease the time (in minutes) for activating the screen saver.
- ⑥ Click on **Apply** button for applying the changes made.
- ⑦ Click on **OK** button to continue.

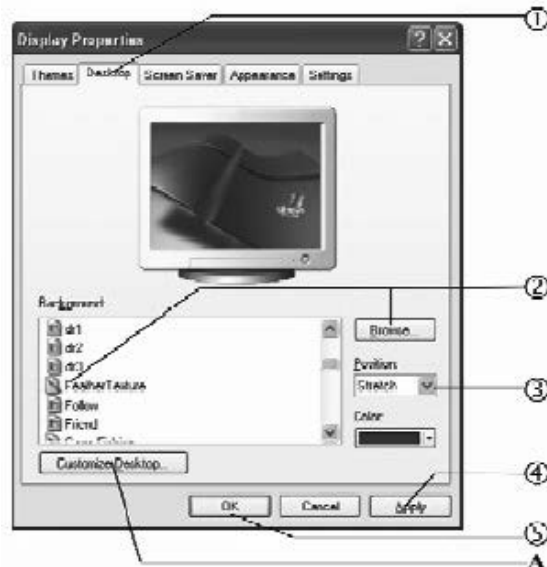


Wallpaper or Background

Desktop wallpapers do exactly what your room wallpaper does—decorate your surrounding and add more colour and fun to your life. Use this option to modify and apply different pictures and images as your desktop wallpaper. You can also use repetitive designs called Patterns to get a woven fabric or block printed look on your desktop.

- ① Click once on the **Desktop** option from the **Display Properties** pop-up screen.
- ② Click once on the background/wallpaper you desire from the list. Background lists the available wallpapers you can use to decorate your desktop. When you click on a wallpaper in this list, a preview of how the wallpaper will look will appear on the dialog box screen. Alternatively, you can click on the **Browse** tab to select a file from the desired location.
- ③ You can define the position of the wallpaper here, i.e., whether you want the wallpaper in the centre of the screen, tiled across your screen, or you want it stretched to fit the screen.
- ④ Click on the **Apply** button to apply the settings that you have changed.
- ⑤ Click on the **OK** button to continue.

A You can further customize your desktop. These include options like choosing the icons displayed on your desktop and icons used for representing a program. Click on the **Customize Desktop** button to see available options.



Mouse

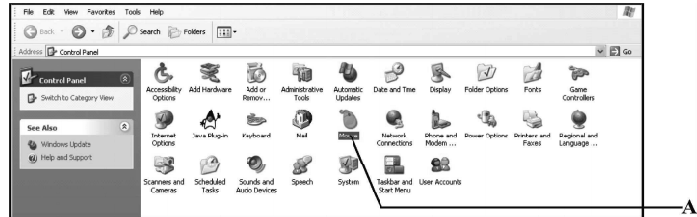
As explained earlier, a mouse is a small hand-held device through which you can quickly and easily navigate different parts of the screen. You can customize various aspects of the mouse to make it work the way you prefer. You can control the mouse pointer speed, shape, button configuration, double-click speed, etc. You

NOTES

can even swap the left and right buttons. If you frequently lose track of where your mouse pointer is, you can choose to have a mouse trail effect, which will make the mouse pointer more visible. To change any of the above-mentioned mouse settings:

NOTES

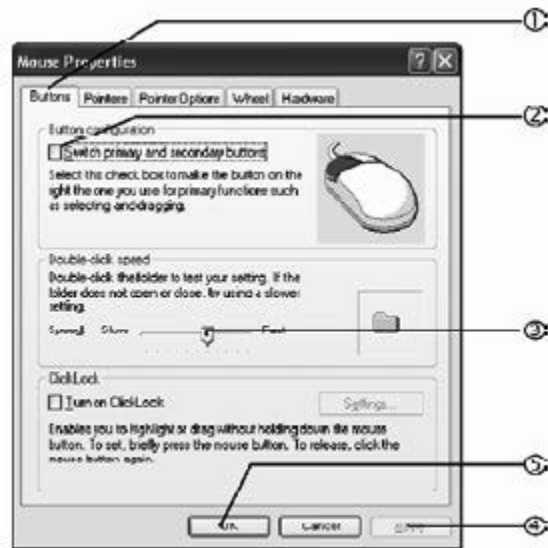
A Choose **Mouse** option from the **Control Panel**.



Button Configuration

To customize button configuration follow the steps given below:

- 1 Click on the **Buttons** tab.
- 2 Click on the check box under the **Button configuration** option to convert your left mouse button into right button and vice versa.
- 3 Drag this button to increase or decrease the clicking speed of the mouse.
- 4 Click on the **Apply** button to apply the changes that you have made.
- 5 Click on the **OK** button to continue.



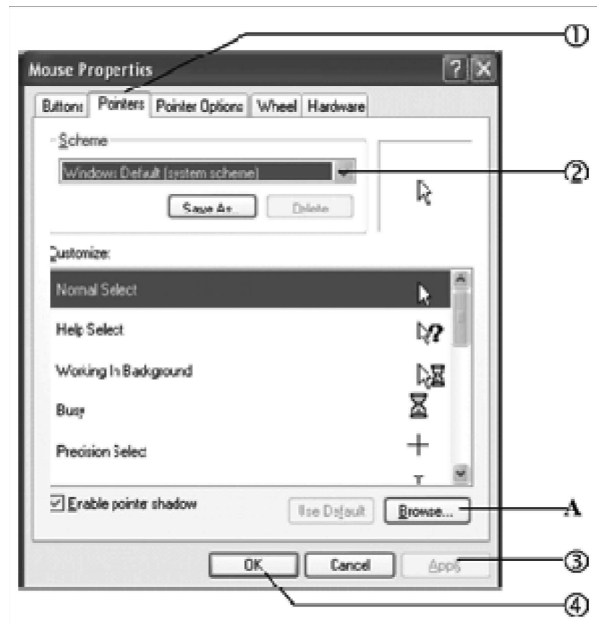
Pointer Configuration

To customize pointer configuration:

- 1 Click on **Pointers** tab.
- 2 Click on the pull-down list to choose the scheme that you want to apply.

- A You can choose a scheme from another location by clicking on the **Browse** button.
- ③ Click on **Apply** button to apply the chosen scheme.
- ④ Click on the **OK** button to continue.

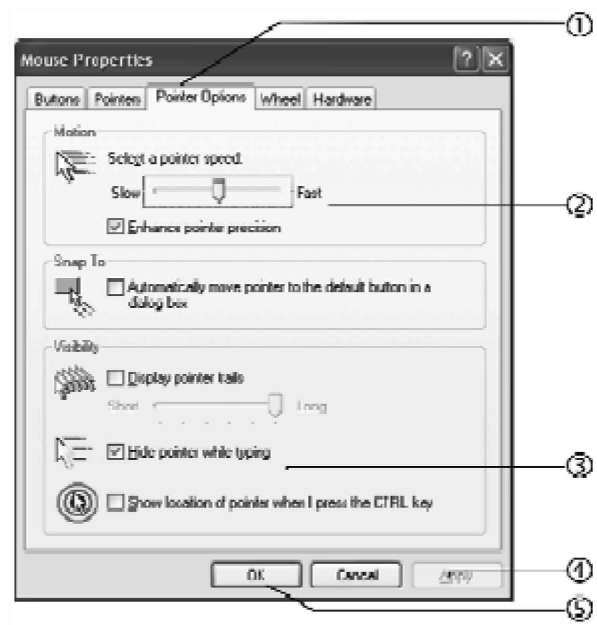
NOTES



Motion Configuration

You can also customize the motion configuration of the mouse. To do so:

- ① Click on the **Pointer Options** tab.
- ② Click on this button to increase or decrease the pointer speed.
- ③ You can also click on the **Hide pointer while typing** option to hide the pointer while you are typing.
- ④ Click on the **Apply** button to apply the changes you have made.
- ⑤ Click on the **OK** button to continue.



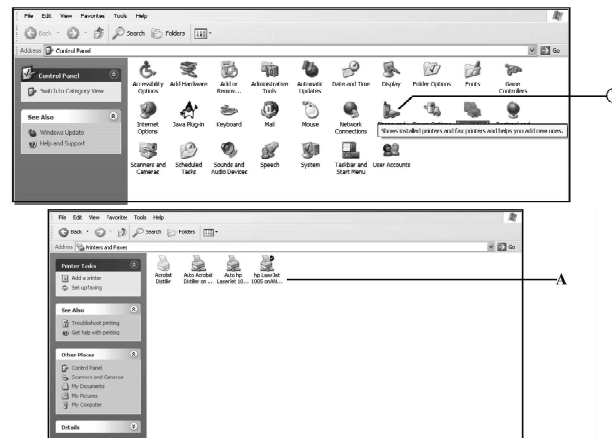
NOTES

Printers and Faxes

Use this option to install new printers. You would need the Windows XP CD or the Printer Installation CD to run this option. Windows XP installation CD includes many printer drivers from various well-known printer manufacturers. However, if your printer or its exact model is not listed in the driver's list provided by Windows XP, you need not worry — simply insert the CD or floppy (which must have come along with the printer) and complete the printer installation procedure.

① Choose the **Printers and Faxes** option from the **Control Panel** to see the printers installed.

A You will get a list of the printers installed on your machine.



Cameras and Scanners

Scanners and Cameras option in Control Panel enables you to install scanners, digital cameras, digital video cameras and other image-capturing devices. After a device is installed, you can use the **Scanners and Cameras** Wizard to download and save pictures on your computer in a folder you specify. You can also view device properties, delete pictures from your camera or print photos. You can even test your device to make sure everything is working properly. You can use **Scanners and Cameras** to link your device to a program on your computer. For example, you can set your computer to automatically open all of your scanned pictures in your program of choice. Windows automatically saves pictures to the **My Pictures** folder or to a subfolder you specify. If you save your files to any **My Pictures** subfolder or any folder customized as a pictures folder Windows provides specialized tools and features you can use **Windows Picture and Fax Viewer** and the ability to view your pictures as a slide show.

Sound and Audio Devices

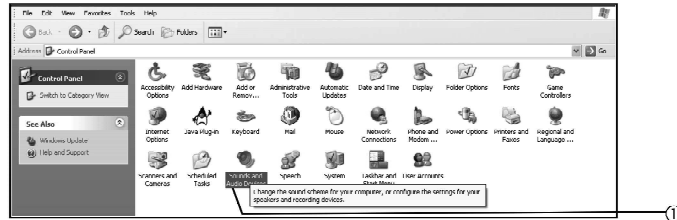
Multimedia, as the name suggests, is a combination of many mediums—sounds, static pictures, movie clips, animation, text, etc. The main components of Windows XP Multimedia are: Sound Recorder, CD player, Media Player and Active Movie Player. Although most new computers today have inbuilt multimedia facilities, multimedia is not a standard or necessary feature of all computers. You must have multimedia hardware like the CD Rom drive, sound card, speakers and microphone pre-installed before you can use multimedia features.

Sounds

You can use this option for assigning different sounds to various system events, such as error messages, warning dialog boxes, starting and exiting windows, etc. Needless to say, this option will work only if you have a sound card and speakers installed. To do so:

NOTES

- 1 Choose the **Sounds and Audio Devices** option from the **Control Panel**.



- 2 Choose the **Sounds** icon from the **Sound and Audio Devices Properties** pop-up menu.
- 3 Click on the **Sound scheme** drop-down list to choose a scheme. This list displays schemes that you can use to simultaneously change the sounds assigned to many system events.
- 4 Click on the **OK** button to apply the changes that you have made.



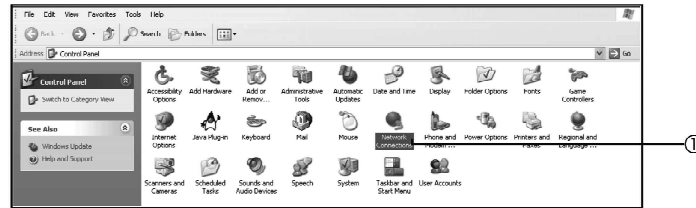
Network

In a computer network you can link several computers so that they can exchange information between themselves. Another advantage of a computer network is that expensive resources like plotters, scanners and printers can be shared between many computers, thereby eliminating the need for duplicating these resources for each computer.

The **Network Connections** window contains utilities like Network Setup, Mail, Schedule and Remote Access.

- 1 Choose the **Network Connections** command from the **Control Panel**.

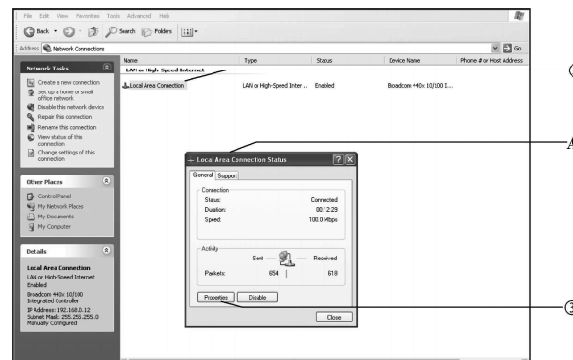
NOTES



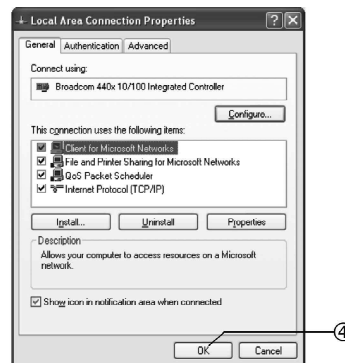
- 2 Double-click on the **Local Area Connection** icon.

A You will get this **Local Area Connection Status** pop-up window.

- 3 Choose **Properties** from the window.



- 4 You can modify any network setting you want and then click on **OK** once to apply the settings.



1.8 INTRODUCTION TO UNIX

In 1960, the Massachusetts Institute of Technology, American Telephone Telegraph or AT&T Bell Laboratories and General Electric worked on an operating system, MULTiplexed Information and Computing Service (MULTICS). The MULTICS operating system was designed to run the GE-645 mainframe computers. The operating system was not very successful. Ken Thompson, who was a developer in the AT&T Bell Laboratories, developed a game called Space Travel for the

GE-645 mainframe computers. The system was running at a very slow speed. Then, he wrote the game in the assembly language with the help of Dennis Ritchie.

Ken Thompson and Dennis Ritchie included Rudd Canady and formed a team of developers. The aim of the team was to develop an operating system with the support for file system and multitasking for the Program Decision Package or PDP-7 machines. The operating system included a command line interpreter and different utility programs.

In 1970, an operating system named UNiplexed Information and Computing System (UNICS) was developed, which allowed two users to work simultaneously. The name was changed to UNIX from UNICS. UNIX as an operating system was developed in the early 1970s at AT&T Bell Laboratories.

NOTES

UNIX as an Operating System

UNIX is a multi-user and multitasking operating system. In a multi-user environment, the computer can receive the commands from a number of end users to run programs, access files, and print documents simultaneously. Figure 1.36 shows a multi-user environment.

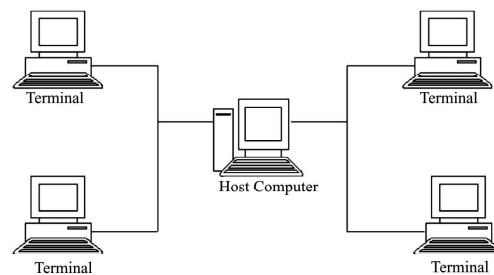


Fig. 1.36 Multi-user Environment

The host computer, which has a UNIX operating system, provides services to the terminals, such as file access services. Four terminals are connected to one host computer and all the terminals are sharing resources from the host computer.

Multitasking feature performs scheduling of work while one task is waiting for input and another task reads input from the hard disk. As a result, you can perform more than one task simultaneously.

Features of UNIX

UNIX as an operating system supports all features that are expected in an operating system. UNIX also provides additional features that are not supported in any other operating system. The general and additional features of UNIX operating system are as follows:

- **File and Process:** File and process are two entities that are supported by UNIX. A file contains information, such as text, code or directory structure that you need to save in the computer. The file is stored in the hard disk of the computer at a particular location which can be easily remembered whereas a process is the name given to a file or a program that is currently running. UNIX provides various tools that enable you to control a process, change the sequence of the process and kill the process.

NOTES

- **Multi-user System:** UNIX supports multitasking system as the kernel is designed to handle multiple processes. A single user can run multiple processes simultaneously. For example, an end user can print a file and edit another file simultaneously. The kernel handles the multiple processes as foreground and background processes. The current process runs in the foreground and the other processes run in the background. The foreground and background processes can be switched according to the requirements. This multitasking feature is an advantage for the programmers, as they do not have to close the editor and run the program; this can be done simultaneously.
- **UNIX Toolkit:** The UNIX toolkit provides various tools that enable you to perform different tasks in UNIX as kernel alone cannot perform every task. The tools that are included in the UNIX toolkit are as follows:
 - o General purpose tools, such as vi editor.
 - o Text manipulation utilizes filters that are used to retrieve the output from two or more commands simultaneously.
 - o Compiler and interpreter.
 - o Network administration and system tools, such as mailx and pine.
- **Pattern Matching:** UNIX supports pattern matching feature that enables you to retrieve the output according to the required pattern. Pattern matching in UNIX can be implemented using a special characters, such as * known as metacharacter. The * character denotes multiple characters. For example, if you need to retrieve all the files whose name start with A, you can give the argument for the list command as A*, instead of specifying all the names of the files.
- **Programming Facility:** UNIX provides a programming facility known as shell that is developed specifically for programmers and not for the users. The shell programming includes all programming features, such as variables, loops and control structures that help you to create a shell script.

1.9 DATA PROCESSING

Information is handled in the computer by electrical components such as transistors, integrated circuits, semiconductors and wires, all of which can indicate only two states or conditions. Transistors are either conducting or non-conducting; magnetic materials are either magnetised or non-magnetised in a direction; a pulse or voltage is either present or not present. All information can therefore be represented within the computer by the presence (ON) or absence (OFF) of these various signals.

Thus, all data to be stored and processed in computers are transformed or coded as strings of two symbols, one symbol to represent each state. The two symbols normally used are 0 and 1. These are known as BITS, an abbreviation for BInary digiTS.

Let us now understand some commonly used terms:

Value	Meaning
0	Off
1	On

Fig. 1.37 Bits

NOTES

BITS: The smallest element that a computer uses. It can hold one of two possible values.

A bit which is ON is seen as SET or TRUE; and a bit which is OFF is seen as NOT SET or FALSE.

Since only two values can be stored in a single bit, there can only be four unique combinations:

00 01 10 11

Bits therefore have to be combined together into larger units for holding greater ranges of values.

NIBBLE: A set of FOUR bits. There can only be a maximum of sixteen different values possible:

$$2^4 = 16 \text{ (2 to the power of the number of bits)}$$

BYTES: A set of two nibbles or eight bits. Characters are often stored in bytes along with numeric values.

$$2^8 = 256 \text{ (2 to the power of the number of bits)}$$

WORD: Just like we express information in words, so do computers. A computer 'word' is a group of bits, the length of which varies from machine to machine, but is normally pre-determined for each machine. The word may be as long as 64 bits or as short as 8 bits.

Character Representation

Binary data is not the only data handled by the computer. We also need to process alphanumeric data like alphabets (upper and lower case), digits (0 to 9) and special characters like + - * / () space or blank etc. These also must be internally represented as bits.

Decimal Number	Binary Equivalent
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001

Fig. 1.38 Data Representation in Bits

NOTES

BCD

Binary Coded Decimal (or BCD) is one of the early memory codes. It is based on the concept of converting each digit of a decimal number into its binary equivalent rather than converting the entire decimal value into a pure binary form. It further uses 4 digits to represent each of the digits. The table below shows the BCD equivalent of the decimal digits.

Converting 42_{10} into its BCD equivalent would result in:

$$42_{10} = \begin{array}{cc} 0100 & 0010 \\ 4 & 2 \end{array} \text{ or } 01000010 \text{ in BCD}$$

As seen, 4-bit BCD code can be used to represent decimal numbers only. Since 4 bits are insufficient to represent the various other characters used by the computer, instead of using only 4-bits (giving 16 possible combinations), computer designers commonly use 6 bits to represent characters in BCD code. In this the 4 BCD numeric place positions are retained, but two additional zone positions are added. With 6 bits it is possible to represent 2^6 or 64 different characters. This is therefore sufficient to represent the decimal digits (10), alphabetic characters (26), and special characters (28).

EBCDIC

A major drawback of the BCD code is that only sixty-four distinct characters can be represented by it. Since this proved insufficient for representing many special characters (28 plus), uppercase letters (26), lowercase letters (26) and decimal numbers (10), the BCD code was extended from a 6-bit to an 8-bit code. The additional two bits function as zone bits, thus expanding the zone bits to four. The code that results is known as the Extended Binary-Coded Decimal Interchange Code (EBCDIC). It was developed by IBM and is used in many computers, particularly the majority of IBM models. It allows 2^8 or 256 characters to be represented. This takes care of the character requirement along with a large quantity of non-printable and printable control characters (movement of the cursor on the screen, vertical spacing on printer etc.).

Since EBCDIC is an 8-bit code, it is possible to divide it into two groups of 4-bits each. One hexadecimal digit represents each group (explained earlier in this unit). Thus, computers using the EBCDIC code for internal representation of characters can use the hexadecimal number system as a notation for memory dump.

ASCII

A computer code that is very widely used for data interchange is called the 'American Standard Code for Information Interchange' or ASCII. Several computer manufacturers have adopted it as their computers' internal code. This code uses 7 digits to represent 128 characters. Now an advanced ASCII is used having 8-bit character representation code allowing for 256 different characters. This representation is being used in Micro Computers.

We will now study the encoding method. The following table presents the bit combinations that are required for each character.

Bit Combinations for Each Character

	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F
00	NUL	SOH	STX	ETX	EOT	ENQ	ACK	BEL	BS	TAB	LF	VT	FF	CR	SO	SI
10	DLE	DC1	DC2	DC3	DC4	NAK	SYN	ETB	CAN	EM	SUB	ESC	FS	GS	RS	US
20		!	"	#	\$	%	&	'	()	*	+	,	-	.	/
30	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
40	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
50	P	Q	R	S	T	U	V	W	X	Y	Z	[\]	^	_
60	'	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
70	p	q	r	s	t	u	v	w	x	y	z	{		}	~	DEL

NOTES

Thus, the text string ‘Hello.’ is coded in ASCII using hexadecimal digits as follows:

H	e	l	l	o	.
48	65	6C	6C	6F	2E

The byte sequence 48 65 6C 6C 6F 2E represents the string.

Processing of Data

Data processing can be defined as the process of converting raw data into suitable information using a series of operations like classifying, sorting, summarizing and tabulating it for easy storage and retrieval. Processed data is called information.

Data, especially large volumes of it, unless processed properly, are not of much use in the current information driven world. Relevant information can give a definite edge to a business to stay ahead of its competition and plan for the future. In fact, the speed at which information can be extracted from data (a process called data processing) is just as crucial as the information itself. Information usually loses its value if it’s not up-to-date. Automatic Data Processing (ADP) applications are gaining wide popularity in the market to solve this very problem. They not only save time but also reduce the cost of data processing. An ADP application, once configured, is ideal for converting similarly structured data into specific sets of information using predefined rules of selection, processing and presentation. Data processing can also include the conversion of one type of information into another for legacy systems transfer.

Typically, a data processing cycle can be broadly divided into five stages:

- Data Collection
- Data Preparation
- Data Input
- Data Processing
- Information Output

Just as there are different types of data (classified either by usage, attributes or content), there are different methods of processing data. These are:

1. *Real-time Processing*: In this mode, data is processed almost immediately (in real time) and in a continuous flow. This is of particular advantage when

NOTES

the lifespan of information is small and core business activities are involved. The advantages of real-time processing are that the derived information is up-to-date and so it is more relevant for decision making. For instance, in a bank or in an ATM (Automatic Teller Machine), as soon you deposit money in your account, your account status (balance standing to your credit) is updated instantaneously. This enables you as well as the bank to know the exact status of funds, in real time mode, or in other words, *as of this minute*. Similarly, in a railway reservation system, a train ticket booked from anywhere in the world MUST update the central database in real-time to ensure that the seat once booked is not sold to anybody else in the world. Real time processing also requires relatively little storage compared to batch processing.

2. **Batch Processing:** Real time processing requires high speed broadband connections so that the data inputted from different computers or locations can be used to update a centralized server and database. Setting up such networks is expensive and not always feasible because sometimes the data does not need to be processed immediately. For instance, in a BPO (Business Processing Outsourcing) outfit, hundreds of operators may be inputting data, which can be made available to the client only after it is checked and verified by a supervisor(s). Such situations call for batch mode processing, which is used when the conversion of data into information is not required to be done immediately and therefore this data processing is done in lots or batches. The advantages of batch processing are that it is cheaper and processing can be done offline.

It should be noted that data processing and data conversion are technically quite different; while data conversion only means converting data from one form to another, data processing means conversion of data into information or sometimes vice versa.

CPU

Execution of programs is the main function of the computer. The program to be executed is a set of instructions that is stored in the computer's memory. Tasks are completed when the instructions of the program are executed by the Central Processing Unit (CPU). Also, all the major calculations and comparisons are carried out inside the CPU. Additionally, the CPU is responsible for activating and controlling the operations of various units of the computer system. It activates the peripherals to perform input or output.

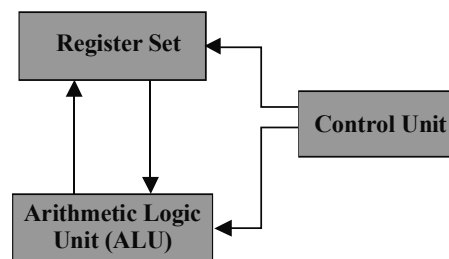


Fig. 1.39 Major Components of a CPU

The CPU is made up of three major components (as seen in Figure 1.39): the register set (associated with the main memory) that stores the intermediate data during the execution of instructions, the Arithmetic Logic Unit (ALU) that performs the required micro-operations for executing the instructions, and the control unit that supervises the transfer of information among the registers and instructs the ALU as to which operation to perform.

NOTES

Control unit

The control unit is necessary if the CPU is to function efficiently and information/data is to be transferred between the CPU and other devices. It does not perform the actual processing of the data, but manages and coordinates the entire computer system, including the input and, output devices. It retrieves and interprets the instructions from the program stored in the main memory, and issues signals that cause the other units of the system to execute them.

It does this through some special purpose registers and a decoder. The special purpose register called the **Instruction register** holds the current instruction to be executed, and the **Program control register** holds the next instruction to be executed. The decoder interprets the meaning of each instruction supported by the CPU. Each instruction is also accompanied by a **Microcode**, i.e., the basic directions to tell the CPU how to execute the instruction.

Arithmetic logic unit

The arithmetic logic unit or ALU provides arithmetic and logic operations. This means that when the control unit encounters an instruction that involves an arithmetic operation (add, subtract, multiply, divide) or a logic operation (equal to, less than, greater than), it passes control to the ALU. The ALU has the necessary circuitry to carry out these arithmetic and logic operations.

As an example, a comparison of two numbers (a logical operation) may require the control unit to load the two numbers in the requisite registers and then pass on the execution of the 'compare' function to the ALU.

Figure 1.40 represents the basic structure of a CPU.

NOTES

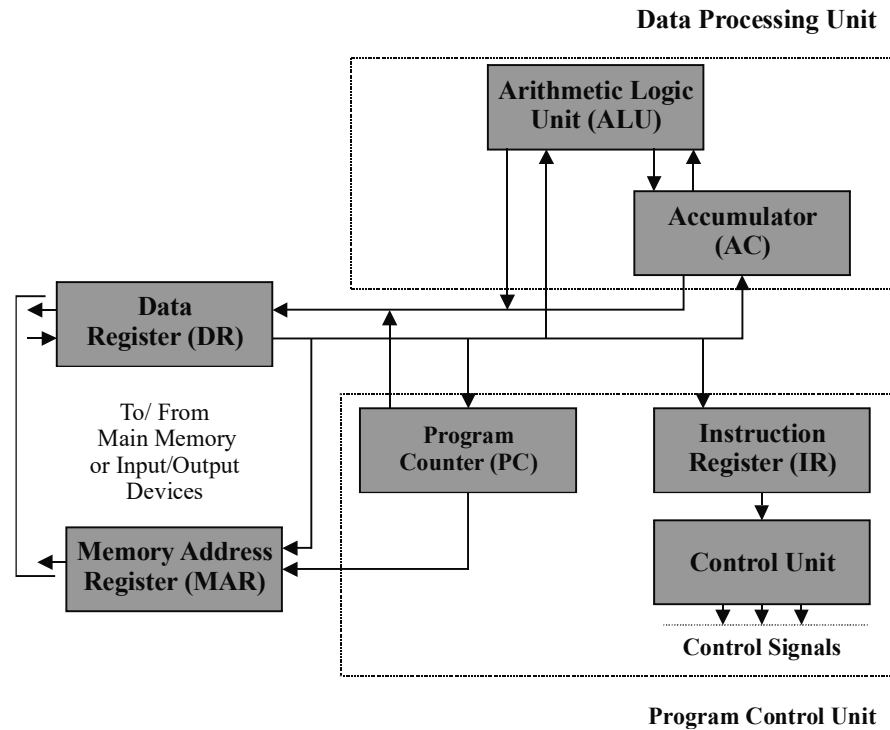


Fig. 1.40 Basic Structure of a CPU

Instruction set

The primary function of the processing unit in the computer is to interpret the instructions given in a program and carry out the instructions. Processors are designed to interpret a specified number of instruction codes. Each instruction code is a string of binary digits. All processors have input/output instructions, arithmetic instructions, logic instructions, branch instructions and instructions to manipulate characters. The number and type of instructions differ from processor to processor. The list of specific instructions supported by the CPU is termed as its **Instruction set**.

An instruction in the computer should specify the following:

- The task or operation to be carried out by the processor. This is termed as the **opcode**.
- The address(s) in memory of the operand(s) on which the data processing is to be performed.
- The address in the memory that may store the results of the data processing operation performed by the instruction.
- The address in the memory for the next instruction, to be fetched and executed.

The next instruction which is executed is normally the next instruction following the current instruction in the memory. Therefore, no explicit reference to the next instruction is provided.

Instruction representation

An instruction is divided into a number of fields and is represented as a sequence of bits. Each of the fields constitutes an element of the instruction. A layout of an instruction is termed as the **instruction format**. (See Figure 1.41)

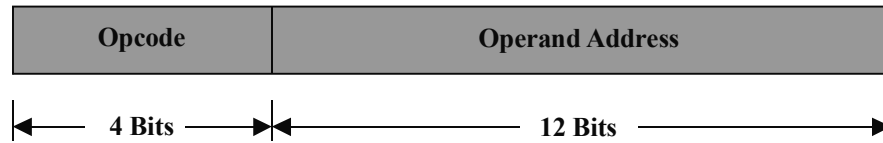


Fig. 1.41 A Sample Instruction Format

In most instruction sets, many instruction formats are used. An instruction is first read into an instruction register (IR), then it is decoded by the CPU which extracts the required operands on the basis of references made on the instruction fields, and processes it. Since the binary representation of the instruction is difficult to comprehend, it is seldom used for representation. Instead, a symbolic representation is used. (See Figure 1.42)

Instruction	Interpretation	Number of Addresses
ADD A,B,C	Operation $A = B + C$ is executed	3
ADD A,B	$A = A + B$. In this case the original content of operand location is lost	2
ADD A	$AC = AC + A$. Here A is added to the accumulator	1

Fig. 1.42 Examples of Typical Instructions

Typically, CPUs manufactured by different manufacturers have different instruction sets. This is why machine language programs developed for a particular CPU do not run on a computer with a different CPU (having a different instruction set).

Registers

The primary task that the CPU performs is the execution of instructions. It executes every instruction by means of a number of small operations known as micro-operations. Thus, it can be seen that:

- The CPU needs an extremely large main memory
- The speed of the CPU must be as fast as possible

To understand further, let us define two relevant terms:

Memory cycle time: Time taken by the CPU to access the memory.

Cycle time of the CPU: The time that the CPU takes for executing the shortest well-defined micro-operation.

It has been observed that the time taken by the CPU to access the memory is about 1–10 times higher than the time that the CPU takes for executing the shortest well-defined micro-operation. Therefore, CPU registers serve as temporary

NOTES

NOTES

storage areas within the CPU. CPU registers are termed as fast memory and can be accessed almost instantaneously.

Further, the number of bits a register can store at a time is called the length of the register. Most CPUs sold today have 32-bit or 64-bit registers. The size of the register is also called the word size and indicates the amount of data that a CPU can process at a time. Thus, the bigger the word size, the faster the computer can process data.

The number of registers varies among computers, but typical registers found in most computers include:

Memory Buffer Register (MBR): Data is received from the memory (in the case of read operations), and it is held in the memory (in the case of write operations) by MBR.

Memory Address Register (MAR): The memory location's address where data is to be stored (in case of write operations) and the location from where data is to be accessed is (in case of read operations) is specified by MAR.

Accumulator (AC): Interactions with the ALU are carried out by the accumulator, in which the output and input operands are stored. This register, therefore, holds the initial data to be operated upon, the intermediate results and the final results of processing operations.

Program Counter (PC): The next instruction to be executed subsequent to the current instruction being executed is kept track of by the program counter.

Instruction Register (IR): Instructions are loaded in the instruction register prior to being executed, i.e., the instruction register holds the current instruction that is being executed.

Instruction processing, in its simplest form, can be defined as a two-step process:

1. Codes or instructions are read (fetched) from the CPU one by one.
2. The operation indicated by that particular instruction is performed or executed.

The PC fetches the instruction and tracks which instruction to fetch next. Since the execution of a program is in a sequential manner, the PC usually fetches the next instruction in the sequence. This instruction appears in the binary code form and is loaded into an IR. It is then interpreted by the CPU, and the desired action is carried out. Generally, the following categories can be identified for these actions:

- *Data transfer:* From I/O to CPU, from CPU to I/O, from memory to CPU, or from CPU to memory.
- *Data processing:* A logic or an arithmetic operation may be carried out on the data by the CPU.
- *Sequence control:* This action is typically required for altering the sequence in which the instructions are executed. For instance, if an instruction from location 50 specifies that the next instruction to be fetched should be from location 100, then the program counter will need to be

modified to contain the location 100 (which otherwise would have contained 51).

Instructions can be executed involving many combinations of these actions.

Processor speed

The term processor has replaced the term central processing unit (CPU). The processor in a personal computer that is embedded in small devices is often called a **microprocessor**.

The speed at which the processor executes commands is called the processor speed or **clock speed**. Every computer contains an internal clock (known as the system clock) that regulates the rate at which the instructions are executed and synchronises the various computer components. The processor requires a fixed number of clock cycles (electric pulses per second) to execute each instruction. Clock cycles are required to fetch, decode and execute a single program instruction. Thus, the shorter the clock cycle, the faster the processor.

In a computer, clock speed therefore, refers to the number of pulses per second generated by an oscillator that sets the tempo for the processor. It is usually measured in MHz (**Megahertz** - Million pulses per second) or GHz (**Gigahertz** - Billions pulses per second).

Computer clock speed has been roughly doubling every year. The Intel 8088, common in computers around the year 1990, ran at 4.77 MHz. Today's personal computers run at clock speeds of a 100–1000 MHz and some even exceed one gigahertz.

Although the processing speed in personal computers is measured in terms of megahertz, the processing speed of mini computers and mainframe systems is measured in terms of MIPS (millions of instructions per second) or BIPS (billions of instructions per second). This is because personal computers generally employ a single microprocessor chip as their CPU while other classes of computers employ multiple processors to speed up their overall performance. Thus a minicomputer having a speed of 500 MIPS is capable of executing 500 million instructions per second.

Clock speed is a measure of computer 'power,' but it is not always directly proportional to the performance level. If you double the speed of the clock, leaving all other hardware unchanged, you will not necessarily double the processing speed. The type of microprocessor, the bus architecture, and the nature of the instruction set, all make a difference. In some applications the amount of random access memory (RAM) is important too.

Memory

Data and instructions are stored and subsequently retrieved from a computer's memory. We saw earlier that the CPU contains several registers for storing data and instructions. But these can store only a few bytes. If all the instructions and data being executed by the CPU were to reside in Secondary storage (like magnetic tapes and disks), and be loaded into the registers of the CPU as the program execution proceeded, it would lead to the CPU being idle for most of the time

NOTES

NOTES

since the speed at which the CPU processes data is much higher than the speed at which data can be transferred from disks to registers. Every computer thus requires storage space where instructions and data of a program can reside temporarily when the program is being executed. This temporary storage area is built into the computer hardware and is known as the primary storage or main memory. Devices that provide backup storage (like magnetic tapes and disks) are called secondary storage or auxiliary memory.

Thus, the following three sets of memories form the memory system:

- (a) **Primary storage or main memory:** The primary storage refers to a large memory that is faster than the auxiliary memory and slower than the internal memory. It is primarily based on internal circuits and directly interacts with the CPU.
- (b) **Secondary storage or auxiliary memory or backing store:** This exceeds the main memory in size, but falls behind it in speed. All the software and system programs are usually stored in the secondary memory.
- (c) **Internal processor memory:** It consists of a small set of high-speed registers that are internal to a processor and are used as temporary locations for the actual processing to be done.

Another type of memory, the cache memory, is widely gaining acceptance in modern computers. Its logical position is between the main memory and the internal memory (registers). Its primary function is to store and cache parts of the main memory's contents that the processor is currently using.

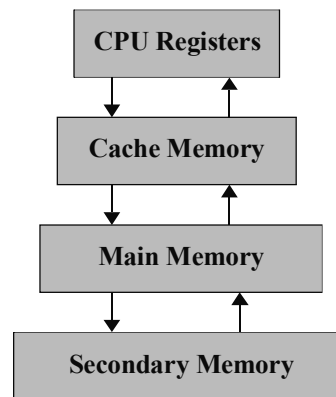


Fig. 1.43 The Memory Hierarchy

The total memory capacity of the computer can therefore be visualized as being a hierarchy of components consisting of all storage devices employed in a computer system from the slow but high-capacity auxiliary memory, to a relatively faster main memory, to an even smaller and faster cache memory accessible to the high-speed processing logic. Thus, as one goes down in the hierarchy, the following conditions occur:

1. Decreasing cost per bit
2. Increasing capacity

3. Increasing access time
4. Decreasing frequency of access of the memory by the processor

Memory Capacity

Capacity, in a computer system, is defined in terms of the number of bytes that it can store in its main memory. This is usually stated in terms of kilobytes (kB) which is 1024 bytes, or Megabytes (MB) which is equal to 1024 KB (10,48,576 bytes). The rapidly increasing memory capacity of computer systems has resulted in defining the capacity in terms of Gigabytes (GB) which is 1024 MB (1,07,37,41,824 bytes).

Thus, a computer system having a memory of 256 MB is capable of storing $(256 \times 1024 \times 1024)$ 26,84,35,456 bytes or characters.

Factors Affecting Speed

The speed or performance of a computer depends both upon the combination of hardware and software, and data. The fastest CPU, overloaded with fragmented data, or combined with a sluggish Operating System, will result in slower operations. Generally, if the speed of a computer decreases over time, it's either because of failing hardware, software clutter or due to the presence of malicious software.

The factors affecting the speed of a computer can be broadly classified into two major groups:

Hardware

Different components inside a computer work together in unison, and their combined efficiency determines the final throughput speed of the computer. The main components inside a computer which determine computer speeds are:

Processor: The CPU (Central Processor Unit) which is both the heart and the brain of a computer is responsible for performing all the calculations and all tasks pertaining to processing the data. The clock speed, bit size, and the amount of cache present in the CPU collectively affect the overall speed. A burned out CPU from over-clocking or over-heating is bound to perform low.

Motherboard: All the components on the computer are interconnected via the motherboard, also known as the backbone of the computer. The motherboard's design, technology and compatibility affect the overall performance of the computer, and determine how well the parts integrated on it perform collectively. A fast processor on a slow motherboard with lower bus speed would not perform at its optimum level.

RAM: Random Access Memory is the basic memory in which the operating system, the application software as well as the data in question are loaded for processing. An insufficient RAM would cramp the performance of a computer since there is little working space for the computer to store its interim results and semi-processed data, thereby necessitating data to be continuously deleted or moved from memory (to the hard disk) to make space for new calculations. Different types of RAM also determine their respective speeds, for instance a DDR DIMM RAM is slower than DDR2 and DDR3 RAM.

NOTES

NOTES

Hard Disk Drive: The speed of a HDD, measured in RPM (Rotations per Minute) determines the optimum speed at which the HDD can retrieve and store data. A HDD at, or near, full capacity provides less space to the operating system for making temporary files, such as page files, thereby causing the computer operations to slow down. If the hard disk drive has too many bad clusters or has highly fragmented data, this slows down the read and write speed—also called the Input / Output (IO) operations- of the computer.

Unused network drives and printers: The operating system reconnects and maps network drives and other shared resources such as printers and plotters every time it boots, or prepares itself for any input, output or printing operations, thus causing the system to slow down.

Besides these, the availability of specific performance boosting hardware such as a graphics card (used for games and heavy graphics processing), or a sound card with built-in memory for sound processing reduce the load on the CPU and memory space, and thus increase the overall computer speed.

Software and Data

The speed at which your computer performs now is probably much slower than what it was a few months ago. This cannot only be attributed to new and more resource demanding applications but to the fact that over time, computers get slower for a number of reasons such as:

Fragmentation of Data: Over a period of time when files are routinely created, edited and deleted, this causes data to be stored in a fragmented manner on the HDD. Whenever the user creates a file, the entire file is not stored in a physically contiguous location, different pieces of this file are scattered all over the HDD, depending upon wherever free space is available. These different pieces are linked to each other through a File Allocation Table (FAT) which tells the computer how to link these pieces together. Similarly, deleting files causes different segments of the HDD to become free. Over time, this continuous deletion of old files and creation of new ones causes the data to be stored on the HDD in a highly fragmented manner. This fragmentation increases the disk read/write time because the read/write head has to continuously move from one location to another to read or write files. More fragmentation means that the HDD has to move over a larger physical area for reading and writing data, thereby causing the computer to slow down.

Corruption of Programs and Data: The presence of malicious softwares such as Viruses, Adware and Malware causes computer resources to become overloaded with unnecessary and undesirable activities being processed in the background, thereby significantly slowing down the computer's ability to process authentic data. Since most of the time viruses and malware function in the background, the users are unaware of their presence, but suffer the consequences in terms of reduced overall speed and efficiency of the computer.

Applications: When several applications or processes are running at the same time, this necessitates the system resources such as CPU time and memory to be shared (called multi tasking) across different applications. Therefore, the

greater the number of applications that are running simultaneously, the less system resources are available for each of them, thereby reducing the speed of operations. Also, there are some applications that run faster than others even though they use the same amount of system resources. This variation occurs due to the better compatibility that exists between the software and hardware, which enables the higher optimization of computer resources, resulting in improved speed of operations.

System Registry: Every time you install or uninstall software, the Systems Registry is updated automatically. Over a period of time, when you have completed several installation or un-installation processes, this tends to clutter the Systems Registry, which if cluttered, can really slow down the computer operations.

Desktop: Giving the desktop a beautiful make-over, installing multimedia themes and making too many shortcuts on the desktop will slow down its performance as they constantly use up system resources, specially the RAM. You have to keep in mind that every icon stored on your desktop takes up a bit of your RAM space. Similarly, loading a beautiful but big-sized image as your wallpaper may improve the look of your desktop but will certainly slow down the speed of your computer since this large image effectively blocks up a small portion of the computer memory permanently.

Startup Programs: Some softwares load a small monitor or assistant program at windows startup. This helps them to monitor related trigger events or to decrease their startup time, as in the case of Power DVD, Microsoft Office, etc. This slows the overall speed of the system.

NOTES

1.10 PRINCIPLES OF PROGRAMMING: ALGORITHMS AND FLOW CHARTS

In order to write computer programs without any logical error, it is recommended to programmers to prepare a rough writing to show the steps involved in the program. This is called an algorithm. An algorithm is a rough writing of a program. It contains step-by-step instructions to solve a given problem. The steps must appear in the order in which they are executed. The information to be given (input), computed (processing) and printed are identified. The sequence of instructions in an algorithm is written with the following characteristics:

- (i) Instructions should be written in the correct sequence in which they are to be executed.
- (ii) Instructions should be precise and unambiguous.
- (iii) Instructions should be executed or repeated only a finite number of times.
- (iv) Check for possible infinite loop.
- (v) Make sure the instructions in the algorithm are written in the correct order.

Example 1.1. Write an algorithm and draw a flowchart to find the biggest of the given two numbers.



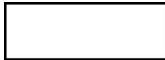
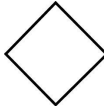



NOTES

Solution: For this problem, a new name *big* is used to store the biggest value. Initially, *a* is assumed as *big*, then *b* is compared with the existing *big* to get the biggest value.

1. Read *a, b*
2. $\text{big} \leftarrow a$
3. If $b > \text{big}$ then
 $\text{big} \leftarrow b$
4. Print *big*
5. Stop

Flowcharts

An algorithm gives the step-by-step instructions required to solve any problem. These steps can be shown diagrammatically using a flowchart. The flowchart 'as mentioned' is a symbolic or diagrammatic representation of an algorithm. It uses several geometrical figures to represent the operations and arrows to show the direction of flow. The following are the commonly used symbols in flowcharts.

Symbol	Operation	Meaning
	Start/Stop	Represents the beginning and the end of the flowchart.
	Input/Output	Represents the values to be given the user and the results to be displayed.
	Processing	Represents the arithmetic operations to compute a value.
	Checking/decision-making	Represents the logical checking to decide the flow sequence.
	Looping	Represents the looping which is repeated based on a condition/value of a variable.
	Connector	Represents the continuity of the flowchart in another place/page.
	Arrows	Represents direction of flow.

It is recommended that beginners must practice algorithm and flowcharts before starting to write programs.

Before writing a program code, the sequence of statements and the relationship between the various elements are shown with the help of a flowchart

or pseudocode. The flowchart is a common method to define the logical steps of flow within the program. It uses various symbols to represent the functions within the program. A flowchart shows the sequence, selection and iterations within a program. Consider the following flowchart to find the biggest of given two numbers.

NOTES

Rules for Flowcharting

- (i) Use consistent methods in drawing a flowchart.
- (ii) Use common and easily understandable words.
- (iii) Use consistent words or names in the flowchart.
- (iv) Avoid crossing flow lines in the flowchart.
- (v) Draw the flowchart from top to bottom and left to right.
- (vi) Flowcharts that exceed a page should be properly linked using connectors to the portions of the flowchart on different pages.

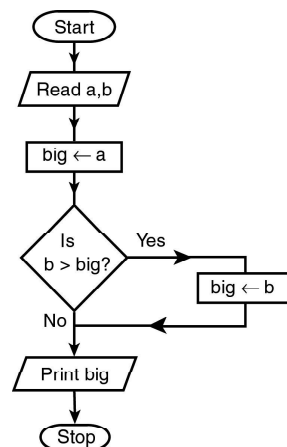
Advantages of Flowchart

- (i) A flowchart can easily explain the program logic to a program development team.
- (ii) It is useful to prepare a detailed program documentation.
- (iii) Its detail help prepare efficient program coding.
- (iv) The flowchart helps detect and remove the mistakes in a program.
- (v) It is useful to test the logic of the program.

Limitations of Flowchart

- (i) Flowcharting is a laborious process when proper symbols are used.
- (ii) Modifications in flowcharting are very difficult and it consumes more time.
- (iii) Redrawing the flowchart is also a tedious task.
- (iv) No standards are strictly followed to draw the flowchart.

Example 1.2. Draw a flowchart to find the biggest of the given two numbers.



Example 1.3. Write the algorithm and draw the flowchart to find the sum and product of given two numbers.

NOTES

Solution: It is necessary to understand the data given in the problem and the results expected.

In this problem, two numbers, say a and b , are given (input) and the results, Sum ($a + b$) of two numbers and Product ($a \times b$) of two numbers, are to be calculated.

Algorithm

1. Read a, b
2. Sum $\leftarrow a + b$
3. Product $\leftarrow a \times b$
4. Print Sum, Product
5. Stop

Notes:

- (i) Usually words, such as Read, Accept or Input can be used to represent input operation to give values of variables to the computer.
- (ii) The words print, Write or Display can be used to represent output operations to show the results computed by the computer.
- (iii) Back arrow (\leftarrow) represents the value obtained by evaluating the right side expression/variable to the left side variable. The symbol '=' can also be used instead of ' \leftarrow ' but it leads to confusion in certain applications. (for example, $S = S + X$) representing the logical equality, and so on.
- (iv) Down arrow (\downarrow) is optional.

Check Your Progress

9. What is MS DOS?
10. Write one of the security features of Windows XP.
11. Why was MULTICS operating system designed?
12. What is bit?
13. What does an algorithm contain?
14. What is a flowchart?

1.11 ANSWERS TO 'CHECK YOUR PROGRESS'

1. The two basic operations of computers are as follows:
 - i. It exchanges information with the outside world via Input/Output (I/O) devices.
 - ii. It transfers data internally within the CPU.
2. The three essential parts of computers are the input device, the CPU and the output device.
3. Hand-held input devices include trackball, joystick, and light pen and touch screen.

4. An electronic card reader is a device that allows direct data input into a computer system. It is connected to a computer system and reads the data encoded in an electronic card and transfers it to the computer for further processing.
5. Printers can be classified on the basis of their printing technology, printing speed, or their printing quality.
6. The storage unit of a computer system is evaluated on the basis of its storage capacity, storage cost, access time, access mode and permanence of storage.
7. A disk drive is a peripheral device used to store and collect information. It can be removable or fixed, high capacity or low capacity, fast or slow speed, and magnetic or optical.
8. Instructions written using sequences of 0s and 1s are known as a machine language.
9. Microsoft Disk Operating System (MS DOS) is a single user, single tasking operating system.
10. Windows XP's Security Zone feature lets you assign security options for Website access. It restricts you from exploring different sites by selecting the desired security level and asks for your approval before opening a file or running a program from the Internet.
11. The MULTICS operating system was designed to run the GE-645 mainframe computers.
12. A BIT is the smallest element that a computer uses. It can hold one of two possible values.
13. An algorithm is a rough writing of a program. It contains step-by-step instructions to solve a given problem.
14. The flowchart is a symbolic or diagrammatic representation of an algorithm.

NOTES

1.12 SUMMARY

- The input unit performs the process of transferring data and instructions from the external environment into the computer system. Instructions and data enter the input unit depending upon the particular input device used (keyboard, scanner, card reader, etc).
- Processing refers to the performing of arithmetic or logical operations on data, to convert them into useful information. Arithmetic operations include operations of add, subtract, multiply, divide and logical operations are operations of comparison like less than, equal to, greater than.
- A computer has limited storage capacity. It becomes necessary to store large volumes of data. Therefore, additional memory called secondary storage or auxiliary memory is used in most computer systems.
- The control unit and arithmetic logic unit are together known as the central processing unit. It is the brain of any computer system.

NOTES

- Keyboard devices allow input into the computer system by pressing a set of keys, mounted on a board connected to the computer system. Keyboard devices are typically classified as general-purpose keyboards and special-purpose keyboards.
- A mouse is a small device that is pushed across a desk surface by a user. It serves the function of indicating a point on the display screen and selects one or more possible actions from that particular location.
- A touchpad is a touch sensitive input device which takes user input to control the onscreen pointer and perform other functions similar to that of a mouse.
- The trackball is a pointing device that is much like an inverted mouse. It consists of a ball inset in a small external box, or adjacent to—and in the same unit as—the keyboard of some portable computers.
- The joystick is a vertical stick that moves the graphic cursor in the direction the stick is moved.
- An output device is an electromechanical device that accepts data from the computer and translates it into a form that can be understood by the outside world.
- The display device is an important peripheral component of a computer system. Display terminals called alphanumeric terminals were used by conventional computers. These used a form of multi-dot (7×5 or 9×7) array to display characters.
- The main memory is the central storage unit in a computer system. It is a relatively large and fast memory. It is used to store programs and data during computer operations.
- The storage unit of a computer system is evaluated on the basis of its storage capacity, storage cost, access time, access mode and permanence of storage.
- A disk drive is a peripheral device used to store and collect information. It can be removable or fixed, high capacity or low capacity, fast or slow speed, and magnetic or optical.
- A computer language is a language that can be understood by the computer. It is the computer's native language. A computer language consists of lexicon and syntax, i.e., characters, symbols and rules of usage that allow the user to communicate with the computer.
- Microsoft Disk Operating System (MS DOS) is a single user, single tasking operating system.
- Windows XP's Security Zone feature lets you assign security options for Website access. It restricts you from exploring different sites by selecting the desired security level and asks for your approval before opening a file or running a program from the Internet.
- Menus are very useful parts of the Windows system. They consist of various commands and subcommands for default and other applications and programs stored on your computer.

- UNIX is a multi-user and multitasking operating system. In a multi-user environment, the computer can receive the commands from a number of end users to run programs, access files, and print documents simultaneously.
- A bit is the smallest element that a computer uses. It can hold one of two possible values.
- An algorithm is a rough writing of a program. It contains step-by-step instructions to solve a given problem.
- The flowchart is a symbolic or diagrammatic representation of an algorithm.

NOTES

1.13 KEY TERMS

- **Central Processing Unit:** It is the portion of a computer that retrieves and executes instructions.
- **I/O Devices:** These are the devices that provides a means of communication between the computer and the external world.
- **Keyboard:** It that allows input into the computer system by pressing a set of keys, mounted on a board connected to the computer system.
- **Mouse:** A small device that is pushed across a desk surface by a user. It serves the function of indicating a point on the display screen and selects one or more possible actions from that particular location.
- **Touchpad:** A touch sensitive input device which takes user input to control the onscreen pointer and perform other functions similar to that of a mouse.
- **Printers:** Devices used for producing output on paper.
- **Cache Memory:** This memory provides data retrieval without loss of memory size.
- **Interpreter:** A computer program executing instructions written in a programming language.

1.14 SELF-ASSESSMENT QUESTIONS AND EXERCISES

Short-Answer Questions

1. State about the control unit.
2. What is arithmetic logic unit?
3. State about the dot matrix printer.
4. What is the function of I/O devices in a computer system?
5. What are special-purpose keyboards?
6. Write the functions of interpreter.
7. Write the limitations of MS DOS.
8. How to switch between programs in Windows XP?

NOTES

9. What is UNIX?
10. Define the term instruction set.
11. What are the advantages of flowchart?

Long-Answer Questions

1. Briefly explain the basic design and components of a computer with the help of diagram.
2. Discuss the special function keys on a keyboard giving appropriate examples.
3. Describe the input/output devices with the help of appropriate example.
4. Discuss the functioning of the cache memory.
5. Briefly explain the memory and its types with the help of example.
6. Discuss the difference between machine language and assembly language with the help of examples.
7. Describe the simple DOS commands and file operations commands with the help of examples.
8. Explain the installation process of mouse and printers in Windows XP.
9. Briefly explain UNIX operating system and its features.
10. Describe some of the memory codes used to represent data.
11. Briefly explain the importance of algorithms and flowcharts.

1.15 FURTHER READING

- Dey, Pradip and Manas Ghosh. *Computer Fundamentals and Programming in C*. New Delhi: Oxford Higher Education, 2006.
- Bronson, Gary J. *A First Book of ANSI C*, 3rd edition. California: Thomson, Brooks Cole, 2000.
- Kanetkar, Yashwant. *Understanding Pointers in C*. New Delhi: BPB Publication, 2001.
- Kanetkar, Yashwant. *Let us C*. New Delhi: BPB Publication, 1999.
- Kernighan, Brian W. and Dennis Ritchie. *C Programming Language*, 2nd edition. New Jersey: Prentice Hall, 1988.
- Foster, W. D. and L. S. Foster. *C by Discovery*. Boston: Addison-Wesley, 2005.
- Kanetkar, Yashwant. *Working with C*. New Delhi: BPB Publication, 2003.
- Horton, Ivor. *Instant C Programming*. New Jersey: Wrox Press (John Willey & Sons), 1995.
- Lawlor, Steven C. *The Art of Programming Computer Science with 'C'*. New Jersey: West Publishing Company, 1996.

UNIT 2 COMPUTER PROGRAMMING IN FORTRAN/C/BASIC

NOTES

Structure

- 2.0 Introduction
- 2.1 Objectives
- 2.2 Introduction to C Language
 - 2.2.1 Importance of C
 - 2.2.2 Execution of a C Program
- 2.3 Elements of the Computer Language
 - 2.3.1 Character Set
 - 2.3.2 Identifiers and Keywords
 - 2.3.3 Data Types
- 2.4 Constants and Variables
- 2.5 Operators and Expressions
 - 2.5.1 Operators
 - 2.5.2 Expressions
- 2.6 Input/Output Operations
- 2.7 Branching
 - 2.7.1 if Statement
 - 2.7.2 if...else Statement
 - 2.7.3 Nesting of the if...else Statements
 - 2.7.4 Logical Operators and Branching
 - 2.7.5 Conditional Operator and if...else
- 2.8 Loops and Control Constructs
 - 2.8.1 Iteration using if
 - 2.8.2 for Statement
 - 2.8.3 Symbolic Constants and Looping
 - 2.8.4 Other Forms of the for Loop
 - 2.8.5 The while Loop
 - 2.8.6 do . . . while Loop
 - 2.8.7 switch Statement
 - 2.8.8 Break, Continue, Return and goto
- 2.9 Logical Variables
- 2.10 Double Precision Variable
- 2.11 Subscripted and Dimension Variable
- 2.12 Function and Subroutine
- 2.13 Common and Data Commands
- 2.14 Answers to 'Check Your Progress'
- 2.15 Summary
- 2.16 Key Terms
- 2.17 Self-Assessment Questions and Exercises
- 2.18 Further Reading

NOTES

2.0 INTRODUCTION

C language is a powerful programming language that is used to write programs for a variety of applications. It was developed in the 1970's by Dennis Ritchie at Bell Laboratories. It has several flexible features to write programs for numerical, commercial and graphical applications. In addition, it is used by software developers to write programs for lower level accessing along with assembly language. Several operating system programs for the latest computers and compilers are written using C language. In this unit, you will learn about the basics of C language, elements of computer language, constants and variables, operators, expressions, I/O operations, decision, looping and branching statements.

2.1 OBJECTIVES

After going through this unit, you will be able to:

- Understand the basics of C language
- Describe the various elements of computer language
- Explain the significance of constants and variables
- Enumerate the operators and expressions used in a C program
- Discuss the concept of type casting
- Understand the input and output operations
- Describe the formatting of output in C programs
- Understand the basic concept of branching
- Use `if` and `if...else` branching statements in C programs
- Use `if`, `for`, `while` and `do...while` in a program
- Define `switch`, `break`, `continue` and `goto` statements
- Define the logical and double precision variables
- Elaborate on the subscripted variables and dimension
- Describe the function and subroutine

2.2 INTRODUCTION TO C LANGUAGE

The C language was developed under the influence of BCPL (Basic Combined Programming Language) and B. It is developed in 1960s at Cambridge University. B language was modified by Dennis Ritchie and was implemented at Bell Laboratories in 1972. The new language was named as C. Since, it was developed along with the UNIX operating system, it is strongly associated with UNIX. This operating system was developed at Bell Laboratories and was coded almost entirely in C.

NOTES

C was used mainly in academic environments for many years, but eventually with the release of C compiler for commercial use and the increasing popularity of UNIX, it began to gain widespread support among compiler professionals. Today, C is running under a number of operating systems including MS-DOS. It is now standardized by American National Standard Institute (ANSI C).

2.2.1 Importance of C

The popularity of C is increasing probably due to its many desirable qualities. It is a robust language having rich set of built-in functions and operators that can be used to write any complex program. The C compiler combines the capabilities of an assembly language with the features of a high-level language and therefore it is well suited for writing both system software and business packages. In fact, many of the C compilers available in the market are written in C. Programs written in C are efficient and fast. This is due to its rich set of data types and powerful operators. It is many times faster than BASIC (Beginners All Purpose Symbolic Instruction Code – a high level programming language).

There are only 32 keywords and its strength lies in its built-in functions. Several standard functions are available which can be used for developing programs. C is highly portable. This means that C programs written for one computer can be run on another with little or no modification. C Language is well suited for structured programming thus requires the user to think of a problem in terms of function modules or blocks. A proper collection of these modules would make a complete program. This modular structure makes program debugging, testing, and maintenance.

Another important feature of C is its ability to extend itself. A C program is basically a collection of functions that are supported by the C library. We can continuously add our own function to the C library. The structure of a C program is given below.

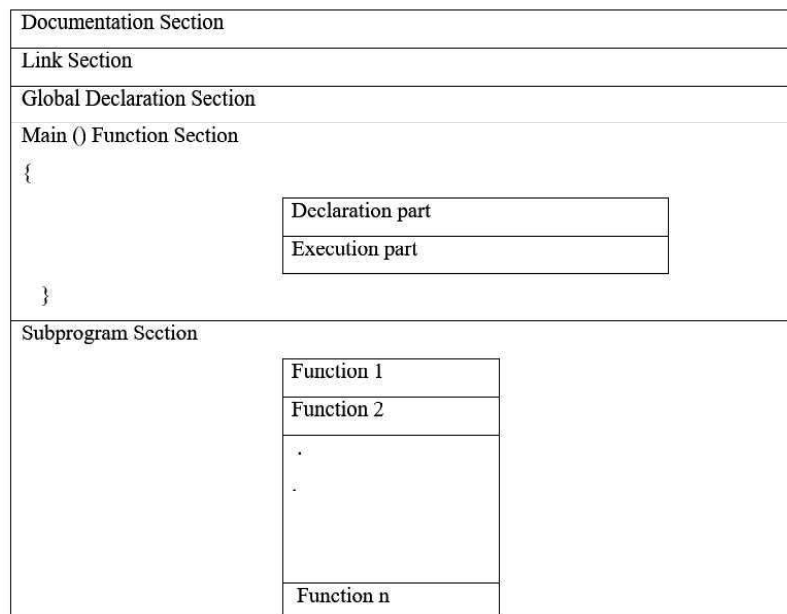


Fig. 2.1 Basic Structure of a C program

NOTES

The documentation section consists of a set of comment lines giving the name of the program, the programmer, and other details which the programmer would like to use later. The link section provides instructions to the compiler to link function from the system library. The definition defines all the symbolic constants. There are some variables that are used in more than one functions. Such variables are called global variables and are declared in global declaration section that is outside of all the function.

Every C program must have one main () function section. This section consists two parts: declaration part and executable part. The declaration part declares all the variables used in the executable part. These two parts must appear between the opening and the closing braces. The program execution begins at the opening braces and ends at the closing braces. The closing brace of the main () function section is the logical end of the program. All the statements in the declaration and executable parts ends with a semicolon.

The subprogram section contains all the user-defined functions that are called in the main () function. User-defined functions are generally placed immediately after the main () function, although they may appear in any order. All section, except the main () function section may be absent when they are not required.

2.2.2 Execution of a C Program

A program written in C involves a series of steps for executing:

1. Creating the program;
2. Compiling the program;
3. Linking the program with functions that are needed from the C library; and
4. Executing the program.

The program can be created using any word processing software in non-document mode. The file name should end with the characters “.c” like program .c, lab1.c, etc. Then, the command under MS DOS operating system would load the program stored in the file program.c i.e. MSC pay .C and generate the object code. This code is stored in another file under name ‘program.obj’. In case any language errors are found, the compilation phase generate errors. Then, the program should be corrected and compiled again. The linking is done by the command link program.obj which generates the executable code with the filename program.exe. Now the command.Program would execute the program and generate the results.

Program 1: write a C program to print Hello, World and save it as hello.c.

```
# include <stdio.h> /* header file */
main ( ) /* main ( ) function */
{
    Print ("Hello, World \n"); /* statement */
}
```

Output: Hello, World

Program 2: write a C program to calculate the area of a circle.

```
# include <stdio.h> /* library file access */
main ( ) /* function heading */
{
float radius, area; /* variable decleration */
clrscr (); /* Console function :Used to clear screen */
printf ("Enter radius?="); /* output statement */
scanf ("%f", & radius); /* input statement */
area = 3.14159 * radius*radius ; /* assignment statement
*/

printf ("Area=%f", area); /* output statement */
getch (); /* Console function :Used to hold the same
screen */
}
```

NOTES

2.3 ELEMENTS OF THE COMPUTER LANGUAGE

C Fundamentals is concerned with the basic elements used to construct simple C statements. These elements include the C character set, identifiers and keywords, data types, constants, variables and arrays, declaration of expressions and statements.

A programming language is designed to help process certain kinds of data consisting of numbers, characters and strings and to provide useful output known as information. The task of programming of data is accomplished by executing a sequence of precise instructions called a program. These instructions are formed using certain symbols and words according to some rigid rules known as syntax rules.

2.3.1 Character Set

C uses the uppercase letters A to Z, the lowercase letters a to z, the digits 0 to 9, and certain special characters as building blocks to form basic program elements (e.g. constants, variables, operators, expressions, etc.). The special characters are listed below:

+ - * / = % & # ! ? ^ " ' ~ \ | < > () [] { } ; : , . - (Blank space) (Horizontal tab) (White Space)

Most versions of the language also allow certain other characters, such as @ and \$ to be included with strings & comments.

2.3.2 Identifiers and Keywords

C Tokens: In a passage of text, individual words and punctuation marks are called tokens. Similarly, in a C program the smallest individual units are also known as C tokens. C has six types of tokens:

NOTES

- Identifiers e.g.: x, area, temperature, PI
- Keywords e.g.: int, float, for, while
- Constants e.g.: -15.5 100
- Strings e.g.: “ABC”, “year”
- Operators e.g.: + - *
- Special Symbols e.g.: () [] { }

Identifiers: Identifiers are names that are given to various program elements, such as variables, functions and arrays. Identifiers consisted of letters and digits, in any order, except that first character must be a letter. Both upper and lower case letters are permitted, though common usage favors the use of lowercase letters for most type of identifiers. Upper and lowercase letters are not interchangeable (i.e. an uppercase letter is not equivalent to the corresponding lowercase letters). The underscore (`_`) can also be included, and considered to be a letter. An underscore is often used in middle of an identifier. An identifier may also begin with an underscore.

Rules for defining identifier:

1. First character must be an alphabet (or Underscore).
2. Must consist of only letters, digits or underscore.
3. Only first 31 characters are significant.
4. Cannot use a keyword.
5. Must not contain white space.

The following names are valid identifiers:

X, a12, sum_1, _temp, name, area, tax_rate, TABLE

The following names are not valid identifier for the reason stated

- 4th The first character must be letter
- “x” Illegal characters (“
- Order-no Illegal character (-)
- Error flag Illegal character (blank space)

Keywords: 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 programmer-defined identifiers. Some of the standard keywords are given below:

auto	Break	case	char	const
continue	default	do	double	else
enum	extern	float	for	goto
if	Int	long	register	return
short	signed	sizeof	static	struct
switch	typedef	union	unsigned	void
volatile	While			

All the keywords are in lowercase. Since upper and lowercase characters are not equivalent, it is possible to utilize an uppercase keyword as an identifier. However, this is not done, as it is considered a poor programming practice.

2.3.3 Data Types

C language is rich in its data types. C supports different types of data, each of which may be represented differently within the computer memory. There are three categories of data types.

1. Basic data types (Primary or Fundamental) e.g.: int, char
2. Derived data types e.g.: array, pointer, function etc.
3. User defined data types e.g.: structure, union, enum etc.

The basic data types are also known as built in data types. Some of the built-in data types are listed below. Typical memory requirements are also given:

Data Types	Description	Typical Memory Requirement
char	single character	1 byte
int	integer quantity	2 bytes
float	floating-point number	4 bytes
double	double-precision floating point number	8 bytes

In order to provide some control over the range of numbers and storage space, C has following classes: signed, unsigned, short, long.

Types	Size
char or signed char	1 byte
unsigned char	1 byte
int	2 bytes
short int	1 byte
unsigned short int	1 byte
signed int	2 bytes
unsigned int	2 bytes
long int	4 bytes
signed long int	4 bytes
unsigned long int	4 bytes
float	4 bytes
double	8 bytes
long double	10 bytes

void is also a built-in data type used to specify the type of function. The void type has no return values.

NOTES

2.4 CONSTANTS AND VARIABLES

NOTES

Constants in C refer to fixed values that do not change during the execution of a program. There are four types of constants in C. They are integer constants, floating point constants, character constants and string constants.

Integer and floating point constants represent numbers. They are often referred to collectively as numeric type constants. The following rules are applied to all numeric type constants.

1. Commas and blank spaces cannot be included within the constants.
2. The constant can be preceded by a minus (-) if required. The minus sign is an operator that changes the sign of a positive constant though it can be thought of as a part of the constant itself.
3. The value of a constant cannot exceed the specified minimum and maximum bounds. For each type of constant, these bounds will vary from one C compiler to another.

Integer Constants

An integer constant is an integer-valued number. Thus it consists of a sequence of digits. Integer constants can be written in three different number systems: decimal (base 10), octal (base 8) and hexadecimal (base 16).

A decimal integer constant can consist of any combination of digits taken from the set 0 through 9. If the constant contains two or more digits, the first digit must be something other than 0. Several valid decimal integer constants are shown below:

0 1 143 5280 12345 9999

The following decimal integer constants are written incorrectly for reason stated:

12,452	Illegal character (,)
36.0	Illegal character (.)
10 20 30	Illegal character (blank space)
123-45-6743	Illegal character (-)
0900	the first digit cannot be zero.

An octal integer constant can consist of any combination of digits taken from the set 0 through 7. However, the first digit must be 0, in order to identify the constant as an octal number.

Valid octal number (integer) constants are shown below:

0 01 0743 07777

The following octal integer constants are written incorrectly for the reason stated:

743	Does not begin with 0.
05280	Illegal character (8)
777.777	Illegal character (.)

A hexadecimal integer constant must begin with either 0x or 0X. It can then be followed by any combination of digits taken from the sets 0 through 9 and a through f (either upper or lower case). The letters a through f (or A through F) represent the (decimal) quantities 10 through 15 respectively. Several valid hexadecimal integer constants are shown below:

0x 0X1 0X7FFF 0xabcd

The following hexadecimal integer constants are written incorrectly for the reason stated:

0X12.34	Illegal character (.)
013E38	Doesn't begin with 0x or 0X.
0x.4bff	Illegal character (.)
0XDEFG	Illegal character (G)

NOTES

Unsigned and Long Integer Constants

Unsigned integer constants may exceed the magnitude of ordinary integer constants by approximately a factor of 1, though they may not be negative. An unsigned integer constant can be identified by appending the letter (U) (either upper or lowercase) to the end of the constant.

Long integer constants may exceed the magnitude of ordinary integer constants, but require more memory within the computer. A long integer constant can be identified by appending the letter L (either upper or lowercase) to the end of the constant.

An unsigned long integer may be specified by appending the letters UL to the end of the constant. The letters may be written in either upper or lowercase. However, the U must precede the L. Some unsigned and long integer constants are shown below:

Constant	Number System
50000 U	decimal (unsigned)
123456789 L	decimal (long)
123456789 UL	decimal (unsigned long)
0123456 L	octal (long)
0777777 U	octal (unsigned)
0X50000 U	hexadecimal (unsigned)
0XFFFFFFUL	hexadecimal (unsigned long)

Floating Point Constants

A floating point constant is a base 10 number that contains either a decimal point or an exponent (or both).

NOTES

Several valid floating point constants 0.

1. 0.2 827.602

500. 0.000743 12.3

2E.8 0.006e.3 1.6667e+8

The following are invalid floating point constants for the reason stated.

1. Either a decimal point or an exponent must be present.

1,000.0 Illegal character (,)

2. E+10.2 The exponent must be an integer (it cannot contain a decimal point)

3. E 10 Illegal character (blank space) in the exponent.

The quantity 3×10^5 can be represented in C by any of the following floating point constants:

300000. 3e5 3e+5 3E5 3.0e+5

.3e5 0.3E6 30E4 30.E4 300e3

Character Constants

A character constant is a single character, enclosed in apostrophes (i.e. single quotation marks). Some character constants are: 'A' 'X' '3' '?' ' '.

Character constants have integer values that are determined by the computer's particular character set. Thus, the value of a character constant may vary from one computer to another. The constants themselves, however, are independent of the character set. This feature eliminates the dependence of a C program on a particular character set.

Most computers, and virtually all personal computer make use of ASCII (i.e. American Standard Code for Information Interchange) character set, in which each individual character is numerically encoded with its own unique 7-bit combination (hence a total of $2^7=128$ Different characters).

Some character constant and their corresponding values, as defined by ASCII character set are shown below:

Constant	Value
'A'	65
'X'	120
'3'	51
'?'	63
' '	32

These values will be the same in all computer that utilize the ASCII character set.

String Constants

A string consists of any number of consecutive characters (including none), enclosed in (double) quotation marks. Some string constants are shown below:

"green"	"Washinton, D.C. 2005"	"207-32-345"
"\$19.95"	"THE CORRECT ANSWER IS"	"2*(I+3)"
" "	"Line 1\n Line 2\n line 3"	" "

NOTES

The string constants "Line 1\n Line 2\n Line 3" extends over three lines, because of the newline characters that are embedded within the string. Thus, the string would be displayed as

Line 1

Line 2

Line 3

The compiler automatically places a null character (\0) at the end of every string constant, as the last character within the string (before the closing double quotation mark). This character is not visible when the string is displayed.

A character constant (e.g. 'A') and the corresponding single-character string constant ("A") are not equivalent. 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 is followed by the null character (\0).

Variables

A variable is an identifier that is used to represent some specified type of information within a designated portion of the program. In its simplest form, a variable is an identifier that is used to represent a single data item, i.e., a numerical quantity or a character constant. The data item must be assigned to the variable at some point in the program. A given variable can be assigned different data items at various places within the program. Thus, the information represented by the variable can change during the execution of the program. However, the data type associated with the variable cannot be changed.

Check Your Progress

1. What is logical variables?
2. State about the popularity of C language.
3. Define the term program.
4. Define identifiers.
5. Write the types of constants.
6. State about the term variables.

2.5 OPERATORS AND EXPRESSIONS

NOTES

2.5.1 Operators

An *operator* refers to a symbol which indicates an operation to be executed. Operators are used to manipulate data in a program. The data items that operators act upon are called *operands*.

E.g.: Sum = A + B

Where 'A' and 'B' are operands and '+' is an operator.

Classification of Operators

- **Arithmetic** [+ , - , * , / , %(modulo division)]
- **Assignment** [= , += , -= , *= , /= , %=]
- **Relational** [> , < , <= , >= , == , !=]
- **Logical** [&& , || , !]
- **Conditional** [<expression 1> ? <expression 2> : <expression 3>]
- **Increment and Decrement** [++ , --]
- **Bitwise** [~ , >> , << , & , | , ^]
- **Comma Operator** [,]
- **Address operator** [&]
- **Indirection operator** [*]
- **sizeof operator** [sizeof()]

Note: ~: is complement

>>: is right shift

<<: is left shift

&: bitwise AND

|: bitwise OR

^: bitwise Exclusive OR (XOR)

Arithmetic operators: These operators are used for arithmetic calculations. C has the following arithmetic operators:

Operator	Name	Purpose
+	plus	addition
-	minus	subtraction
*	asterisk	multiplication
/	slash	division (returns quotient)
%	modulus	division (returns remainder)

Note: Precedence of the arithmetic operators while manipulating arithmetic expressions

First preference is for the expression within the brackets (). Second precedence is for the asterisk (*), slash (/) and modulus (%) and the precedence for plus (+) and minus (-) is equal.

Demo program based on arithmetic operators

```
/*-----START OF PROGRAM-----*/
#include<stdio.h>
void main()
{ printf( "%d ", 10+2 );
  printf( "%d ", 10-2 );
  printf( "%d ", 10*2 );
  printf( "%d ", 10/2 );
  printf( "%d", 10%2 );
}
/*-----END OF THE PROGRAM-----*/
```

Output: 12 8 20 5 0

Explanation: In the preceding (arithmetic operator) program, all basic arithmetic operations have been used on integer values.

Demo program based on the precedence of arithmetic operators

```
/*-----START OF PROGRAM-----*/
#include<stdio.h>
void main()
{ printf( "%d ", 2+3*2 );
  printf( "%d ", 6/6*3 );
  printf( "%d ", 3+3/6 );
  printf( "%d", (3+3)/6 );
}
/*-----END OF THE PROGRAM-----*/
```

Output: 8 3 3 1

Explanation: In the preceding, `printf("%d ", 2+3*2);` statement, asterisk has got more precedence than plus; so the expression value is 8 rather than 10. In the `printf("%d ", 6/6*3);` statement, asterisk and slash have got same precedence and the manipulation will take place from left to right; so value of the expression is 3. In the `printf("%d ", 3+3/6);` statement, slash has got more precedence than plus; so the expression value is 3. In the `printf("%d", (3+3)/6);` statement, expression within the brackets has got more precedence than slash; so the expression value is 1.

Relational Operators

These operators are used to compare arithmetic, logical and character expressions.

Operator	Activity
==	equal
!=	not equal

NOTES

NOTES

<	less than
>	greater than
<=	less than or equal
>=	greater than or equal

Note:

- All the above operators compare their left-hand side with their right-hand side.
- All binary operators and expression returns Boolean value (0 or 1). So the returning type is integer. If the expression is TRUE, it returns 1; otherwise, 0.

Demo program to show typical characteristics of relational operators

```
#include<stdio.h>
void main()
{
    printf("%d ",2<3); /* true returns 1 */
    printf("%d ",2>3); /* false returns 0 */
    printf("%d ",2==3); /* false returns 0 */
    printf("%d ",2!=3); /* true returns 1 */
    printf("%d ",2<=3); /* true returns 1 */
    printf("%d ",2>=3); /* false returns 0 */
    printf("%d ",2<5<3); /* 2<5 = true(=1) */
        /* 1<3 = true(=1) */
        /* so, 2<5<3 is true */
}
/*-----END OF THE PROGRAM-----*/
```

Output: 1 0 0 0 1 1 0

Logical Operators

These operators are used to compare or evaluate logical and relational expressions. The result of the logical expression will be either *true* (1) or *false* (0).

Operator	Activity
&&	AND
	OR
!	NOT

&& (AND) operator: If the expressions each side of the && are true the result of the && is true. If one of them is false the result is false.

Tips to remember the functioning of && and || operators:

General representation

Representation in C

'false' AND <anything> becomes false. (0 && true/false) == 0

'true' OR <anything> becomes true.

(1 || true/false) == 1

|| (OR) operator: If either of the expressions each side of the || are true the result of the whole expression is true. The expression is only false if both expressions are false.

!(NOT) operator: For ! if operand is false (zero value) then result will be true vice versa.

NOTES

Demo program to display truth table of && (AND)

```
/*-----START OF PROGRAM-----*/
#include<stdio.h>
void main()
{
    printf("0 && 0 = %d \n",0 && 0);
    printf("0 && 1 = %d \n",0 && 1);
    printf("1 && 0 = %d \n",1 && 0);
    printf("1 && 1 = %d \n",1 && 1);
}
/*-----END OF THE PROGRAM-----*/
```

Output: 0 && 0 = 0
0 && 1 = 0
1 && 0 = 0
1 && 1 = 1

Note: For && if either operands are true (any non-zero value), then result will be true.

Demo program to show the typical characteristics of && (AND)

```
/*----- START OF PROGRAM -----*/
#include<stdio.h>
void main()
{
    int i=5,j=5;
    printf("%d ", 0 && (i=10));
    printf("%d ", 1 && (j=10));
    printf("i=%d j=%d",i,j);
}
/*----- END OF THE PROGRAM -----*/
```

Output: 0 1 i=5 j=10

Explanation: If operand1 is false, then && operator will not allow to check the second operand and return value of the result will be 0. For example, in the above example, first printf statement's first operand is 0; so irrespective of second expression's value, it will return 0.

Demo Program to display truth table of || (OR)

```
/*-----START OF PROGRAM-----*/
#include<stdio.h>
void main()
```

NOTES

```
{ printf("0 || 0 = %d \n",0 || 0);  
  printf("0 || 1 = %d \n",0 || 1);  
  printf("1 || 0 = %d \n",1 || 0);  
  printf("1 || 1 = %d \n",1 || 1);  
}  
/*————END OF THE PROGRAM————*/
```

Output: 0 || 0 = 0
0 || 1 = 1
1 || 0 = 1
1 || 1 = 1

Explanation: If operand1 is true, then || operator will not allow checking the second operand and returning value of the result will be 1. For example, in the above example, second printf statement's first operand is 1, so irrespective of the second expression, returns 1.

Note: For || if both operands are false (zero), then result will be false.

Demo program to display truth table of ! (NOT) operator

```
/*————Program Beginning————*/  
#include<stdio.h>  
void main()  
{  
  printf("!0 = %d \n",!0 );  
  printf("!1 = %d \n",!1);  
}  
/*————END OF THE PROGRAM————*/
```

Output: !0 = 1
!1 = 0

Explanation: The above out put shows the truth table of ! operator

Note: for ! if operand is false (zero value), then result will be true and vice versa.

Increment and Decrement Operators

C has two useful operators: ++ (*increment operator*) and — (*decrement operator*).

The ++ operator adds 1 to its operand and — operator subtracts 1 from its operand. These operators may be either pre-incremented/pre-decremented (precedes the operand) or post-incremented/post-decremented (follows the operand).

E.g.

```
1. int k,i=1;  
   k=i++;          /* k=i; i= i+1; */  
   printf("k=%d, i=%d.",k,i);
```

Result of the preceding statements will be k=1, i=2

Note: The expression `i++` will assign first and increments value of `k` later.

```
int k, i=1;
k=++i; /* i=i+1; k=i; */
printf("k=%d, i=%d.", k, i);
```

Result of the statements will be `k=2, i=2`.

```
2. int k, i=1;
k=++i; /* i=i+1; k=i; */
printf("k=%d, i=%d.", k, i);
```

Result of the preceding statements will be `k=2, i=2`.

Note: The expression `++i` will increment 1 first and this value is assigned to `k`.

Note: pre- or post-increments difference exists when we are assigning values; otherwise, there will be no difference.

E.g.: `i++` or `++i`

Bitwise Operators

These operators are used to operate with bits of data for complementing, testing, setting or shifting the bits of data in a byte or word.

Operator	Operator	Purpose
~	complement	for 1's complementation
>>	right shift	to move bits to right
<<	is left shift	to move bits to left
&	bitwise AND	to reset (0)/compare bits
	bitwise OR	to set (1)/compare bits
^	bitwise Exclusive OR (XOR)	to toggle or clear the bits

Note: Bitwise operators work only with `char` and `int` data types with its qualifiers and are not useful for float, double, void or complex data types.

Demo program for ~ (complement operator):

```
/*-----START OF PROGRAM-----*/
#include <stdio.h>
void main()
{
char num=10;
printf("%d", ~num);
}
/*-----END OF THE PROGRAM-----*/
```

Output: -11

Note:

1's complement of a binary number obtained by changing 0s to 1s and vice versa.

2's compliments = 1's compliment + 1

NOTES

NOTES

Comma (,) Operator

The comma operator separates the expressions.

Syntax: `exp1, exp2, ... expn`

Where `exp1, exp2, ... expn` are expressions.

E.g.: `k= (i=5, j=10, i+j);`

Here, first the value 5 is assigned to `i`, then 10 is assigned to `j` and the result `i+j` is assigned to `k`.

Generally, the comma operator is used in `for` loops.

E.g.: `for(i=0, j=1; i<=10; i++, j++)`

Demo program using comma operator

```
#include<stdio.h>
void main()
{
    int a,b,c;
    a=(b=10,c=20,b+c,b-c);
    printf("%d",a);
}
```

Output: -10

Explanation: In the statement `a=(b=10,c=20,b+c,b-c);` first `b-c` will be calculated, i.e. `10 - 20 = -10` and it will be assigned to the variable `a`.

Address operator (&): This operator is useful in finding the address of a variable of any data type.

Indirection operator (*): This operator is used to find the value at a particular memory location. This operator is also called as de-referencing operator.

Note: If both address operator (&) and indirection operator (*) are used side by side (&* or *&), they do nothing to the variable. For example, If 'ptr' is a pointer variable, then `&*ptr` and `*&ptr` equal to `ptr`.

sizeof() operator: This operator is used to obtain the size of a variable, which occupies system's memory. The `sizeof ()` return value (in bytes) is the size of a variable or type within the parentheses.

Syntax: `sizeof(object);`
(Where object may be any data type, variable or expression.)

E.g.:

```
1. int n1;
   n1 = sizeof(int);
2. int n2;
   n2 = sizeof(n1);
```

Explanation:

In the first example, `sizeof(int)` returns 2, because the size of the `int` data type is 2.

In the second example, `n2` returns 2, because the size of the integer variable also 2.

2.5.2 Expressions

All valid combinations of variables, constants, operators and functions are termed as expressions. Expressions always return a result. They can be as simple as a single value and as complex as a large calculation.

Examples of C:

Algebraic Expression	equivalent C expression
$a + b$	$a + b$
$a \times b$	$a * b$
$(a + b)^2$	$(a + b) * (a+b)$
$x^2+xy+10$	$x*x+x*y+10$
$\frac{(p + q)}{(x + y + z)}$	$(p+q) / (x+y+z)$

Type Casting

Conversion of a value of one data type to another is called type casting.

- Explicit type casting
- Automatic type casting
- Mixed type casting

Explicit type casting

In explicit type casting, the compiler converts the value of an expression to a particular data type, which is given explicitly.

Syntax: (data type) expression;

E.g.: The expression (float) 3/2 evaluates to type float and gives output as 1.5. Without cast (float), only integer division would have been performed and the output would be 1 (ignores the fractional part).

Program based on casting

```
#include<stdio.h>
void main()
{
printf("%d", (int)5.0/2);
}
Output: 2
```

Explanation: In the preceding program, double value (5.0) has cast to integer value 5, so the integer division will take place and result of the expression (int)5.0/2 is 2.

NOTES

NOTES

Automatic type casting

If an expression contains different types of constants and variables, they are all converted to the same data type. The C compiler automatically converts all operands to the higher type of the existing operands.

Table 2.1 Different Conversion Tables

Type conversion in C for various data types with char data type:		
Type of operand 1	Type of operand 2	Resultant type
char	char	char
short int	char	short int
int	char	int
long int	char	long int
float	char	float
double	char	double
long double	char	long double

Type conversion in C for various data types with char data type:		
Type of operand 1	Type of operand 2	Resultant type
char	int	int
short int	int	int
int	int	int
long int	int	long int
float	int	float
double	int	double
long double	int	long double

Type conversion in C for various data types with char data type:		
Type of operand 1	Type of operand 2	Resultant type
char	long int	long int
short int	long int	long int
int	long int	long int
long int	long int	long int
float	long int	float
double	long int	double
long double	long int	long double

Type conversion in C for various data types with char data type:		
Type of operand 1	Type of operand 2	Resultant type
char	float	float
short int	float	float
int	float	float
long int	float	float
float	float	float
double	float	double
long double	float	long double

Type conversion in C for various data types with char data type:		
Type of operand 1	Type of operand 2	Resultant type
char	double	double
short int	double	double
int	double	double
long int	double	double
float	double	double
double	double	double
long double	double	long double

Type conversion in C for various data types with char data type:		
Type of operand 1	Type of operand 2	Resultant type
char	long double	long double
short int	long double	long double
int	long double	long double
long int	long double	long double
float	long double	long double
double	long double	long double
long double	long double	long double

E.g.: The expression `3 + 2.0` evaluates to type double and gives output as 5.0. Without cast (double), addition would have been performed casting the result to double automatically.

Demo program based on auto type casting

```
#include<stdio.h>
void main()
{ printf("%d", sizeof(3+2.0));
}
```

Output: 8

Explanation: In the preceding program, int value (3) is added to double value (2.0), which is type cast to double value (3.0). So the size of value (3.0) is 8. The C compiler automatically converts all operands to the higher type of the existing operands.

Mixed type casting

If an expression contains different types of constants and variables, some of them have external type casting, then C compiler automatically converts all operands to the higher type of the existing operands.

E.g.: The expression `3 + (int)2.0` evaluates to type `int` and gives output as 3.

Demo program based on mixed type casting

```
#include<stdio.h>
void main()
{ printf("%d", sizeof(3+(int)2.0f));
}
```

Output: 2

Explanation: In the preceding program, float value (2.0f) is type cast to int value (2.0). Then it is added to the int value (3) resulting int value (5). So the size is 2.

NOTES

2.6 INPUT/OUTPUT OPERATIONS

Input and output operations are very much concerned with interacting with programmers and so their form is highly dependent on the specific programming language used, and occasionally, even on the computer system itself.

Here, two new types of statement are defined to handle input and output. The read statement allows us to read given values into indicated variables and the write statement allows us to display results. Input may come from a terminal or from some other form of input device. Output may appear on a display screen or printed on paper.

The form of the read statement is as follows:

Read(input list)

The `input list` gives the names of the variables to which values are to be given as they are encountered in the input stream.

For example:

Read(A, B, C)

The next three values encountered would be given to the variables A, B, and C. The first to A, the second to B, and the third to C.

Suppose the actual values entered were 16, 3, 7, 21, 16, 0, 4, 8, 1. These could be either on one input line, or spread over several. Consider the following three read statements:

Read(A, B, C)

Read(D, E, F, G)

Read(X, Y)

The values are given to the variables A, B, C, D, E, F, G, X, Y one, by one in the order written. The process of giving these values to the indicated variables is like the assignment operation in two aspects. First, it is a destructive

NOTES

operation in the same sense that the assignment operation was a destructive operation. Whatever values, if any, that the variables had previously possessed are overwritten. Data of any type can be input; if the value in the input stream is of a different type from that of the input list, a conversion is required.

The output statement is like the input statement in format. It is possible to display the contents of any variable, the result of any expression or the value of any constant. The general form is as follows:

```
Write(output list)
```

The following sequence of statements illustrates the action of a write statement. Assume that all variables are real.

```
MARK1 73
MARK2 65
MARK3 94
MARK4 87
AVERAGE (MARK1 + MARK2 + MARK3 + MARK4) / 4
Write(AVERAGE)
```

The first four statements assign the indicated variables on four tests. The fifth statement computes the average grade. The write statement then displays this calculated result:

Preceding sequence of 79.75

Let us observe the case when we replace the write statement with

```
Write(MARK1, MARK2, MARK3, MARK4, AVERAGE)
```

The resulting output would be as follows:

```
73 65 94 87 79.75
```

Note that all numbers are written as real numbers. This is because the variables themselves are all real variables. Now, however, you have simply five numbers displayed. Suppose, instead, you provided the following output statements:

```
Write('Individual grades are', MARK1, MARK2, MARK3,
MARK4)
Write('Final average is', AVERAGE)
```

In this case, the output displayed would be as follows:

```
Individual grades are 73 65 94 87
Final average is 79.75
```

In terms of information content, the last form is definitely superior. This last example shows the use of two different kinds of items in an output list. You used variables such as MARK1 and AVERAGE. You have also used constants, in this case string constants, i.e., Final average is.

There can be many special features associated with input and output. Here, a very general approach has been provided.

Finally while designing the input portion of an algorithm, some consideration should be given to robustness of an algorithm. A robust algorithm is one, which

produces reasonable results no matter what input is supplied. The requirements of robustness are:

- (i) the program should check the validity of its input
- (ii) the program should take some appropriate action for unacceptable input.

In a batch environment, these actions may involve including proper messages in the output stream and ending execution. For example, in an interactive environment, a more proper action may include output of a message followed by a prompt for input of a corrected value. Checking for types of invalid data is relatively easy. Here, in this example, negative grades should be rejected. Most compilers automatically terminate execution with a standard message for some types of errors. These often include illegal characters when expecting numeric input and many more.

Input–Output Operations

In C, input and output operations can be classified as follows:

- Console Input/Output operations
- Disk Input/Output operations
- Port Input/Output operations

Console Input/Output Operations

These functions get input from the keyboard and gives output to the VDU. These functions can be further classified into two categories:

Formatted Console Input/Output functions

Unformatted Console Input/Output functions. All these functions are defined in standard input–output library and the header files `stdio.h` and `conio.h` includes all input/output functions.

Formatted Input/Output functions get the input from the keyboard and give the output to the VDU as per the specified requirement. The `scanf ()` and `printf ()` comes under the formatted input/output functions.

The `scanf ()` function allows the user to enter numerical values and string values. This function is described in the system file `stdio.h`.

Syntax for `scanf()`:

```
int scanf (<format string>, arguments)
```

Where

<format string> consists of format specifiers

Arguments represent address of variables and this function returns an integer value that shows the number of successful input readings.

Note: In `scanf ()` function, space(s) should not give after the last format specifier in the format string.

The `printf ()` function is the opposite of `scanf ()`. It writes data available in the variables to the screen and like `scanf ()`, it is also described in the system file called `stdio.h`.

NOTES

NOTES

Syntax for printf():

```
int printf(<format string>, arguments)
```

Where

<format string> may consists of format specifiers

Arguments represent variables, constants or addresses and this function returns an integer value that shows the number of characters printed out to the screen.

Unformatted Input/Output functions/macros get the input as a single character or multiple characters by using the keyboard and gives output of a single character or multiple characters to the VDU.

Function Name	Operation
int getch (void)	Reads a character form keyboard without echo and without pressing enter key it returns character value.
int getche (void)	Reads a character from keyboard with echo and without pressing enter key it returns character value.
int fgetc (stdio)	Reads characters from keyboard with echo and with pressing enter key it returns first character value.
int getc (stdin)	Reads characters from keyboard until enter key is pressed having echo and with pressing key it returns first character value. It is a macro version of fgetc() .
int getchar (void)	Reads characters from keyboard until enter key is pressed having echo and with pressing enter key it returns first character value. It is the macro form of getc() .
int putch (int)	Prints a character to the screen.
int fputc (int, stdout)	Prints a character to the screen.
int fputc (int, stdprn)	Prints a character to the printer.
int putc (int, stdout)	Prints a character to the screen. It is the macro version of fputc(stdout) .
int putc (int, stdprn)	Sends a character to the printer. It is the macro version of fputc(stdout) .
int putchar (int)	Prints a character to the screen. It is the macro form of putc(stdout) .

Note:

- **getch()**, **getche()**, **fputc()**, **putchar()** and **putc()** are defined in **<conio.h>**,
- **getchar()**, **getc()**, **fgetc()** and **putch** are defined in **<stdio.h>**
- **stdout** refers to standard output buffer.
- **stdin** refers to standard input buffer.

- `stdprn` refers to standard printer buffer.
- In reading a character, all functions and macros first check whether there are any characters in the console buffer. If any characters are present, it directly reads from the buffer without waiting for input. To clear the input buffer, use `fflush(stdin)`. Where `stdin` points input buffer and `fflush` clears the input buffer.

Unformatted I/O functions/macros for string type:

Function Name	Operation
<code>char * gets (char *string)</code>	Receives a string until a newline character (<code>\n</code>) is found (until enter key is pressed). It returns a pointer to the argument string.
<code>int puts (const char *s)</code>	Sends a string to the screen and appends a newline character. It returns the last character written. Otherwise, a value of EOF is returned.
<code>char *cgets (char *str)</code>	Reads a string from console. <code>str[0]</code> must contain the maximum length of the string to be read. On return, <code>str[1]</code> is set to the number of characters actually read. The string begins at <code>str[2]</code> . The function returns <code>&str[2]</code> .
<code>int cputs (const char *str)</code>	Writes a string to the text window on the screen. It returns the last character printed.

Note: `getch()` and `puts()` are defined in `<stdio.h>`.
`cgets()` and `cputs()` are defined in `<conio.h>`.

Check Your Progress

7. Define the term operator.
8. Give the definition of expressions.
9. State the functionality of the read statement and write statement.
10. What is a robust algorithm?

2.7 BRANCHING

Real-life application programs do not merely consist of simple multiplication or addition. They call for solving complex problems. Depending on the occurrence of a particular situation, we may follow different paths; the `if` and `else` keywords are quite handy in branching to different segments of the program. 'C' is ideal for handling branching because the syntax is clear and unambiguous. You will now read about the branching constructs. Relational operators are used in conjunction with branching constructs. Hence, you look at them first.

NOTES

2.7.1 if Statement

The syntax of the `if` statement is as follows:

```
if (condition)
    {statements}
```

NOTES

If the condition is true, then a single statement or group of statements following the `if` will be executed. If more than one statement is to be executed, then the statements are grouped within braces. If it is a single statement, then curly braces are not required.

If the condition turns out to be false, then the next statement after those belonging to the `if` will be executed. Example 2.1 will make the concept clear.

Input two integers from the keyboard. If they are equal, then the program will print, 'you typed equal numbers'; otherwise, it will print nothing.

/*Example 2.1

This program demonstrates the use of `if`*/

```
#include <stdio.h>
main()
{
    unsigned int a,b;
    printf ("enter two integers\n");
    scanf ("%u%u", &a, &b);
    if (a==b)
        {
            printf("you typed equal numbers\n");
        }
}
```

Each opening curly brace has to have a matching closing curly brace. In Example 2.1 the first closing curly brace corresponds to the `if` statement and the second one to the main function. Execute the program by first keying in two equal valued unsigned integers.

Result of the program

```
enter two integers
56 56
you typed equal numbers
```

After you are satisfied, you can try the program with unequal numbers. You will not get any message.

2.7.2 if...else Statement

You did not get any message when the numbers were unequal and this can be avoided by using the `else` statement.

The usage of `if .. else` is shown below.

```
if (condition true)
{
    statements s1
}
else
{
    statements s2
}
statements s3;
```

The statement `else` is always associated with an `if`.

If the condition is true, then statements `s1` will be executed. After executing them, the program will skip the `else` block and control goes to statement `s3` that follows the `else` block.

If the condition is false, then the statements in the `else` block, i.e., `s2` will be executed followed by statement `s3`.

Statements `s1` will not be executed at all when the condition becomes false. The usage of braces clearly brings out which statements belong to the `if` block and which to the `else` block.

Example 2.2 brings out the usage of `if... else`.

/*Example 2.2

This program demonstrates use of `if.. else*`

```
#include <stdio.h>
main()
{
    unsigned int a,b;
    printf ("enter two integers\n");
    scanf ("%u%u", &a, &b);
    if (a==b)
    {
        printf("you typed equal numbers\n");
    }
    else
    {
        printf("numbers not equal\n");
    }
}
```

The output of the program when unequal numbers were keyed in is as follows.

NOTES

NOTES

Result of the program

```
enter two integers
17 13
numbers not equal
```

2.7.3 Nesting of the `if . . . else` Statements

You witnessed the usage of a single `if` statement in Example 2.1. You saw `if` followed by `else` in Example 2.2. There is no restriction to the number of `if`, which can be used in a program. This applies to `else` as well, but `else` can only follow an `if` statement.

You can have the following in a program:

```
{
if (condition1)
{
    if (condition2)
        {statements-s1}
    else
        if (condition3)
            {statements-s2,}
}
else
    {statements-s4}
statements-s5
}
```

This is called a nested `if` and `else` statement. As the level of nesting increases, it will be difficult to analyse and logical mistakes will be made more easily.

In the above example, when `condition1` is false, `statements-s4` will be executed. If `condition1` is true and `condition2` is also true, then `statements-s1` will be executed.

If `condition1` is true and `condition2` and `condition3` are false, `statements-s5` will be executed directly.

To execute `statements-s2`

Condition1 has to be true;

Condition2 has to be false,

And condition3 has to be true.

Try to analyse this yourself. There are better methods to solve the above problem, which will be discussed later.

For example, three unequal integers are keyed in and are called `x`, `y` and `z`. Write a program to find the greatest of the three numbers.

Before writing a program, we must write the algorithm. We should not straight away get down to programming.

ALGORITHM 1

Algorithm for finding the greatest of 3 integers.

Step 1: Print a message to enter 3 integers.

Step 2: Get three numbers and store them at &x, &y and &z.

Step 3: Check if $x > y$

Step 4: If false, go to step 9

Step 5: If true

Step 6: Check if $x > z$

Step 7: If true, write x is the greatest; End

Step 8: If false, write z is the greatest; End

Step 9: Check if $y > z$

Step 10: If true, write y is the greatest; End

Step 11: If not, write z is the greatest.

Step 12: End

Now code these steps into a 'C' program, which is shown in Example 2.3.

/*Example 2.3

This program demonstrates the use of the nested if..else*/

```
#include <stdio.h>
main()
{
    int x,y,z;
    printf("enter three unequal integers\n");
    scanf("%d%d%d", &x, &y, &z);
    if(x>y)
    {
        if(x>z)
        {
            printf("x is greatest\n");
        }
        else
        {
            printf("z is greatest\n");
        }
    }
    else
    {
        if(y>z)
        {
            printf("y is greatest");
        }
    }
}
```

NOTES

NOTES

```
        else
        {
            printf("z is greatest");
        }
    }
}
```

Test the correctness of the program by giving a different set of values for x, y and z.

Result of the program

```
enter three unequal integers
908 231 907
x is greatest
```

Look at the example. It uses multiple nesting of `if .. else`.

Take care to see that every opening brace has a corresponding closing brace. It is better to indent the braces as shown in the example so that no mistake is committed. Take care to see that `else` matches with the corresponding `if` and each opening brace `{` matches with a corresponding closing brace `}`; if either an opening `{` or closing `}` is extra, then an error will result.

2.7.4 Logical Operators and Branching

In the above examples you have been checking one condition at a time. It would be nice if you could check more than one condition at a time. 'C' provides three logical operators for combining more than one condition. These are as follows:

Logical AND represented as `&&`

Logical OR represented as `||`

Negation or NOT represented as `!` (exclamation).

Take a look at some examples of usage of the logical operators. In Example 2.3 it was concluded that,

`if x > y and if x > z, then x is the greatest.`

You will represent the same as,

```
if ((x > y) && (x > z))
printf ("x is the greatest");
```

You will see that the program has become much more elegant.

The syntax for `&&` is,

```
if ((condition1) && (condition2))
{
    statements-s1
}
```

Statements-s1 will be executed only if both the conditions are true.

The syntax for 'or' is as follows:

```
if ((condition 1 ) || (condition 2))  
{  
  statements-s2  
}
```

In this case, even if one of the conditions is true, the statements-s2 will be executed. At least one condition has to be true for the execution of s2. However, if both are false, s2 will not be executed.

The NOT operator with symbol ! can be used along with any other relational or logical operator or as a stand-alone operator. It simply negates the operator following it. The syntax of '!' is as follows:

```
if ! (condition) statement s3;
```

s3 will be executed only when the condition is not true or the condition is false.

Now rewrite Algorithm 1 by using the logical operators. The revised Algorithm 2 is shown here:

ALGORITHM 2

Step 3: If $(x > y)$ and $(x > z)$, x is the greatest.

Step 4: Else if $(x < y)$ and $(y > z)$, y is the greatest.

Step 5: Else print z is the greatest.

The complete program is given in Example 2.4.

/*Example 2.4

This Example demonstrates the use of logical operators*/

```
#include <stdio.h>  
main()  
{  
  int x,y,z;  
  printf ("enter three unequal integers\n");  
  scanf ("%d%d%d", &x, &y, &z);  
  if ((x>y) && (x>z))  
    printf ("x is greatest\n");  
  else  
  {  
    if((x<y) && (y>z))  
      printf("y is greatest\n");  
    else  
      printf("z is greatest\n");  
  }  
}
```

NOTES

NOTES

Result of the program

```
enter three unequal integers
12 23 78
z is greatest
```

Now write a program to convert a lower case letter typed into an upper case letter. For this purpose you may have to refer to the ASCII table in Annexure 1.

It is obvious that if you subtract 32 from the ASCII value of a lower case alphabet, you will get the ASCII value of the corresponding upper case letter. Now write an algorithm for the conversion of lower case to an upper case letter. It is given in Algorithm 3.

ALGORITHM 3

Step 1: Send a message for getting a character

Step 2: Get a character

Step 3: Check whether the character typed is $\geq a$ and $\leq z$

(This is essential since you can only convert a lower case alphabet into upper case.)

Step 4: If so, subtract 32 from the value of the character; if not, go to step 6

Step 5: Output the character with the revised ASCII value; END

Step 6: Print 'an invalid character' END

The algorithm is implemented in Example 2.5.

/*Example 2.5

Conversion of lower case letter to upper case*/

```
#include <stdio.h>
```

```
main()
```

```
{
```

```
    char alpha;
```

```
    printf ("enter lower case alphabet\n");
```

```
    alpha=getchar();
```

```
    if (( alpha >=' a' ) && ( alpha <=' z' ))
```

```
    {
```

```
        alpha= (alpha-32);
```

```
        putchar (alpha);
```

```
    }
```

```
    else
```

```
        printf("invalid entry; retry");
```

```
}
```

Now you can test the program by giving both the valid and invalid inputs; valid inputs are the lower case letters and invalid inputs are all other characters.

Result of the program

The result for the invalid input is as follows:

```
enter lower case alphabet
8
invalid entry; retry
```

The result when tried with a valid input is given below:

```
enter lower case alphabet
n
N
```

The programs should be executed, i.e., tested with both the valid and invalid inputs.

2.7.5 Conditional Operator and `if...else`

The syntax for the conditional operator is as shown here:

```
(Condition)? statement1: statement2;
```

What does it mean? If the condition is true, execute `statement1`; else, execute `statement2`. Here nesting is not possible. The `if...else` statement is more readable than the conditional (`?`) operator. However, the conditional operator is quite handy in simple situations as follows:

```
(a > b) ? print a greater
        : print b greater;
```

Thus, the operator has to be used in simple situations. If nothing is written in the position corresponding to `else`, then it means nothing is to be done when the condition is false.

Example 2.2 is rewritten using the `?` operator in Example 2.6.

/*Example 2.6

This Example demonstrates use of the `?` operator*/

```
#include <stdio.h>
main()
{
    unsigned int a,b;
    printf ("enter two integers\n");
    scanf ("%u%u", &a, &b);
    (a==b)?printf("you typed equal numbers\n"):
    printf("numbers not equal\n");
}
```

Result of the program

```
enter two integers
123 456
numbers not equal
```

NOTES

NOTES

2.8 LOOPS AND CONTROL CONSTRUCTS

Quite often, you have to perform the same operation a number of times. You may also have to repeat the same operation with one or more of the values changed, which is known as loop or iteration. It is definitely possible to write a program for such situations with the concepts you have learned so far. However, there are special constructs in any programming language for carrying out repeated calculations.

2.8.1 Iteration using **if**

Before you look at loop constructs, let us consider an example to see the need for repetitive calculations. Assume that you want to find the sum of the first 10 natural numbers 1 to 10. This can be achieved through successive addition, i.e., first you initialize the sum to 0 and then add 1 to the sum. Next you add 2 to the sum, then 3, and so on till you add 10 to the sum. Thus, by repeated addition 10 times, you have found the sum of first 10 natural numbers.

The algorithm below summarizes what you have done:

Step 1: Sum = 0

Step 2: I = 1

Step 3: If $I \leq 10$ perform the following operations:

$sum = sum + I; I = I + 1;$

Step 4: Print the sum

Now analyse the algorithm:

At the beginning, steps 1 and 2 are entered with $sum = 0$ and $I = 1$

since $I \leq 10$

Sum will be equal to $sum + I$,

i.e., $sum = 0 + 1 = 1$

$I = I + 1$, i.e., $I = 2$

Now the program goes to step 3.

with $I = 2$ and $sum = 1$

Since $I \leq 10$

$sum = sum + I$

sum was 1 and I is 2

$sum = 1 + 2 = 3$

Next I will be incremented to 3

Third iteration:

Step 3 is approached with $I = 3$ and $sum = 3$

since $I \leq 10$

$sum = sum + I = (1 + 2) + 3$

$I = 4$

Ninth iteration:

Step 3 is approached with $I = 9$

since $I \leq 10$

$sum = (1 + 2 + 3 + \dots + 8) + 9$

I is incremented to 10

Tenth iteration:

Step 3 is approached with $I = 10$

since $I \leq 10$

$sum = sum + I$

$= (1 + 2 + 3 + \dots + 8 + 9) + 10$

Now I is incremented to 11

since $I \leq 10$ is not true, the program does not execute the statements following the `if` and jumps to step 4.

In step 4 the sum is printed.

This algorithm is implemented in program 2.7.

/*Example 2.7

Demonstrates use of `if` for iteration*/

```
#include <stdio.h>
```

```
main()
```

```
{
```

```
    int sum=0, i=1; /*declaration and initialization  
    combined*/
```

```
    step3:      /*label- loop starts here*/
```

```
    if (i <=10)
```

```
    {
```

```
        Sum = sum + i;
```

```
        i = i + 1;
```

```
        goto step3;
```

```
    }
```

```
    printf("sum of first 10 natural numbers=%d\n", sum);
```

```
}
```

Result of the program

```
sum of first 10 natural numbers = 55
```

The program has implemented the algorithm in `goto`. The program uses `if` and `goto` keywords. According to the algorithm, the program has to go to step3. Step3 in this program is called a label, which is followed by a colon. The rules for coining a label name are the same as for an identifier. The label can be placed anywhere in the same function where the `goto` statement is found. Usage of `goto` is considered to be bad programming practice since it leads to errors when changes are made in the program and also affects readability. It is always possible to write a program without using `goto`. The program can be rewritten without `goto` by using a `for` statement.

NOTES

2.8.2 for Statement

The `for` statement is meant for the easy implementation of iterations unlike `if`. The syntax of `for` is given ahead:

```
for (exp1; exp2; exp3)
{statements;}
```

NOTES

Note the keyword, the parentheses and semicolons. There is no semicolon after `exp3`, `exp1`, `exp2` and `exp3` are expressions. The usage of the `for` loop is given below:

`exp1`– Contains the initial value of an index or a variable.

`exp3`– Contains the alteration to the index after each iteration of the body of the `for` statement. The body of the statement is either a single statement or a group of statements enclosed within braces.

If a single statement has to be executed, then braces are not required.

`exp2`– Condition that must be satisfied if the body of statements is to be executed.

An example of a `for` loop is given below:

```
for (i = 0; i < 5; i++)
printf("%d", i);
```

The loop will start with an initial value of `i = 0`. Since `i < 5`, the body of the `for` loop will be executed and it will print 0. Now the `exp3` will be executed and `i` will be incremented to 1. Since `i` is less than 5, body of the loop will again be executed to print 1. This will continue till 4 is printed. `i` will now be incremented to 5 and since `i` is not less than 5, the `for` loop will be terminated. This is how `for` is used to carry out repetitive operations.

Now write a program for finding the sum of the first 10 natural numbers using the `for` statement.

The program is given in Example 2.8

```
/*Example 2.8
demonstrates the use of the for statement to find the sum
of the first 10 natural numbers*/
#include <stdio.h>
main()
{
    int sum=0, i; /*declaration and initialization
combined*/
    for (i=1; i<=10; i++) /*loop starts here*/
    {
        Sum = sum + i;
    }
    printf("sum of the first 10 natural numbers=%d\n",
sum);
}
```


Result of the program

sum of first 10 natural numbers = 55

Note the difference between Example 2.7 and Example 2.8.

You have eliminated the label Step 3 and the goto statement.

The initialization of $i = 1$ is carried out as part of the for statement.

The incrementing of i is also carried out as part of the for statement. The program has, therefore, been simplified.

How does the program work?

Step 1: $i = 1$

i is checked with $i \leq 10$

Since i is less than 10, the for loop is executed.

$sum = sum + i = 0 + 1 = 1$

Step 2: i is incremented to 2

2 is ≤ 10

Therefore, the for loop is executed.

$sum = sum + i = (1) + 2$

Step 9:

i is incremented to 9

9 is ≤ 10

Therefore, the for loop is executed.

$sum = sum + i = (1 + 2 + \dots + 8) + 9$

Step 10:

i is incremented to 10

10 is ≤ 10

$sum = sum + i = (1 + 2 + \dots + 9) + 10$

Step 11:

i is incremented to 11.

11 is not ≤ 10 .

Therefore, the for loop is now terminated.

The printf() function is now executed automatically.

Now summarize the operation of the for loop.

When a program encounters a for loop, it first checks the condition through the expression in the middle. If the condition is satisfied, it executes the group of statements. After executing the statements in the body of the loop, the program transfers the execution to the for statement and the third expression is executed, which is usually incrementing or decrementing. Then the condition is checked. If the condition is not satisfied, the group of statements will not be executed and the program will skip to the next statement after the statements pertaining to the for statement.

NOTES

By chance if the initial value was typed as 11 instead of 1 in the program, the condition will turn out to be false and the group of statements will not be executed at all.

NOTES

Three Components of `for`

The three components of a `for` statement are as follows:

`exp1` and `exp3` are assignments or function calls. Function calls will be discussed at a later stage; `exp2` is a relational expression. The three expressions may not always be present. However, even if an expression is not present, the associated semicolon should be present.

For example,

```
for (; exp2 ;)  
{s1}
```

Here, the initial value is not specified and the incrementing does not take place after every iteration. Presumably, the initial value is assigned elsewhere and incrementing or a similar operation takes place as part of the group of statements following the `for`. However, since `exp2` is present, the loop will terminate.

However, if all three expressions are omitted as follows:

```
for ( ; ; )
```

the loop will never terminate because the conditional statement is absent. If `exp2` is not present, it is assumed that the condition is true always. Such a statement should not be used.

Instead of incrementing, you can use `i += 2` as `exp3` when `i` will be incremented by 2 every time.

Now try to print the list of even numbers up to 50. The program is as shown:

```
/*Example 2.9  
variation in for statement - to print even numbers*/  
#include <stdio.h>  
main()  
{  
    int i=2;  
    for (; i<10; i+=2)      /*loop starts here*/  
    {  
        printf("%i is an even number\n",i);  
    }  
}
```

Here you initialize `i = 2` before the `for` loop itself. However, the corresponding semicolon is present at the right place.

Result of the program

```
2 is an even number
4 is an even number
6 is an even number
8 is an even number
```

2.8.3 Symbolic Constants and Looping

So far you have been giving the initial value, the increment and final value as part of the programs. Assuming you want to change one or more of them later on, how are you to go about it? You would then have to painfully rewrite the program. C provides a method by which this can be done with the least changes by using the # define statement. The format of this statement is as follows:

```
# define name constant
```

For example, you can define,

```
# define INITIAL 1
```

Which defines INITIAL as 1. The INITIAL types of definitions are called symbolic constants. They are not variables and hence, they are not defined as part of the declarations of variables. They are specified on top of the program before the main(). The symbolic constants are to be written in capital or upper case letters. Wherever the symbolic constant names appear in a program, the compiler will replace them with the corresponding replacement constants defined in the # define statement. In this case, 1 will be substituted wherever INITIAL appears in the program. Note that there is no semicolon at the end of the # define statement.

You can now write a program to print out the numbers between a given range, say 100 to 150, which are divisible by 3, i.e., when divided by 3 the modulus = 0. Such numbers are 'evenly divisible by 3'.

/*Example 2.10

```
Demonstrates the use of symbolic constants-
program to find numbers between 100 and 150
evenly divisible by 3*/
#include <stdio.h>
#define LOW 100
#define UPPER 125
#define STEP 1
main()
{
    int num;
    for (num=LOW; num<UPPER; num+=STEP) /*loop starts
here*/
    {
        if(num%3 == 0)
            printf("%i is evenly divisible by 3\n", num);
    }
}
```

NOTES

NOTES

Result of the program

```
102 is evenly divisible by 3
105 is evenly divisible by 3
108 is evenly divisible by 3
111 is evenly divisible by 3
114 is evenly divisible by 3
117 is evenly divisible by 3
120 is evenly divisible by 3
123 is evenly divisible by 3
```

You can use this technique in future programs. Assume that you want to find out all the numbers evenly divisible by 3 between 1 and 1000.

You have to define LOWER as 1 and UPPER as 1000. Assuming that later on you want to find out the numbers evenly divisible by 7, you would have to again rewrite the program. This can be avoided by defining the DIVISOR as 7 and substituting within the condition (`number % DIVISOR == 0`). In this way, you can write a program to find out the numbers evenly divisible by any number in any range.

2.8.4 Other Forms of the `for` Loop

The `for` loops can be nested as follows:

```
for (i = 1; i <= 10; i++)
{
    for (j = 1; j <= 5; j++)
    {
        for (k = 1; k <= 2; k++)
        {
            s1
        }
    }
}
```

The statement `s1` will be executed as follows:

First time	<code>i = 1,</code>	<code>j = 1,</code>	<code>k = 1</code>
Second time	<code>i = 1,</code>	<code>j = 1,</code>	<code>k = 2</code>
Third time	<code>i = 1,</code>	<code>j = 2,</code>	<code>k = 1</code>
	<code>i = 1,</code>	<code>j = 2,</code>	<code>k = 2</code>
	<code>i = 1,</code>	<code>j = 3,</code>	<code>k = 1</code>
	<code>i = 1,</code>	<code>j = 3,</code>	<code>k = 2</code>
Lastly	<code>i = 10,</code>	<code>j = 5,</code>	<code>k = 2</code>

`s1` will be executed $2*5*10 = 100$ times.

Any level of nesting is acceptable; However, the higher the level of nesting is, the more easy it will be to commit mistakes and more difficult to understand.

Now, look at some more `for` loops.

```
For ( x = -5;   - -x >= -10; )  
{  
}
```

Here, the decrement and conditional statements are combined in `exp2`.

Since decrement is a prefix, the decrement of `x` is carried out first. The condition is then checked in order to decide whether to continue or not. Then the loop is executed. Therefore, the first iteration will be carried out with `x = -6` and the last with `x = -10`.

Another variation of the statement is as follows:

```
for (y = 100;   y ++<= 200;)  
{  
    s2  
}
```

Here too `exp2` and `exp3` are combined. This is a postfix notation. The following sequence is carried out: condition check, increment and execute the loop. Therefore, the statement `s2` following the `for` loop will be executed the first time with `y = 101` and finally with `y = 201` as well.

The `for` loop is a popular iteration construct not only in 'C' but also in other languages. Here, the initial value, the step and the final value are clear and unambiguous and simple to write. There are other loop statements also. In the next section, you will study the `while` loop.

2.8.5 The while Loop

The `while` loop is a subset of the `for` loop. The syntax for the `while` loop is given below:

```
while (expression)  
{statements;}
```

This means that the statement(s), which is a single statement or multiple statements, will be executed while the expression is true. When it becomes false, the execution will stop.

The `while` is similar to the `for` loop without `exp1` and `exp3`. The `for` loop can be simulated or replaced with the `while` loop as given below.

```
exp1;  
while (exp2)  
{  
    statements  
    exp3;  
}
```

The programmer can use `while` or `for` at his discretion. The `for` loop is preferred when the initialization and incrementation are simpler.

Let us look at an example.

Let us write a program for the generation of any multiplication table.

NOTES

NOTES

The program is given below:

/*Example 2.11

Use of while - You can generate multiplication tables of your choice using this program.

caution: Don't exceed maximum limits of integer */

```
#include <stdio.h>
main()
{
    int a,b,product;
    a=1;
    b=0;
    product=0;
    printf("Enter which table you want");
    scanf("%d",&b);
    while (a <=10)
    {
        product = a*b;
        printf("%2d X %d= %3d\n",a,b,product);
        a++;
    }
}
```

When the program asks you to enter a table and you type 12, you will get the 12th table as given below.

Result of the program

```
Enter which table you want 12
1 X 12= 12
2 X 12= 24
3 X 12= 36
4 X 12= 48
5 X 12= 60
6 X 12= 72
7 X 12= 84
8 X 12= 96
9 X 12= 108
10 X 12= 120
```

Note here that the condition is (a < =10); incrementing is done within the loop in a++. Variable a is initialized as 1 before entering the while loop.

If you want to print the table up to 16 × 12 = 192 then simply change the condition to while(a < =16). Simple, isn't it?

2.8.6 do . . . while Loop

This is a modification of the `while` statement. In the `while` statement, before the group of statements following the `while` are executed, the condition associated with the `while` is checked. If the condition is true or fulfilled, then the associated statements are executed. If not, the program skips the statements associated with the `while` loop. This is depicted in Figure 2.2.

NOTES

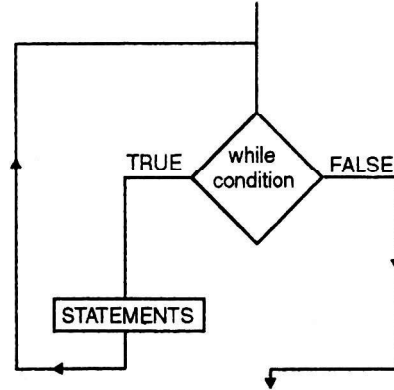


Fig. 2.2 while Loop

After execution of the statements, the program will check again whether the condition is true and then continue to execute or skip the statements depending on the condition. The statements may not be executed even once if the condition was false at the entry point.

However, the `do . . . while` works differently as shown in Figure 2.3.

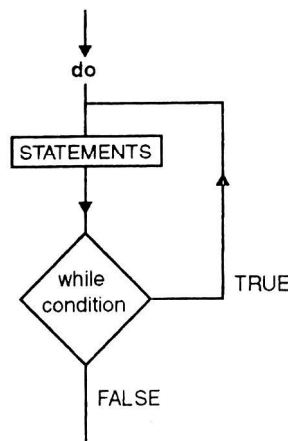


Fig. 2.3 do...while Loop

Here the statements following `do` will be executed once before the condition is checked. If it is true, then the statements will be executed again. If not, the program will skip the statements and proceed further. Thus, whatever be the condition, the statements following `do` will be executed once before checking the condition. This is the essential difference between `do . . . while` and `while`. The `while` loop tests the condition on top of the loop; but `do . . . while` tests at the bottom after executing the statements. The `while` loop executes the

NOTES

statements after checking the condition; but `do . . . while` executes the statements before testing the condition. The syntax of `do . . . while` is as follows:

```
do
{
    statements
}
while (expression);
```

The statement `do . . . while` is not used as frequently as the `while` loop.

Example 2.12 converted upper case alphabets to lower case. In case you want to convert more alphabets you will have to execute the program again and again. This can be avoided by the `while` loop. The program will continue to run as long as you want to convert more and more alphabets. The program has been rewritten with `while` for converting upper case to lower case. You can convert as long as you want. When you want to stop, enter 1. The program, which uses `while` for converting an upper case character to a lower case is given below:

/*Example 2.12

Conversion of upper case to lower case alphabet*/

```
#include <stdio.h>
#include <conio.h>
main()
{
    int alpha=0;
    while (alpha!='1')
    {
        printf ("\nenter upper case alphabet- enter 1 to
quit\n");
        alpha=getche();
        if ( alpha >='A' && alpha <='Z' )
        {
            alpha= (alpha+32);
            putchar(alpha);
        }
        else
        {
            if(alpha!='1')
                printf("\ninvalid entry; retry");
            else
                printf("End of session");
        }
    }
}
```


Result of the program

```
enter upper case alphabet- enter 1 to quit
Pp
enter upper case alphabet- enter 1 to quit
p
invalid entry; retry
enter upper case alphabet- enter 1 to quit
Qq
enter upper case alphabet- enter 1 to quit
1End of session
```

The above program works till 1 is pressed. It continues to convert upper case to lower case till 1 is pressed. The program has been designed in such a manner that it will perform at least one iteration of the statements following while. This can be rewritten using do . . . while. The rewritten program is as follows:

/*Example 2.13

Conversion of upper case to a lower case alphabet*/

```
#include <stdio.h>
#include<conio.h>
main()
{
    int alpha=0;
    do
    {
        printf ("\nenter upper case alphabet- enter 1 to
quit\n");
        alpha=getche();
        if ( alpha >='A' && alpha<='Z' )
        {
            alpha=(alpha+32);
            putchar(alpha);
        }
        else
        {
            if(alpha=='1')
                printf("End of Session");
            else
                printf("\ninvalid entry; retry");
        }
    }while(alpha!='1');
}
```

Result of the program

```
enter upper case alphabet- enter 1 to quit
Gg
```

NOTES

NOTES

```
enter upper case alphabet- enter 1 to quit
o
invalid entry; retry
enter upper case alphabet- enter 1 to quit
Dd
enter upper case alphabet- enter 1 to quit
lEnd of Session
```

How does it differ? Here too, the program will attempt to convert one character before it can be terminated. Assuming that the first character was 1, the program will still attempt to convert it and print the message "End of Session" before it quits.

Suppose the first character is a valid one and a number of characters are converted in succession; when you want to terminate the program, 1 has to be pressed and even then the program will not stop immediately. It will stop only after the statements are executed. Since the problem is the same, a detailed look at both the examples will bring out the similarity in operation between both the constructs. However, there are occasions when it is quite suitable as given in the next section.

2.8.7 switch Statement

switch statements allow clear and easy implementation of multiway decision-making. Assuming that a number is received from the keyboard and depending on the value, we want to carry out some operations, the switch statement can be used effectively in this situation. In simpler situations `if...else` could be used, and in complex situations, switch can be used. For example, if you get numbers starting from 1 to 4 and print their values in words, you can use the `if...else` statement as given in the program 2.14 below:

/*Example 2.14

Converts the digits 1-4 in words using `if*/`

```
#include <stdio.h>
#include <conio.h>
main()
{
    int a;
    char ch = 'c';
    while (ch=='c')
    {
        printf("\nEnter a digit 1 to 4\n");
        scanf("%d",&a);
        if(a== 1)
            printf("One\n");
        else
            if (a== 2)
                printf("Two\n");
            else
                if(a== 3)
```

```
        printf("Three\n");
    else
        if(a==4)
            printf("Four\n");
        else
            printf("Illegal character\n");
            printf("enter 'c' if you want to continue\n");
            printf("or any other character to end\n");
                ch=getche();
                if (ch!='c')
                    printf("End    of
Session");
            }
    }
```

Result of the program

```
Enter a digit 1 to 4
3
Three
enter 'c' if you want to continue
or any other character to end
c
Enter a digit 1 to 4
1
One
enter 'c' if you want to continue
or any other character to end
c
Enter a digit 1 to 4
2
Two
enter 'c' if you want to continue
or any other character to end
nEnd of Session
```

You have struggled hard to do this exercise. Assuming that you want to get a one digit number up to 9 and print its value in words, it would become a more complex task. The `switch` statement comes in handy in such situations. The syntax of the `switch` statement is as follows:

```
switch (expression)
{
    case    constant or expression : statements
    case    constant or expression : statements
    ..
default : statements }
```

NOTES

NOTES

When the **switch** keyword is encountered, the associated expression is evaluated. The program now looks for the **case**, which matches with the value of the expression. Execution then starts from the statement corresponding to the **case** which matches. Each **case** has to be accompanied by integer expressions, which must be unique, as otherwise the program will not know where to start. For example, if the first two cases are as follows:

```
case 10 : s1;
case 10 : s2;
```

In this case, the program would not know whether to execute `s1` or `s2` when the expression of `switch` evaluates to 10. Therefore, the constant expressions following the `case` keyword should all be unique. There may be occasions when none of the constant expressions matches the `switch` expression in which case the `default` statements will be executed. Thus, `switch` allows branching of the program execution to an appropriate place. A program to print the values of the digits in words is given below:

/*Example 2.15

Converts the digits 0-9 in words*/

```
#include <stdio.h>
#include <conio.h>
main()
{
    int a;
    char ch = 'c';
    while (ch=='c')
    {
        printf("\nEnter a digit 0 to 9\n");
        scanf("%d", &a);
        switch(a)
        {
            case 0:printf("Zero\n");
                break;
            case 1:printf("One\n");
                break;
            case 2:printf("Two\n");
                break;
            case 3:printf("Three\n");
                break;
            case 4:printf("Four\n");
                break;
            case 5:printf("Five\n");
                break;
            case 6:printf("Six\n");
                break;
```

```
        case 7:printf("Seven\n");
            break;
        case 8:printf("Eight\n");
            break;
        case 9:printf("Nine\n");
            break;
        default:printf("Illegal character\n");
    }
    printf("enter 'c' if you want to continue\n");
    printf("or any other character to end\n");
    ch=getche();
    if (ch!='c')
        printf("End of Session");
}
}
```

NOTES

Result of the program

```
Enter a digit 0 to 9
6
Six
enter 'c' if you want to continue
or any other character to end
c
Enter a digit 0 to 9
7
Seven
enter 'c' if you want to continue
or any other character to end
nEnd of Session
```

We have used the `do...while` concept in this example also. The program first enters the `do` loop. After the execution of the loop, the program asks you to enter `c` if you want to continue or any other character to end the program. If you enter `c`, the program will continue. Thus, if you want to exit you can press any other letter, say `'n'`. If you press `'n'` the program stops instantly. This is the right way of using the `do...while` loop.

Now the `switch` is analysed. The program asks for entry of a digit 0 to 9. The entered digit is stored in variable `a`. If `a = 5`, then the program goes to case 5. It is followed by printing the value Five and then `break`. If you press 8, then case 8 matches and Eight will be printed.

What is a `break` statement ?

Assume that all the `break` statements are removed from the program. Then if you press 1, 'One' will be printed and all the statements following that will be executed. This means that Two, Three, ... till Nine will be printed. If you enter

NOTES

7, it will print all the numbers starting from Seven, which is not desirable. The `break` statement has, therefore, been introduced. After printing the value of the number, the `break` statement takes the program to the end of the `switch` statement. The end is just the closing brace corresponding to `switch` after the default `printf()` statement. Therefore, the combination of `switch` and `break` does the trick.

Assuming that a character other than 0 to 9 is entered, none of the cases match. Therefore, default is executed. The program will print 'illegal character'. The **default** is optional and even in its absence, when no match is found, the program will come out of the `switch` statement without any action.

The `case` statements need not be in any specified order. The default can be on top before `case 0` and the `case` can occur in any order. The program is made to execute, as long as you want, by the `do` statement. If `do` is absent, then the program will be executed only once. The `do...while` is necessary to make it an iterative program.

Every `switch` statement, therefore, contains a condition in the form of an expression. The expression could also be a single variable as in this case. The expression will be evaluated at the time of program execution and must be an integer. Then depending on the value it goes to a `case` label, which is like a label in a `goto` statement. The label should match with the value of the expression.

Unless the program is made to exit by statements such as `break` after executing the group of statements corresponding to a particular case, the program will execute all the statements in the program from then on. This should be noted. Therefore, the programmer has to specify where to end. In the case of `if...else`, where to begin and where to end is clear as also in the case of `switch`. The program starts at the beginning of the `case` that meets the condition, but ends at the bottom of the `switch` unless otherwise specified. It will also execute the statements following default. It will of course not bother about the keywords. That is the reason for the `break` statement, since we do not want the program to execute irrelevant statements.

It is a good programming practice to include a `break` statement after the default as well. If this is not done at the inception, at a later stage when more `case` statements are added after default, this would lead to problems. Whenever default is executed all the statements following it, even if they belong to some other case, will also be executed if `break` is not included after default.

Now the program can be extended to print the value of the number up to 99 in words. This makes it more complicated since the words are unique up to nineteen. A program is given below for achieving the task. This is made to loop using `do...while`.

/*Example 2.16

```
To print out in words the value  
of a number typed in the range 1-99*/  
#include <stdio.h>  
#include <conio.h>  
main()
```

```
{
  int num,m,ch='y';
  do
  {
    printf("Type a number 1 to 99\n");
    scanf("%d",&num);
    if ((num >0) && (num <100))
      /*only if the number is within the valid range 0 to
      99 the following will be executed*/
      {
        if (num >= 20)
        {
          m=num/10;
          switch(m)
          {
            case 2:printf("TWENTY ");
                    break;
            case 3:printf("THIRTY ");
                    break;
            case 4:printf("FORTY ");
                    break;
            case 5:printf("FIFTY ");
                    break;
            case 6:printf("SIXTY ");
                    break;
            case 7:printf("SEVENTY ");
                    break;
            case 8:printf("EIGHTY ");
                    break;
            case 9:printf("NINETY ");
                    break;
          }
        }
      }
    if (num >20)
      num= num%10;
    switch(num)
    {
      case 1:printf("ONE\n");
              break;
      case 2:printf("TWO\n");
              break;
      case 3:printf("THREE\n");
              break;
    }
  }
}
```

NOTES

NOTES

```
case 4:printf("FOUR\n");
    break;
case 5:printf("FIVE\n");
    break;
case 6:printf("SIX\n");
    break;
case 7:printf("SEVEN\n");
    break;
case 8:printf("EIGHT\n");
    break;
case 9:printf("NINE\n");
    break;
case 10:printf("TEN\n");
    break;
case 11:printf("ELEVEN\n");
    break;
case 12:printf("TWELVE\n");
    break;
case 13:printf("THIRTEEN\n");
    break;
case 14:printf("FOURTEEN\n");
    break;
case 15:printf("FIFTEEN\n");
    break;
case 16:printf("SIXTEEN\n");
    break;
case 17:printf("SEVENTEEN\n");
    break;
case 18:printf("EIGHTEEN\n");
    break;
case 19:printf("NINETEEN\n");
    break;
    }
}
else
    printf("number outside range\n");
    printf("\nenter y if you want to continue\n");
    ch=getche();
if (ch!='y')
    printf("End of session");
}
while (ch=='y');
}
```


Result of the program

```
Type a number 1 to 99
123
number outside range
enter y if you want to continue
yType a number 1 to 99
78
SEVENTY EIGHT
enter y if you want to continue
yType a number 1 to 99
6
SIX
enter y if you want to continue
nEnd of session
```

NOTES

2.8.8 Break, Continue, Return and goto

The keywords `while`, `for` and `switch` test the condition on top, while `do...while` checks at the bottom for quitting the loop. The `break` statement helps immediate exit from any part of the loop as demonstrated with the `switch` statement. It can be used with any other loop construct or anywhere in the program. When the `break` statement is executed it goes to the bottom of the block. Recall that a block is a group of statements enclosed between an opening brace and the corresponding closing brace.

The `continue` statement is related to `break`. When `continue` is executed, it causes the next iteration of the corresponding `for`, `do...while` or `while` loop to begin. Therefore, `continue` takes the program to the top of the block and in the `for` loop, it will cause the next increment operation, followed by checking whether the condition is true or false in order to decide the next course of action. This is similar to skipping the current execution and continuing with the next operation after incrementing. The statement `continue` skips the rest of the statements in the loop for that iteration, whereas `break` terminates the loop.

Write a program to check whether a given number is positive or negative. If it is zero, the program should terminate after printing the value. If it is a positive integer above zero and ≤ 20000 , the value will be printed; if negative, it will go to fetch the next number. If the number is > 20000 , the program terminates. The program is given below:

```
/*Example 2.17
/*Program to demonstrate continue*/
#include <stdio.h>
main()
{
    int a;
    do
    {
```

NOTES

```
printf("enter a number-enter 0 to end session\n");
scanf("%d", &a);
if(a > 20000)
{
    printf("you entered a high value-going out of
range\n");
    break;
}
else
if(a>=0)
    printf("you entered %d\n", a);
if (a <0)
{
    printf("you entered a negative number\n");
    continue;
}
}
while(a !=0);
printf(" End of session\n");
}
```

Result of the program

```
enter a number-enter 0 to end session
33
you entered 33
enter a number-enter 0 to end session
-60
you entered a negative number
enter a number-enter 0 to end session
45
you entered 45
enter a number-enter 0 to end session
25000
you entered a high value-going out of range
End of session
```

If the number typed > 20000 , or if it is equal to zero, the program comes out of the loop and prints "End of session". If the number is negative, $a < 0$ and hence, `continue` will be executed. It will go to the top of the loop. The next integer will be received. The program, therefore, terminates when $a = 0$ as well as $a > 20000$, but there is a difference. If the number entered is zero, the program checks whether $a > 20000$. Since the condition fails, it checks whether $a > = 0$ and since it is true, 0 will be printed and then the `while` condition is checked. The program terminates after the `while` condition is checked.

However, if the number entered is > 20000 , the loop terminates instantly without transacting any business except printing messages as shown.

The `return` statement can appear anywhere in a function and when it is encountered a value is returned to the called function. The `return` may also not return a value in statements as shown:

```
return ;  
return (0) ;
```

The `return` statement may appear anywhere in the function and not necessarily at the end of the function. Whenever `return` is executed, the program returns to the function called the current function. The program returns to the place from where it called the function. Thus `return` is also used to suddenly exit from a function or a loop in a function.

Exit Function

There is a library function `exit()`, which causes the termination of the current program. Note that, `exit()` terminates the execution of the program itself, and not the block. The statement `break` enables coming out of the block or loop in which it is executed but `exit` terminates the program at whatever stage the program may be. The `exit` is a powerful function.

goto Statement

The `goto` statement in C refers to structured programming principles. This statement leads to 'spaghetti' code, which is difficult to understand. The syntax of code is written as follows:

```
Goto Syscrash  
//Other statement  
Syscrash:  
//Control will begin here following goto
```

The label is always terminated by a colon. The `goto` statement is a jump statement, which jumps from one point to another point within a function. This keyword is marked by label statement. Label statement can be used anywhere in the function above or below the `goto` statement. The following C code displays the list of numbers from 0 to 9. For this, you need to define the label statement `loop` above the `goto` statement. The given program declares a variable `n` initialized to 0. The `n++` increments the value of `n` by 1 till the loop reaches 10. Then on declaring the `goto` statement, it will jump to the label statement and prints the value of `n`. The code is written in the C language as follows:

```
#include <stdio.h>  
#include <conio.h>  
int main()  
{  
clrscr(); //Clear the screen  
int n = 0;  
loop: ;  
printf("\n%d", n);  
n++;
```

NOTES

NOTES

```
    if (n<10)
    {
        goto loop;
    }
    getch();
    return 0;
}
```

The result of above code is as follows:

```
0
1
2
3
4
5
6
7
8
9
```

A `goto` statement causes your program to unconditionally transfer control to the statement associated with the label specified on the `goto` statement. Because, the `goto` statement can interfere with the normal sequence of processing, it makes a program more difficult to read and maintain. Often, a `break` statement, a `continue` statement, or a function call can eliminate the need for a `goto` statement. If an active block is exited using a `goto` statement, any local variables are destroyed when control is transferred from that block. You cannot use a `goto` statement to jump over initializations.

Check Your Progress

11. Write the syntax for switch statement.
12. How does the switch keyword function?
13. What does the break statement do?

2.9 LOGICAL VARIABLES

Variables that are declared inside a function or block are called logical variables. They can be used only by statements that are inside that function or block of code. Logical variables are not known to functions outside their own. The following example shows how logical variables are used. Here, all the variables `a`, `b`, and `c` are logical to `main ()` function.

Example: 2.18

```
#include <stdio.h>
#include <conio.h>
int main () {
    /* logical variable declaration */
    int a, b;
    int c;
    /* actual initialization */
    a = 10;
    b = 20;
    c = a + b;
    printf ("value of a = %d, b = %d and c = %d\n", a, b, c);
    return 0;
}
```

NOTES

2.10 DOUBLE PRECISION VARIABLE

Double precision variable is also a data type which is used to represent the floating point numbers. It is a 64-bit IEEE 754 double precision floating point number for the value. It has 15 decimal digits of precision.

Syntax:

```
double variable_name;
```

Example: 2.19

```
#include<stdio.h>
#include<string.h>
int main() {
    float x = 10.327;
    double y = 4244.546;
    int z = 28;
    printf ("The float value : %f\n", x);
    printf ("The double value : %f\n", y);
    printf ("The sum of float,
double and int variable : %f\n", (x+y+z));
    return 0;
}
```

Output:

```
The float value : 10.327000
The double value : 4244.546000
The sum of float, double and int variable : 4282.873000
```

2.11 SUBSCRIPTED AND DIMENSION VARIABLE

NOTES

Subscripted variables are used to store the values of the same type (for example in an array). It also helps us to store more values in a single variable name. Subscripted variables are declared by giving the variable name along with the subscript.

Example: 2.20

A[10]

Here A is the variable name and 10 is known as the subscripted value through which we can store 10 values in the variable name A. The 10 values are stored in A[1], A[2], ..., A[10]. As only one subscript is given for the above variable, the array is called as the single dimensional array.

Solution:

```
#include <stdio.h>
#include <conio.h>
void main()
{
    int arr[10];
    int i;
    printf("\n\nRead and Print elements of an array:\n");
    printf("—————\n");
    printf("Input 10 elements in the array :\n");
    for(i=0; i<10; i++)
    {
        printf("element - %d : ", i);
        scanf("%d", &arr[i]);
    }
    printf ("\n Elements in array are: ");
    for(i=0; i<10; i++)
    {
        printf ("%d ", arr[i]);
    }
    printf("\n");
}
```

Output:

Read and Print elements of an array:

Input 10 elements in the array:

```
element - 0 : 1
element - 1 : 1
element - 2 : 2
element - 3 : 3
element - 4 : 4
element - 5 : 5
element - 6 : 6
element - 7 : 7
element - 8 : 8
element - 9 : 9
```

Elements in array are: 1 1 2 3 4 5 6 7 8 9

NOTES

Dimensional Variable

An array is called as a dimensional variable. An array is a group of related items that store values under the common name.

Syntax:

The syntax for declaring an array is as follows:

```
datatype array_name [size];
```

Types of Arrays

Arrays are broadly classified into two types which are as follows.

1. One dimensional arrays
2. Two dimensional arrays

One Dimensional Array

The syntax for one dimensional array is as follows.

```
datatype array name [size]
```

For example, int a[5]

Example: 2.21

```
#include<stdio.h>
#include<conio.h>
int main ( ){
    int a[5] = {10,20,30,40,50};
    int i;
    printf ("elements of the array are");
    for ( i=0; i<5; i++)
        printf ("%d", a[i]);
}
```

Output:

Elements of the array are
10 20 30 40 50

NOTES

Two Dimensional Array

The syntax for two dimensional array is as follows.

```
datatype array_name [rowsize] [column size];
```

For example: int a[5][5];

Example: 2.22

```
#include<stdio.h>
#include<conio.h>
main ( )
{
    int a[3][3] = {10,20,30,40,50,60,70,80,90};
    int i,j;
    printf ("elements of the array are");
    for ( i=0; i<3; i++){
        for (j=0;j<3; j++){
            printf("%d \t", a[i] [j]);
        }
        printf("\n");
    }
}
```

Output:

```
Elements of the array are:
10 20 30
40 50 60
70 80 90
```

2.12 FUNCTION AND SUBROUTINE

A function is a group of statements that together perform a task. Every C program has at least one function, which is main (), and all the most trivial programs can define additional functions. We can divide code into separate functions, but logically the division is such that each function performs a specific task.

Parts of a Function

- **Return Type:** A function may return a value. The **return_type** is the data type of the value that 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. 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.

Function Declarations

A function declaration tells the compiler about a function name and how to call the function. The actual body of the function can be defined separately.

Syntax:

```
return_type function_name ( parameter list );  
For example,  
int max (int num1, int num2);  
int max (int, int);
```

Function declaration is required, when you define a function in one source file and you call that function in another file. In such case, you should declare the function at the top of the file calling the function.

Calling a Function

When a program calls a function, the program control is transferred to the called function. A called function performs a defined task and when its return statement is executed or when its function-ending closing brace is reached, it returns the program control back to the main program.

To call a function, you simply need to pass the required parameters along with the function name, and if the function returns a value, then you can store the returned value.

Example: 2.23 A function returning the max between two numbers.

```
#include <stdio.h>  
#include <conio.h>  
/* function declaration */  
int max(int num1, int num2);  
int main () {  
/* local variable definition */  
int a = 100;  
int b = 200;  
int ret;  
/* calling a function to get max value */  
ret = max(a, b);  
printf( "Max value is : %d\n", ret );  
return 0;  
}  
/* function returning the max between two numbers */  
int max(int num1, int num2) {  
/* local variable declaration */  
int result;
```

NOTES

NOTES

```
if (num1 > num2)
result = num1;
else
result = num2;
return result;
}
```

Output:

Max value is: 200

Subroutines

We will use a function if we need to do a complicated calculation that has only one result which we may or may not want to subsequently use in an expression. Subroutines, on the other hand, can return several results. However, calls to subroutines cannot be placed in an expression.

In the main program, a subroutine is activated using a CALL statement which include the subroutine name followed by the list of inputs and outputs from the subroutine surrounded by parenthesis. The inputs and outputs are collectively called the arguments.

A subroutine name follows the same rules as for function names and variable names: less than six letters and numbers and beginning with a letter. Because of this, subroutine names should be different than those used for variables or functions.

As with functions, there are some rules for using subroutines. Keep these in mind when writing your subroutines:

- We do not need to declare the subroutine name in the main program as we do with a function name.
- They begin with a line that includes the word SUBROUTINE, the name of the subroutine, and the arguments for the subroutine.

All variables used by the subroutine, including the arguments, must be declared in the subroutine. The subroutine name is not declared anywhere in the program. A subroutine is finished off with a RETURN and an END statement.

Example: 2.24

```
Program Subdem
Real A,B,C,Sum,Sumsq
Call Input(+A,B,C)
  Call Calc(A,B,C,Sum,Sumsq)
  Call Output(Sum,Sumsq)
End

Subroutine Input(X,Y,Z)
Real X,Y,Z
Print *, 'Enter Three Numbers => '
Read *,X,Y,Z
Return
End
```

```
Subroutine Calc (A,B,C, Sum, Sumsq)
Real A,B,C,Sum,Sumsq
Sum = A + B + C
Sumsq = Sum **2
Return
End
Subroutine Output (Sum,Sumsq)
Real Sum, Sumsq
PRINT *, 'The sum of the numbers you entered are:
',SUM
PRINT *, 'And the square of the sum is:',SUMSQ
RETURN
END
```

NOTES

2.13 COMMON AND DATA COMMANDS

Data definition commands are used to create, modify and delete database objects such as schemas, tables, views, indexes etc. Common data definition commands are as follows.

Create

The main use of `create` command is to create a new table in database. It has a predefined syntax in which we specify the columns and their respective data types.

Syntax:

```
Create Table <Table Name>
( <Column Name> <Data Type>,
<Column Name> <Data Type>,
<Column Name> <Data Type>,
<Column Name> <Data Type>
);
```

Example: 2.25

Create a student table with columns student name and roll number.

```
CREATE TABLE STUDENT
(STUDENT_NAME VARCHAR(30),
ROLL_NUMBER INT
);
```

Alter

An existing database object can be modified using the `alter` command. `Alter` command can perform following changes to any table.

- Add new columns.

- Add new integrity constraints.
- Modify existing columns.
- Drop integrity constraints.

NOTES

Syntax:

For adding a new column:

```
ALTER TABLE <table_name> ADD <column_name>
```

For renaming a table:

```
ALTER TABLE <table_name> RENAME To <new_table_name >
```

For modifying a column:

```
ALTER TABLE <table_name> MODIFY <column_name > <data type >
```

For deleting a column:

```
ALTER TABLE <table_name> DROP COLUMN <column_name>
```

Drop

This command can delete an index, table or view. Basically, any component from a relational database management system can be removed using the Drop command. Once the object is dropped, it cannot be reused.

The general syntax of drop command is as follows:

```
Drop Table <table_name>;
```

```
Drop Database <database_name>;
```

```
Drop Table <index_name>;
```

Truncate

Using the truncate command, all the records in a database are deleted, but the database structure is maintained.

Syntax:

```
Truncate Table <table name>
```

Comment

This command is used to add comments to the data dictionary.

Syntax:

- Single line comments: use ‘ — ‘ before any text.
- Multiline comments: /* comments in between */

Rename

The rename command renames an object in a record.

Syntax:

```
Rename <old name> to <new name>
```

Check Your Progress

14. Write the use of subscripted variables.
15. Define the term function.

NOTES

2.14 ANSWERS TO 'CHECK YOUR PROGRESS'

1. Variables that are declared inside a function or block are called logical variables.
2. The popularity of C is increasing probably due to its many desirable qualities. It is a robust language whose rich set of built-in functions and operators can be used of built-in functions and operators can be used to write any complex program.
3. The task of programming of data is accomplished by executing a sequence of precise instruction called a program.
4. Identifiers are names that are given to various program elements, such as variables, functions and arrays. Identifiers consisted of letters and digits, in any order, except that first character must be a letter.
5. There are four basic types of constants in C. They are integer constants, floating point constants, character constants and string constants.
6. Variable is an identifier that is used to represent a single data item, i.e., a numerical quantity or a character constant. The data item must be assigned to the variable at some point in the program. A given variable can be assigned different data items at various places within the program.
7. An operator refers to a symbol which indicates an operation to be executed.
8. All valid combinations of variables, constants, operators and functions are termed as expressions.
9. The read statement allows us to read given values into indicated variables and the write statement allows us to display results.
10. A robust algorithm is one, which produces reasonable results no matter what input is supplied.
11. The syntax of the `switch` statement is given below:

```
switch (expression)
{
case constant or expression : statements
case constant or expression : statements
..
default : statements }
```

NOTES

12. When the `switch` keyword is encountered, the associated expression is evaluated. The program now looks for the `case`, which matches with the value of the expression. Execution then starts from the statement corresponding to the `case` which matches. Each `case` has to be accompanied by integer expressions, which must be unique, as otherwise the program will not know where to start. There may be occasions when none of the constant expressions matches the `switch` expression in which case the `default` statements will be executed.
13. The `break` statement takes the program to the end of the `switch` statement. The end is just the closing brace corresponding to `switch` after the default `printf()` statement.
14. Subscripted variables are used to store the values of the same type in an array. It also helps us to store more values in a single variable name. Subscripted variables are declared by giving the variable name along with the subscript.
15. When a program calls a function, the program control is transferred to the called function. A called function performs a defined task and when its return statement is executed or when its function-ending closing brace is reached, it returns the program control back to the main program.

2.15 SUMMARY

- C was an offspring of the 'Basic Combined Programming Language' (BCPL) called B, developed in 1960s at Cambridge University. B language was modified by Dennis Ritchie and was implemented at Bell Laboratories in 1972.
- The new language was named C. Since it was developed along with the UNIX operating system, it is strongly associated with UNIX. This operating system was developed at Bell Laboratories and was coded almost entirely in C.
- The popularity of C is increasing probably due to its many desirable qualities. It is a robust language whose rich set of built-in functions and operators can be used of built-in functions and operators can be used to write any complex program.
- A programming language is designed to help process certain kinds of data consisting of numbers, characters and strings and to provide useful output known as information.
- The task of programming of data is accomplished by executing a sequence of precise instruction called a program.
- There are four basic types of constants in C i.e. integer constants, floating point constants, character constants and string constants.

- Unsigned integer constants may exceed the magnitude of ordinary integer constants by approximately a factor of 1, though they may not be negative. An unsigned integer constant can be identified by appending the letter (u) (either upper or lowercase) to the end of the constant.
- Variable is an identifier that is used to represent a single data item, i.e., a numerical quantity or a character constant. The data item must be assigned to the variable at some point in the program. A given variable can be assigned different data items at various places within the program.
- An operator refers to a symbol which indicates an operation to be executed. Operators are used to manipulate data in a program. The data items that operators act upon are called operands.
- All valid combinations of variables, constants, operators and functions are termed as expressions. Expressions always return a result. They can be as simple as a single value and as complex as a large calculation.
- Input and output operations are very much concerned with interacting with programmers and so their form is highly dependent on the specific programming language used, and occasionally, even on the computer system itself.
- Formatted Input/output functions get the input from the keyboard and give the output to the VDU as per the specified requirement. The scanf() and printf() comes under the formatted input/output functions.
- In the if-else syntax, if <statement1> or <statement2> is another if [-else] statement, then it is said to be a nested if statement.
- Control statements are used in C programs for solving complex problems.
- Depending on the occurrence of a particular situation if and else keywords are used for branching to different segments of the program. Relational operators are used in conjunction with branching constructs.
- The syntax of if statement is, if (condition) {statements}. If the condition is true, then a single statement or group of statements following if will be executed. If more than one statement is to be executed, then the statements are grouped within braces. If it is a single statement then curly braces are not required.
- Each opening curly brace has to have a matching closing curly brace.
- The syntax of if .. else is if (condition true) {statements s1} else {statements s2}. The statement else is always associated with an if. If the condition is true, then statements s1 will be executed else statements s2 will be executed.
- 'C' provides three logical operators for combining more than one condition. These are logical AND represented as &&, logical OR represented as || and Negation or NOT represented as ! (Exclamation).

NOTES

NOTES

- The syntax for the conditional operator is, `(Condition)? statement1: statement2;`. If the condition is true, execute `statement1`; else execute `statement2`.
- The `for` statement is meant for the easy implementation of iterations unlike `if`. The syntax for `for` loop is, `for (exp1; exp2; exp3){statements;}`.
- The `while` loop is a subset of `for` loop. The syntax for the `while` loop is, `while (expression){statements;}`.
- `switch` statements are used for implementation of multiway decision-making. Every `switch` statement contains a condition in the form of an expression.
- The `break` statement is used if we do not want the program to execute irrelevant statements.
- The `return` statement can appear anywhere in a function and when it is encountered a value is returned to the called function.
- Variables that are declared inside a function or block are called logical variables.
- Logical variables can be used only by statements that are inside that function or block of code. Logical variables are not known to functions outside their own function.
- Double precision variable is also a data type which is used to represent the floating point numbers. It is a 64-bit IEEE 754 double precision floating point number for the value.
- Subscripted variables are used to store the values of the same type in an array. It also helps us to store more values in a single variable name. Subscripted variables are declared by giving the variable name along with the subscript.
- A function is a group of statements that together perform a task. Every C program has at least one function, which is `main()`, and all the most trivial programs can define additional functions.
- A function declaration tells the compiler about a function name and how to call the function. The actual body of the function can be defined separately.
- A called function performs a defined task and when its `return` statement is executed or when its function-ending closing brace is reached, it returns the program control back to the main program.

2.16 KEY TERMS

- **Tokens:** It is the smallest element of a program that is meaningful to the compiler.
- **Constants:** It refers to fixed values that do not change during the execution of a program.

- **Variables:** It is an identifier that is used to represent some specified type of information within a designated portion of the program.
- **Operator:** An operator refers to a symbol which indicates an operation to be executed.
- **Expressions:** All valid combinations of variables, constants, operators, and functions are termed as expressions.
- **Loop or Iteration:** It is used to perform the same operation a number of times by repeating the same operation with one or more of the values changed.
- **Logical Variables:** These are the variables that are declared inside a function or block.
- **Function:** It is a group of statements that together perform a task.

NOTES

2.17 SELF-ASSESSMENT QUESTIONS AND EXERCISES

Short-Answer Questions

1. What is C language?
2. Define the term token.
3. Write the use of logical variables.
4. Differentiate between one-dimensional and two dimensional arrays.
5. What are string constants?
6. Define type casting and state its types.
7. State the requirements of a robust program or algorithm.
8. Write all the control structures in C.
9. Define the term do statement.
10. What are the different parts of a function?

Long-Answer Questions

1. Discuss briefly the basic structure of C program with the help of diagram.
2. Describe briefly various elements of the computer language giving appropriate examples.
3. Explain the constants and its types with the help of appropriate examples.
4. Describe the various types of operators with examples.
5. Discuss the classification of input and output operations in C.
6. Describe briefly control structure in C giving appropriate examples.
7. Briefly explain the logical variables with the help of appropriate program.
8. Describe the double precision variables with the help of appropriate examples.

NOTES

9. Discuss about the subscripted variable and dimension with the help of example.
10. Briefly explain the function and subroutine with the help of examples.
11. Discuss about the common and data statements giving appropriate examples.

2.18 FURTHER READING

- Dey, Pradip and Manas Ghosh. *Computer Fundamentals and Programming in C*. New Delhi: Oxford Higher Education, 2006.
- Bronson, Gary J. *A First Book of ANSI C*, 3rd edition. California: Thomson, Brooks Cole, 2000.
- Kanetkar, Yashwant. *Understanding Pointers in C*. New Delhi: BPB Publication, 2001.
- Kanetkar, Yashwant. *Let us C*. New Delhi: BPB Publication, 1999.
- Kernighan, Brian W. and Dennis Ritchie. *C Programming Language*, 2nd edition. New Jersey: Prentice Hall, 1988.
- Foster, W. D. and L. S. Foster. *C by Discovery*. Boston: Addison-Wesley, 2005.
- Kanetkar, Yashwant. *Working with C*. New Delhi: BPB Publication, 2003.
- Horton, Ivor. *Instant C Programming*. New Jersey: Wrox Press (John Willey & Sons), 1995.
- Lawlor, Steven C. *The Art of Programming Computer Science with 'C'*. New Jersey: West Publishing Company, 1996.

UNIT 3 PROGRAMMING IN CHEMISTRY

NOTES

Structure

- 3.0 Introduction
- 3.1 Objectives
- 3.2 Chemistry and Basic Programming
 - 3.2.1 Elements of Programming
- 3.3 Determination of Molecular Weight
- 3.4 Debye-Huckel Limiting Law Theory
- 3.5 Ionic Mobilities of Ions
- 3.6 Thermodynamic Parameters
- 3.7 Van der Waal's Equation
- 3.8 Radioactive Decay
- 3.9 Application of Chemical Kinetics
- 3.10 Determination of Lattice Energy
- 3.11 pH Titration
- 3.12 Program to Obtain the Value of Atomic Mass Unit in MeV
- 3.13 Determination of RMS, Average and Most Probable Velocities of Gases
- 3.14 Answers to 'Check Your Progress'
- 3.15 Summary
- 3.16 Key Terms
- 3.17 Self-Assessment Questions and Exercises
- 3.18 Further Reading

3.0 INTRODUCTION

Computer programming languages allow us to give instructions to a computer in a language the computer understands. Just as many human-based languages exist, there are an array of computer programming languages that programmers can use to communicate with a computer. Programming has become synonymous with computers. Millions of programmers are engaged in developing programs all over the world and billions of lines of program code have been developed and delivered to be used by a wide cross-section of society. Computer programs, also known as software systems, have helped in breaking many barriers. Today, an aircraft may not even take-off without its corresponding software system functioning properly. Satellite and space shuttle technology, weather forecasting, oil exploration, e-commerce, e-governance, the Internet, Intranet, e-mail, etc., have been made easy by successful computer programs. Computer software has helped mankind to enhance productivity, efficiency and, above all, quality of life.

In this unit, you will learn about the development of programs on determination of molecular weight, Debye-Huckel law theory, ionic mobilities, van der Waal's equation, radioactive decay, application of chemical kinetics, lattice energy, pH titrations, value of atomic mass unit in MeV and determination of RMS.

NOTES

3.1 OBJECTIVES

After going through this unit, you will be able to:

- Understand the basic programming of chemistry
- Describe the determination of molecular weight
- Explain the Debye-Huckel law theory
- Discuss the about the ionic mobilities
- Define the significance thermodynamic parameters
- Analyze the van der Waal's equation
- Understand the basic of radioactive decay
- Discuss the various application of chemical kinetics
- Describe the lattice energy
- Explain the pH titrations
- Define the value of atomic mass unit in MeV
- Describe the determination of RMS, average and most probable velocities of gases

3.2 CHEMISTRY AND BASIC PROGRAMMING

Computer programming languages allow us to give instructions to a computer in a language the computer understands. Just as many human-based languages exist, there are an array of computer programming languages that programmers can use to communicate with a computer. A computer can understand the binary language. Translating programming language into binary form is known as 'compiling'. Every programming language has its own distinct features, though many times there are similarities between programming languages.

These languages allows computers to quickly and efficiently process large and complex swaths of information. For example, if a person is given a list of randomized numbers ranging from one to ten thousand and is asked to place them in ascending order, chances are that it will take a sizable amount of time and include some errors. There are dozens of programming languages used in the industry today.

3.2.1 Elements of Programming

Despite notational differences, contemporary computer languages provide almost programming structures. These include basic control structures and data structures. The former provide the means to express algorithms, and the latter provide ways to organize information.

Programs written in procedural languages, the most common kind, are like recipes, having lists of ingredients and step-by-step instructions for using them. The three basic control structures in virtually every procedural languages are:

1. Sequence—combine the liquid ingredients, and next add the dry ones.

2. Conditional—if the tomatoes are fresh then simmer them, but if canned, skip this step.
3. Iterative—beat the egg whites until they form soft peaks.

Sequence is the default control structure in which the instructions are executed one after another. They might, for example, carry out a series of arithmetic operations, assigning results to variables, to find the roots of a quadratic equation $ax^2 + bx + c = 0$. The conditional IF-THEN or IF-THEN-ELSE control structure allows a program to follow alternative paths of execution. Iteration, or looping, gives computers much of their power. They can repeat a sequence of steps as often as necessary, and appropriate repetitions of quite simple steps can solve complex problems.

These control structures can be combined. A sequence may contain several loops; a loop may contain a loop nested within it, or the two branches of a conditional may each contain sequences with loops and more conditionals. In the ‘pseudocode’ used in this article, ‘*’ indicates multiplication and ‘←’ is used to assign values to variables. The following programming fragment employs the IF-THEN structure for finding one root of the quadratic equation, using the quadratic formula:

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

The quadratic formula assumes that a is nonzero and that the discriminant (the portion within the square root sign) is not negative (in order to obtain a real number root). Conditionals check these assumptions as given below:

- IF $a = 0$ THEN
- ROOT ← $-c/b$
- ELSE
- DISCRIMINANT ← $b*b - 4*a*c$
- IF DISCRIMINANT ≥ 0 THEN
- ROOT ← $(-b + \text{SQUARE_ROOT}(\text{DISCRIMINANT}))/2*a$
- ENDIF
- ENDIF

The SQUARE_ROOT function used in the above fragment is an example of a subprogram (also called a procedure, subroutine, or function). A subprogram is like a sauce recipe given once and used as part of many other recipes. Commonly used subprograms are generally available in the library provided with a language. Subprograms may call other subprograms in their definitions, as shown by the following routine (where ABS is the absolute-value function). SQUARE_ROOT is implemented using a WHILE (indefinite) loop that produces a good approximation for the square root of real numbers unless x is very small or very large. A subprogram is written by declaring its name, the type of input data, and the output:

- FUNCTION SQUARE_ROOT(REAL x) RETURNS REAL

NOTES

NOTES

- $ROOT \leftarrow 1.0$
- $WHILE ABS(ROOT*ROOT - x) \geq 0.000001$
- $AND WHILE ROOT \leftarrow (x/ROOT + ROOT)/2$
- $RETURN ROOT$

Subprograms can break a problem into smaller, more tractable subproblems. Sometimes a problem may be solved by reducing it to a subproblem that is a smaller version of the original. In that case the routine is known as a recursive subprogram because it solves the problem by repeatedly calling itself. For example, the factorial function in mathematics ($n! = n(n-1)\cdots 3\cdot 2\cdot 1$ —i.e., the product of the first n integers), can be programmed as a recursive routine:

- $FUNCTION FACTORIAL(INTEGER n) RETURNS INTEGER$
- $IF n = 0 THEN RETURN 1$
- $ELSE RETURN n * FACTORIAL(n-1)$

The advantage of recursion is that it is often a simple restatement of a precise definition, one that avoids the bookkeeping details of an iterative solution.

At the machine-language level, loops and conditionals are implemented with branch instructions that say 'jump to' a new point in the program. The 'goto' statement in higher-level languages expresses the same operation but is rarely used because it makes it difficult for humans to follow the 'flow' of a program. Some languages, such as Java and Ada, do not allow it. Examples have been chosen from organic, inorganic and physical chemistry. Some programs presented in the chapter are very simple and can be solved using a desk calculator. Still they have been included to illustrate the principles and control statements of the BASIC language. Each program is discussed under three sections and they are as follows:

Section 1: Explanations for the chemical equations, formulae and symbols used in the program.

Section 2: The Algorithm of the program.

Section 3: The listing of the computer program along with the output data.

In addition, flow-charts have been included for selected programs.

3.3 DETERMINATION OF MOLECULAR WEIGHT

THE INPUT STATEMENT

Calculation of the Molecular Weights of Organic Compounds

The first program involves the calculation of the molecular weights of cholesterol and indigo using the atomic weight data on carbon, hydrogen, oxygen and nitrogen.

Explanations for Symbols and Formulas used in the Program

NAME.COMP.1\$ and NAME.COMP.2\$ represent the names of the two organic compounds. A. WT.HYD, A.WT.OXY and A.WT.NIT represent the atomic weights of carbon, hydrogen, oxygen and nitrogen. MOL.WT.1 is the numeric

variable representing the molecular weight cholesterol. MOL.WT.2 represents the molecular weight of indigo. MOL.WT.1 is given by the following basic statement:

$$\text{MOL.WT.1} = 27 * \text{A.WT.CAR} + 46 * \text{A.WT.HYD} + \text{A.WT.OXY}$$

The basic statement for MOL.WT.2 is:

$$\text{MOL.WT.2} = 16 * \text{A.WT.CAR} + 10 * \text{A.WT.HYD} + 2 * \text{A.WT.NIT} + 2 * \text{A.WT.OXY}$$

Algorithm

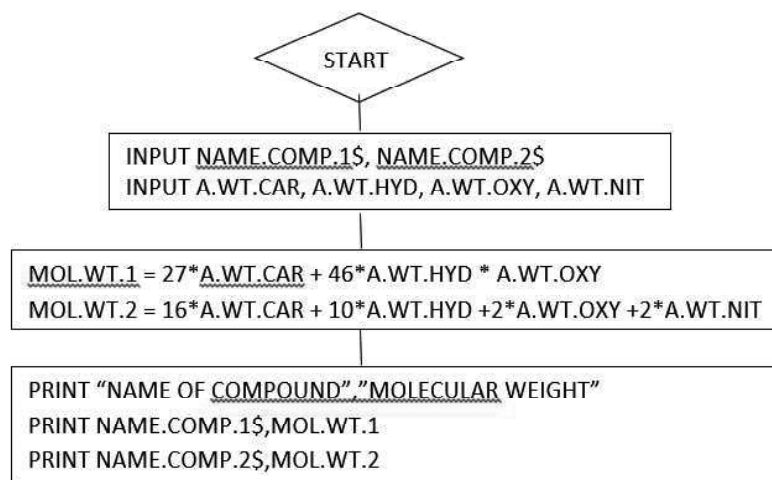
Step 1: Read the names of the organic compounds NAME.COMP.1\$ and NAME.COMP.2\$ Read the atomic weights of the elements C, H, O and N.

Step 2: Calculate the molecular weight of the compounds using the formulae for the molecular weights.

Step 3: Print the titles, name of compound and molecular weight. Print the names of the compounds and the corresponding molecular weights.

Step 4: Stop.

Flow-Chart



Program Listing and Output

```

10 REM MOLECULAR WEIGHTS OF CHOLESTEROLAND
20 REM INDIGO
30 REM SIMPLE PROGRAMS USING INPUT STATEMENT
40 REM CALCULATION OF MOLECULAR WEIGHT OF ORGANIC
   COMPOUND
50 INPUT "Give Names of the Organic Compounds 1 & 2",
   NAME.COMP.1$,NAME.COMP.2$
60 INPUT "Give the atomic wts of C, H, O, N", A.WT.CAR, A.WT.HYD,
   A.WT.OXY, A.WT.NIT
70 PRINT
80 MOL.WT.1=27*A.WT.CAR+46*A.WT.HYD+A.WT.OXY
90 MOL.WT.2 = 16*A.WT.CAR + 10*A.WT.HYD + 2.A.WT.NIT +
   2*A.WT.OXY
  
```

NOTES

NOTES

```
100 PRINT "Name of compound", "Molecular Weight"  
110 PRINT  
120 PRINT NAME.COMP.1$,MOL.WT.1  
130 PRINT NAME.COMP.2$,MOL.WT.2  
140 STOP  
150 END
```

Output:

Name of Compound	Molecular Weight
CHOLESTEROL	386.665
INDIGO	262.272

3.4 DEBYE-HUCKEL LIMITING LAW THEORY

The Read Statement and Library Functions

Calculations of mean activity coefficient of an electrolyte.

Explanations for Formulas and Symbols

The program calculates the activity coefficient of CuSO₄ using the Debye-Huckel limiting law as given below:

$$\text{Log}_{10} f_{\pm} = -0.509 |z_1 z_2| (I)^{1/2}$$

The antilog of $\log_{10} f_{\pm}$ has to be calculated to obtain f_{\pm} . Above equation is alternatively written in the natural logarithm form as shown below:

$$\ln f_{\pm} = -0.509 * 2.303 |z_1 z_2| (I)^{1/2}$$

The relevant BASIC statements are:

$$F1 = -.509 * P * \text{SQR} (\text{IONIC}.\text{STR})$$

Where

$$P = \text{ABS}(\text{CHARGE}.\text{1} * \text{CHARGE}.\text{2})$$

$$\text{IONIC}.\text{STR} = D/2$$

and

$$\text{IONIC}.\text{STR} = D/2$$

D is given by

$$D = \text{CONC}.\text{1} * \text{CHARGE}.\text{1} * \text{CHARGE}.\text{1} + \text{CONC}.\text{2} * \text{CHARGE}.\text{2} * \text{CHARGE}.\text{2}$$

$$F2 = F1 * 2.303$$

F2 corresponds to $\log_e f_{\pm}$

The variables used in chemical equations and the corresponding basic variables are given as follows:

Variables used in equations Basic variables

$z_1 z_2$	CHARGE.1, CHARGE.2
$C_1 C_2$	CONC.1, CONC.2
$C_1 z_1$	D
I	IONIC.STR
$ z_1 z_2 $	$(\text{ABS}(\text{CHARGE.1} * \text{CHARGE.2})) = P$

NOTES

Activity coefficient, ACT.COEF, is evaluated by using the statement 110
ACT.COEF = EXP (F2)

Algorithm

Step 1: Read the name of the electrolyte, NAME.ELECT\$ and the values of CHARGE.1, CHARGE.2, CONC.1 AND CONC.2

Step 2: Calculate P, D and IONIC.STR using the following statements:

$$P = \text{ABS}(\text{CHARGE.1} * \text{CHARGE.2})$$

$$D = \text{CONC.1} * \text{CHARGE.1} * \text{CHARGE.1} + \text{CONC.2} * \text{CHARGE.2} * \text{CHARGE.2}$$

$$\text{IONIC.STR} = D/2$$

Step 3: Calculate F1 and F2 using the statements

$$F1 = -.509 * \text{SQR}(\text{IONIC.STR})$$

$$F2 = F1 * 2.303$$

Step 4: Calculate the activity coefficient using the formula

$$\text{ACT.COEF} = \text{EXP}(F2)$$

Step 5: Print the string constant, "Activity coefficient of Copper Sulphate=" and the value of ACT.COEF

Step 6: Stop.

Program Listing and Output

```

10 REM PROGRAM ILLUSTRATING THE USE OF READ
   STATEMENT AND BUILT IN FUNCTION
20 REM IN FUNCTION
30 REM CALCULATION OF MEAN ACTIVITY COEFFICIENT OF
   STRONG
35 REM ELECTROLYTE
40 READ NAMEELECT$
50 READ CHARGE.1, CHARGE.2, CONC.1, CONC.2
60 P=ABS (CHARGE.1*CHARGE.2)
70 D= CONC.1*CHARGE.1*CHARGE.1 + CONC.2*CHARGE.2*
   CHARGE.2
80 IONIC.STR = D/2
90 F1 = -.509*P*SQR (IONIC.STR)
100 F2 = F1.2.303
110 ACT.COEF = EXP(F2)

```

NOTES

```
120 PRINT "Activity coefficient of copper sulphate=",ACT.COEF
130 DATA COPPER SULPHATE
140 DATA 2.0, -2.0, 0.01, 0.01
150 STOP
160 END
```

Output:

Activity coefficient of copper sulphate = .3914954

3.5 IONIC MOBILITIES OF IONS

Using the If-then Statement for Doing Repetitive Calculations

Determination of ionic mobilities of ions from ion conductance data.

Formulas and Symbols Used in the Program:

The ionic mobility of an ion is related to the ion conductance by the formula,

$$\mu_1 = \frac{?}{I/F}$$

In this program, the ionic mobilities of 5 ions are calculated using conductance data. NAME.ION\$ represents the name of the ion. EQ1.COND is the numeric variable representing I .

Algorithm

Step 1: Initialize the value of the numeric variable I as zero.

Step 2: Read NAME.ION\$ and EQ1.COND

Step 3: Assign to FARADAY to the value, 96484.6. Calculate MOBILITY using the statement

$$\text{MOBILITY} = \text{EQ1.COND} / \text{FARADAY}$$

Print the name of the ion and the mobility in cm/sec/unit postgrad. Increase I by 1, if it is less than 5 go to statement number 50 and read the name of next ion and its ion conductance. Repeat the calculation of the mobility of the ion. Print the name of the ion and the mobility. When I equals 5 stop calculations.

```
10 REM PROGRAM ILLUSTRATING THE USE OF IF
STATEMENT
20 REM TO DO REPETITIVE CALCULATIONS.
DETERMINATION OF
30 REM IONIC MOBILITIES FROM ION CONDUCTANCE
VALUE
40 I = 0
50 READ NAME.ION$.EQ1.COND
60 FARADAY = 96484.6
70 MOBILITY = EQ1.COND / FARADAY
80 PRINT "Name of the Ion = "; name.ion$
```

```

90 print "ionic Mobility="";MOBILITY;"CM/SEC/UNIT POTGRAD."
100 PRINT
110 I = + 1
120 IF 1<5 THEN 50
130 DATA HYDROGEN ION, 349.82
140 DATA POTASSIUM ION, 72.12
150 DATA AMMONIUM ION, 73.40
160 DATA LITHIUM ION, 38.69
170 DATA HYDROXYL ION, 198.00
180 STOP
190 END

```

Output:

Name of the ion = HYDROGEN ION
 Ionic Mobility = 3.625656E – 03 cm/sec/unit potgrad
 Name of the Ion = POTASSIUM ION
 Ionic Mobility = 7.578412E – 04 cm/sec/unit potgrad
 Name of the Ion = AMMONIUM ION
 Ionic Mobility = 7.607432E – 04 cm/sec/unit potgrad
 Name of the Ion = LITHIUM ION
 Ionic Mobility = 4.009966E – 04 cm/sec/unit potgrad
 Name of the Ion = HYDROXYL ION
 Ionic Mobility = 2.052141E – 03 cm/sec/unit potgrad

NOTES

3.6 THERMODYNAMIC PARAMETERS

The if-then-else Statement

Calculation of thermodynamic parameters for adiabatic or isothermal reversible expansion of one mole of an ideal monoatomic gas. This program is used to calculate the ΔE , W and ΔS for the adiabatic or isothermal reversible expansion of monatomic ideal gas.

Formulas and Symbols used in the Program

The formulas for ΔE , W and ΔS for the isothermal reversible expansion of an ideal monatomic gas are:

$$\Delta E = 0$$

$$W = RT \ln \left(\frac{V_2}{V_1} \right) \text{ or } RT \ln \left(\frac{P_1}{P_2} \right)$$

$$\Delta S = R \ln \left(\frac{V_2}{V_1} \right) \text{ or } RT \ln \left(\frac{P_1}{P_2} \right)$$

NOTES

If the expansion is carried out adiabatically and reversibly, then

ΔE , W and ΔS are given by,

$$\Delta E = C_v(T_2 - T_1)$$

$$W = -\Delta E$$

$$\Delta S = 0$$

N\$ corresponds to the string “ISOTHERMAL EXPANSION” OR “ADIABATIC EXPANSION”. The numeric variables T,P1 and P2 represent T,P1 and P2 and ΔE W ΔS and are represented by the variables, ECH, W and SCH. The numeric variables, T1 and T2 CV represent T1, T2 and Cv. R corresponds to the gas constant in joules/deg/mole.

Algorithm

Step 1: Read N\$, the type of expansion. If N\$ = “ISOTHERMAL EXPANSION” then go to statement number 70 and read T,P1 and P2. Calculate ECH, W and SCH using the statements.

$$ECH = 0$$

$$W = R * T * \text{LOG}(P1/P2)$$

$$SCH = W/T$$

R is assigned the value 8.314

Step 2: Print ECH, W and SCH. If N\$ = “ADIABATIC EXPANSION”, then go to statement number 120 and read T1, T2 and CV. Calculate ECH, W and SCH using the statement.

$$ECH = CV*(T2 - T1)$$

$$W = -ECH$$

$$SCH = 0$$

Print ECH, W and SCH

Step 3: Stop

Program Listing and Output

```
10 REM TO DETERMINE THE THERMODYNAMIC
    PARAMETERS
20 REM FOR THE REVIERSIBLE ADIABATIC OR
30 REM ISOTHERMAL EXPANSION OF MONATOMIC IDEAL
    GAS
40 REM ILLUSTRATION OF THE USE OF IF-THEN-ELSE
    STATEMENT
50 READ N$
60 IF N$ = "ISOTHERMAL EXPANSION" THEN 70 ELSE 120
70 READ T,P1,P2
```

```

80 R = 8.314
90 ECH = 0
95 W = R*T*LOG(P1/P2)
100 SCH = W/T
110 PRINT ECH,W,SCH
115 GO TO 190
120 READ T1,T2,CV
130 ECH = CV*(T2 - T1)
140 W = - ECH
150 SCH = 0
160 PRINT ECH, W, SCH
170 DATA ADIABATIC EXPANSION
180 DATA 300,101.2M12.47
190 END

```

Output 1 (for adiabatic expansion):

```
-2479.2036    2479.2036    0.0
```

Output 2 (for isothermal expansion):

```
0          5743.108    19.14369
```

NOTES

3.7 VAN DER WAAL'S EQUATION

Given integers V, T, and n representing the volume, temperature and the number of moles of a real gas, the task is to calculate the pressure P of the gas using Van der Waal's Equation for real gas.

Van der Waal's Equation for Real Gas:

$$(P + a * n^2 / V^2) * (V - n * b) = n R T$$

where, average attraction between particles (**a**) = 1.360,

volume excluded by a mole of particles (**b**) = 0.03186,

Universal Gas constant (**R**) = 8.314

Input: V = 5, T = 275, n = 6

Output: 2847.64

Input: V = 7, T = 300, n = 10

Output: 3725.43

Approach: To solve the problem, simply calculate the pressure P of real gas by using the equation $P = ((n * R * T) / (V - n * b)) - (a * n * n) / (V * V)$ and print the result.

NOTES

Implementation in C++

```
#include <iostream>
#include <string>
#include <cmath>
using namespace std;
// Function to calculate the pressure of a
// real gas using Van der Wall's equation
void pressure_using_vanderwall(double
V, double T, double n)
{
    double a = 1.382;
    double b = 0.031;
    double R = 8.314;
    // Calculating pressure
    double P = ((n * R * T) / (V - n * b))
- (a * n * n) / (V * V);
    // Print the obtained result
    cout << P << endl;
}
// Driver code
int main()
{
    double V = 7, T = 300, n = 10;
    pressure_using_vanderwall(V, T, n);
    return 0;
}
```

3.8 RADIOACTIVE DECAY

A radioactive nucleus decays according to first order rate equation. The radioactive decay constant λ , is related to the time t_1 , the initial concentration a_1 and the concentration of reactant remaining at time t . The relation is given below:

$$\lambda = (1/t) \ln (a/a(a-x))$$

The half-life is given by the equation as:

$$T_{1/2} = 0.693/\lambda$$

The average life is obtained from $t_{1/2}$ using the relation.

$$\text{Average life} = 1.44t_{1/2}$$

The string variable NUCLEUS\$ represents the name of nucleus. The numeric variables TIME, INCON, RECON AND S correspond to t , a , $(a-x)$ and λ HL refers to $t_{1/2}$ and AVILFE corresponds to average life. The listing of the program and output are presented below.

```
10 REM APPLICATION IN NIORGANIC CHEMISTRY
20 REM PROGRAM TO CALCULATE THE HALF LIFE AND
  AVERAGE LIFE
25 REM OF AF RADIOACTIVE NUCLEUS
30 READ N
40 FOR I = 1 TO N
50 READ NUCLEUS$
60 READ TIME, INCON, RECON
70 K = LOG(INCON/RECON)
80 S = K/TIME
90 HL = .693/S
100 AVLIFE = 1.44 * HL
110 PRINT "NAME OF THE NUCLEUS IS ";NUCLEUS$
120 PRINT "HALF LIFE=";HL;"YEARS"
130 PRINT "AVERAGE LIFE = ";AVLIFE;"YEARS"
140 NEXT I
150 DATA
160 DATA RADIUM
170 DATA 3180,100,25
180 STOP
190 END
```

Output:

```
NAME OF THE NUCLEUS IS RADIUM
HALF LIFE = 1589.663 YEARS
AVERAGE LIFE = 2289.114 YEARS
```

NOTES

3.9 APPLICATION OF CHEMICAL KINETICS

In this section, we shall formulate programs to determine; (i) the rate constant of a first order reaction, (ii) the rate constant of a second order reaction using collision theory; (iii) the thermodynamic parameters using ARRT theory and (iv) The rate constant at zero ionic strength using rates for reactions in solution with different ionic strengths.

NOTES

Rate Constant of a First Order Reaction

The decomposition of azomethane ($\text{CH}_2\text{N}_2\text{CH}_3$) to ethane and N_2 is a first order reaction. The rate constant can be obtained by using the values of partial pressure P , of azomethane remaining at time t . The rate constants is evaluated from the following equation:

$$k_1 = (1/t)\ln (P/P_0)$$

Where P and P_0 are the partial pressures at time t_0 and t k_1 values for different P values are obtained and finally the average value is determined. The $t_{1/2}$ is also calculated using the formula,

$$t_{1/2} = 0.693/k_{av}$$

Rate Constant of a Second Order Reaction using Collision Theory

The decomposition of NO_2 to NO and O_2 is a second order reaction. The rate constant of this reaction can be obtained using collision theory.

The pre-exponential term in the Arrhenius equation (frequency factor), A , is raised to the collision cross section, σ , by the formula,

$$A = \sigma P N_0 (8kT/\pi\mu)^{1/2}$$

Where P is the probability factor, k is Boltzmann's constant, T is the absolute temperature, π equals 3.14159 and μ is the reduced mass given by $\mu = M_{\text{NO}_2}/(2N_0)$

The rate constant is given by the formula,

$$k_2 = A \exp(-E/RT)$$

Where, E is the energy of activation and R is the gas constant.

Thermodynamic Parameters from Absolute Reaction Rate Theory

According to the absolute reaction rate theory, the enthalpy change ΔH for a second order reaction given by,

$$\Delta H = E_a - 2RT$$

The entropy change is expressed as,

$$\Delta S = R \ln(hA/kT)$$

Where h is planck's constant, k is Boltzmann's constant is absolute temperature and A is pre exponential term in the Arrhenius equation

The free energy change is given by the formula,

$$\Delta G = \Delta H - T\Delta S$$

Determination of Rate Constant at Zero Ionic Strength from the Study Reaction Rates of Different Ionic Strengths.

According to the Bronsted-Bjerrum equation, the rate constant k is related to the ionic strength of the medium, I by the formula,

$$\log (k/k_0) = 1.02z_a z_b (I)^{1/2}$$

$\log (k_0)$ can be obtained from this equation as

$$\log k_0 = \log k - 1.02z_a z_b (I)^{1/2}$$

$\ln k_0$ is given by the equation,

$$\ln k_0 = 2.303(\log k - 1.02 z_a z_b (I)^{1/2})$$

k_0 is determined by taking the exponential of the expression for $\ln k_0$. z_a and z_b are the charges of the ions involved in the rate determining step and k_0 is the rate constant at zero ionic strength. k_0 is determined for each and the average of k_0 is reported.

The variables used in the calculations and the corresponding FORTRAN variable names are given below:

Variables used in the calculation	FORTRAN variable names
number of computations	N
name of program	TITLE
$K_p, P, P, t, k_p, k_2, g_{1/2}$	BC, PIN, PT, TIME, Y1; RATEX, HALFL
σ, p, N_p, π, μ	SIGMA, PR, AN, PI, YY1*Z1
E, R, A, k_2	AGE, R, FRE, RATEC
AH, E_2, b, A, AS, AG, T	ENTHA, ACE, H, FRE, ENCH, FREEN, T
Number of times k_0 is Computed	M2
$z_a, z_b, k_2, I, k_p, k_0(av)$	ZA, ZB, RT1, AIST, ZER, ZERATE

The program uses the computed GO TO statement involving the integer variable CODE. If CODE=1 then the first order rate constant is evaluated. If CODE=2, then studies of collision theory and ARRT theory are made. If CODE=3, the study of rates of reactions in different ionic strengths is made.

The listing of the program along with input and output data are presented below:

```

C APPLICATIONS IN CHEMICAL KINETICS.
C DETERMINATION OF RATE CONSTANT OF A FIRST ORDER
  REACTON
C DETERMINATION OF RATE CONSTANT OF A SECOND ORDER
  REACTION USING COLLISION THEORY
C DETERMINATION OF THE THERMODYNAMIC PARAMETERS
  USING
C ABSOLUTE REACTION RATE THEORY
C STUDY OF RATE CONSTANT VARIATION WITH IONIC
  STRENGTH
C CODE = 1 INDICATES THE STUDY OF FIRST ORDER REACTIONS
C CODE = 2 INDICATES THE STUDY USING COLLISION THEORY
  AND
C STUDY OF THEIR PARAMETERS BY ARRT THEORY.
C CODE = 3 INDICATES THE STUDY OF RATES, VARIATION WITH
  IONIC STRENGTH
  CHARACTER * 20 TITLE
  INTEGER CODE

```

NOTES

NOTES

```
      READ (1,10)N
10  FORMAT (I2)
      DO 150 I = 1,N
      READ (1,15) CODE
15  FORMAT (I2)
      READ(1,20) TITLE
20  FORMAT(A)
      BC = 1.38E - 16
      GO TO (30,50,90), CODE
30  CONTINUE
      C STUDY OF RATE CONSTANT OF FIRST ORDER REACTIONS
      WRITE(2, 31)
31  FORMAT (1X,STUDY OF RATE CONSTANT OF A FIRST ORDER
      REACTIONS, I)
      SUM = 0.0
      READ(1,29)N
29  FORMAT(E8,2)
      READ(1,32)M1
32  FORMAT(I2)
      DO 40 JJ = 1, M1
      READ(1,33)PT, TIME
33  FORMAT(3E8,2)
      X1=PT/PIN
      Y1 = - ALOG(X1)/TIME
      SUM = SUM + Y1
40  CONTINUE
      RATEC = SUM/FLOAT(M1)
      HALFL = 0.693/RATEC
      WRITE(2,35)
35  FORMAT(1X, NAME OF SYSTEM, 10X, RATE CONST.'10X,
      HALFLIFE'./)
      WRITE (2,37)TITLE,RATEC,HALFL
37  FORMAT(1X,A,E10.4,'PERSEC',5X,E10.4, 'SECS',/)
      WRITE(2,38)
38.  FORMAT (1X, ' _____',/)
      GO TO 150
```

```
50 CONTINUE
   C   STUDY OF 2ND ORDER REACTION BY COLLISION THEORY.
   WRITE(2,51)
51 FORMAT(1X,'STUDY OF 32ND ORDER KINETICS BY
   COLL.THEORY',/)
   READ(1,52)PR, SIGMA,AM,T,ACE
52 FORMAT(E8.2)
   YY1 = AM/2.
   AN = 6.0225E23
   R = 8.314E - 3
   PI = 3.14159
   Z1 = 1./AN
   Z2 = 8. * BC* T/(PI*YY1*Z1)
   Z3 = SORT(Z2)
   FRE = PR* SIGMA* AN* Z3(1000.0)
   ZZ1 = - ACE/(R* T)
   RATEC = FRE* EXP(ZZ1)
   WRITE(2,53)
53 FORMAT (1X,'NAME OF REACTION',5X,'FREQ.FACTOR',8X,'
   RATECONS.',/)
   WRITE(2,54)TITLE,FRE.,RATEC
54 FORMAT(1X,A,2X,E10.4,5X,E10.4,'LIT/MOL/SEC',/)
   WRITE(2,55)
55 FORMAT (1X, ' _____',/)
   C   STUDY OF THER.PARAMETERS BY ARRT THEORY
   WRITE(2,71)
71 FORMAT(1X,'STUDY OF THER.PARAMETERS DEL.S,DEL.H.DEL.
   G BY ARRT,/)
   H = 6.6256E - 27
   TT3 = H* FRE/(BC* T)
   TT4 = ALOG(TT3)
   ENCH = R* TT4* 1000.0
   ENTHA = ACE - 2* R* T
   FREEN = ENTHA - (T* ENCH/1000.0)
   WRITE(2,74)TITLE
74 FORMAT(1X, 'THE THER.PARAMETERS FOR 'A',ARE',/)
   WRITE(2,73)
```

NOTES

NOTES

```
73 FORMAT(1X,'FRE.FACTOR',8X,'DELTA-S',10X,DELTA-
H',9X,DECTAF',/)
WRITE(2,75)
75 FORMAT(1X,'(L/M/SEC)'8X.'(J/DEG/M)',10X,'(KJ/M),9X,'(KJ/M)'/
)
WRITE(2,76)FRE,ENCH,ENTHA,FREEN
76 FORMAT(1X,E10.4,6X,E10.4,6X,E10.4,6X,E10.4,/)
WRITE(2,77)
77 FORMAT (1X, ' _____',/)
GO TO 150
90 CONTINUE
C STUDY OF RATE CONSTANT VARIATION WITH IONIC
STRENGTH
WRITE (2,91)
91 FORMAT (1X, STUDY OF RATE CONS.VARIATION WITH IONIC
STR.',/)
READ(1,92)M2
92 FORMAT(I2)
READ(1,99)ZA,ZB
99 FORMAT(F4,1)
TX1=1.02* ZA * ZB
ZER = 0.0
DO 100 KK = 1,M2
READ(1,93) RTI,AIST
93 FORMAT(F7.4)
TK1 = ALOG10(RTI)
AI1 = SORT(AIST)
TK2 = TX1* AI1
TK3 = TK1 - TK2
TK4 = 2.303* TK3
TK5 = EXP(TK4)
ZER = ZER + TK5
100 CONTINUE
ZERATE = ZER/FLOAT(M2)
WRITE(2,95)TITLE,ZERATE
95 FORMAT(1X,'RATE CONS.AT ZERO ION.STR.FOR'A,'IS',E10.4,/)
)
150 CONTINUE
```

STOP

END

03

01

AZOMETHANE

8.20E - 02

04

5.72E - 02

1.00E + 03

3.99E - 02

2.00E + 03

2.78E - 02

3.00E + 03

1.94E - 02

4.00E + 03

02

2NO₂ - 2NO + O₂

0.05E + 00

0.10E - 14

4.60E + 01

5.00E - 02

03

BIMOLECULAR REACTION

05

- 1.0

- 2.0

01.0500

00.0025

01.1200

00.0037

01.1600

00.0045

01.1800

00.0065

01.2600

00.0085

NOTES

NOTES

Output data:

STUDY OF RATE CONSTANT OF A FIRST ORDER REACTION

NAME OF SYSTEM	RATE CONST.	HALFLIFE
AZOMETHANE	3603E-03PERSEC	1923E+04SECS

STUDY OF 2ND ORDER KINETICS BY COLL. THEORY

NAME OF REACTION	FREQ. FACTOR	RATECONS.
2NO ₂ = 2NO + O ₂	0.2043E+10	.5172E-02 LIT/ MOL./SEC

STUDY OF THER. PARAMETERS DELS, DELH, DELG BY ARRT

THE THER. PARAMETERS FOR 2NO₂ = 2NO + O₂

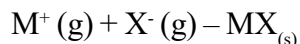
FRE.FACTOR	DELTA-S	DELTA-H DELTA F
(L/M/SEC)	(J/DEG/M) (KJ/M)	(KJ/M)
0.204E+10	-0.7097E+02 .102E+03	.1382e+03

STUDY OF RATE CONS. VARIATION WITH IONIC STR.

RATE CONS. AT ZERO ION STR. FOR BIMOLECULAR REACTION IS .8287E+00.

3.10 DETERMINATION OF LATTICE ENERGY

The energy of the crystal lattice of an ionic compound is the energy released when ions come together from infinite separation to form a crystal.



This energy can be calculated using the Born-Landé equation which is given below:

$$U_0 = (N_0 A Z^+ Z^- e^2) (1/r_0) (1 - (1/n))$$

Constant, Z^+ and Z^- are the charges of positive and negative ions and e is the charge of electron in esu. The Born exponent is represented by n and r_0 is the inter-nuclear distance in the crystal. For NaCl, Z^+ , Z^- , r_0 and n assume the values, 1.74756, +1, 2.81 Å and 8. The Born exponent is the average of the values for Na⁺ (7) and Cl⁻ (9). The variables used in the chemical calculations are listed along with the BASIC variables below.

Variables used in Chemical calculations	Basic variables
N ₀	AN
A	MACON
Z ⁺ Z ⁻	Z1,Z2
Name of crystal	NAMECRYSS
e	CH
Conversion factor from erg to kil. Joule	CF
U ₀ , r ₀ , n	LAT, DIS, BEX

10 REM TO CALCULATE LATTICE ENERGY OF A CRYSTAL
USING BORN-LAND EQUATION

```

20 READ Z1, Z2 BEX, DIS , MACON
30 READ NAMECRYSS$
40 AN = 6.0225E + 23
50 CH = 4.8 E - 10
55 CF = 1E - 10
60 P = Z1*Z2
70 X1 = AN * MACON*P
80 Y = 1/BEX
90 X2 = CH*CH*(1-Y)
100 LAT = X1 * (X2/DIS) *CE
110 PRINT "THE LATTICE ENERGY OF ", NAMECRYSS$, " IS
        ";LAT;"KJ/MOLE"
120 DATA 1 - 1,8,2381E - 8,1.74756
130 DATA NACL
140 STOP
150 END

```

Output:

THE LATTICE ENERGY OF NACL IS - 755.0802 KJ/MOLE

NOTES

3.11 pH TITRATION

Inorganic chemist make use of different types of indicators to carry out acid base titrations. The following program lists the various indicators along with their colors in acidic and basic media. The pH range the pKa of the many string variables and the use of TAB() function the numeric variables N and PK refer to the number of indicators and pKa values for indicators the string variables, IND\$, COLACID\$ AND PH\$ refer to the name of indicator color in acid, color in base and the pH range for specific indicator. The listing of the program along with the output is given below.

```

10 REM INDICATOR AND THEIR COLOUR IN ACIDIC AND BASIC
    MEDIA
20 INPUT "GIVE NUMBER OF INDICATORS";N
30 FOR I=1 TO N
40 INPUT "GIVE NAME OF INDICATORS";IND$
50 INPUT "COLOR IN ACID MEDIUM";COLACID$
60 INPUT "GIVE COLOR IN BASIC MEDIUM";COLDAS$
70 INPUT "GIVE PDA OF INDICATORS";RK
80 INPUT "PH RANGE";PH$
90 PRINT  TABE(1)  ;"NAME  ";TAB(20);"COLOUR  IN
        ACID";TAB(40);"COLOUR IN BASE"

```

NOTES

```
100 PRINT TAB (60);"PKA";TAB (70) PH RANGE"
105 PRINT
110 PRINT TAB(1) INDS;TAB(23);COLACI$;"TAB(43);
COLDA$;TAB(60);PK;TAB(73)'PH$
120 NEXT |
130 STOP
140 END
```

Output:

NAME	COLOUR IN ACID	COLOUR IN BASE	PKA	PH RANGE
PHENOLPHTHALEIN	COLOURLESS	PINK	9.4	8.3-10.0
THYMOL BLUE	RED	YELLOW	1.51	1.2-2.8
BROMOPHENOL BLUE	YELLOW	BLUE	3.98	3.0-4.6
CHLOROPHENOL BLUE	YELLOW	RED	5.98	4.8-6.4
BROMOTHYMOL BLUE	YELLOW	BLUE	7.0	6.0-7.6
CRESOL RED	YELLOW	RED	8.3	7.3-8.8
METHYL ORANGE	ORANGE	YELLOW	3.7	3.1-4.4
MTHYL RED	RED	YELLOW	5.1	4.2-6.3

3.12 PROGRAM TO OBTAIN THE VALUE OF ATOMIC MASS UNIT IN MEV

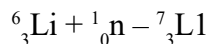
The first part of this program computes the value of one amu in Mev. The value of one amu is obtained using the Einstein mass energy relation, $E = mc^2$ substituting the values of proton mass for m and velocity of light for c we get

$$E = 1.66 \times 10^{-24} \times (2.99795 \times 10^{10})^2$$

$$= 1.491958 \times 10^{-3} \text{ergs.}$$

Using the conversion factor, $1 \text{MeV} = 1.602 \times 10^{-6} \text{ergs}$, we can obtain the value of 1 amu as 932.4744 MeV.

The second part of the program calculates the binding energy of a neutron or a nucleus and using the IF.THEN. ELSE statements. if the string variable N\$ is neutron, the binding energy of neutron is obtained. The binding energy of neutron is obtained from the reaction.



Using the masses of ${}^6\text{Li}$, ${}^1\text{n}$ and ${}^7\text{Li}$ the mass defect is calculated. The binding energy is obtained by multiplying the mass defect by 931.5, if the string variable N\$ is a nucleus then the binding energy is calculated for the nucleus. For example, let us consider the calculation of binding energy of Argon nucleus. This nucleus has 18 protons, 22 neutrons and 18 electrons. The mass defect is calculated using the following equation.

Mass defect = $(22 \times m_p + 18 \times m_n + 18 \times m_e) - \text{actual mass of Argon.}$

Binding energy = mass defect x AMU.

The variables used in the chemical calculations and the corresponding BASIC variables are listed below.

Variables used in Chemical calculations	Basic variables
c	C
m_p	MP, MASP
Conversion factor	CF
AMU	AMU
M_e, M_n	MEL, MASN
Number of protons	NUP
Number of electrons	NUE
Number of neutrons	NUN
Mass defect	MDE
Binding energy	BE

NOTES

The listing of the program and the output and the output are given below.

```

10 REM CALCULATION OF BINDING ENERGY OF A
   NUCLEUS OF PARTICLE
20 REDAD N$
30 C = 2.99795E + 10
40 MP = 1.66E - 24
50 CF = .0000016
60 AMU = C * C * MP / CF
70 PRINT "ONE ATOMIC MASS UNIT IS "; AMU; "MEV"
80 IF N$ = "NEUTRON" THEN 90 ELSE 140
90 READ MLI, MIL2
100 MDE = MIL2 - (MLI + MN)
110 BE = MDE * AMU
120 PRINT "BINDING ENERGY OF NEUTRON IS "; BE; "MEV"
130 GOTO 230
140 READ NUP, NUN, NUE, ACM
150 READ MEL, MASP, MASN
160 MCAL = NUP * MASP + NUN * MASN + NUE * MEL
170 MDE = MCAL - ACM
180 BE = MDE * AMU
190 PRINT "BINDING ENERGY OF "; N$; " IS "; BE; "MEV"
200 DATA 18.22, 18.39, 9.6238
220 DATA 0.0005986, 1.007277, 1.008665

```

230 STOP

240 END

Output:

ONE ATOMIC MASS UNIT IS = 932.4744 MEV

BINDING ENERGY OF NEUTRON IS = 345.0288 MEV

NOTES

3.13 DETERMINATION OF RMS, AVERAGE AND MOST PROBABLE VELOCITIES OF GASES

Formulas and Symbols used in the Program

The root mean square velocity of a gas is given by the formula,

$$C_{RMS} = (3RT/M)^{1/2}$$

The average and most probable velocities of a gas are given by the following formulas:

$$C_{av} = (8RT/\pi M)^{1/2}$$

$$C_{mp} = (2RT/M)^{1/2}$$

The string variables, N\$ and M\$ represent the types of velocity and the name of the molecule respectively. R, T and M represent the gas constant, temperature and molecular weight of the gas. CMP, CAV and CRMS represent the most probable, average and RMS velocities of the gas.

Algorithm

Step 1: Read the types of velocity, N\$ and the name of the molecule, M\$.

Step 2: Read the values of molecular weight, M and temperature, T. Assign to R the value 8.314E7. Assign to P1, the value 3.14152.

Step 3: If N\$ = "RMS VELOCITY" then go to statement number 130 to calculate CRMS using the statement

$$CRMS = SQR(3 * R * T/M)$$

Otherwise go to statement number 80. If N\$ = "AVERAGE VELOCITY" then go to statement number 100 to calculate CAV using the statement,

$$CAV = SQR(8 * R * T/(P1 * M))$$

Print the value of CAV in cm/sec.

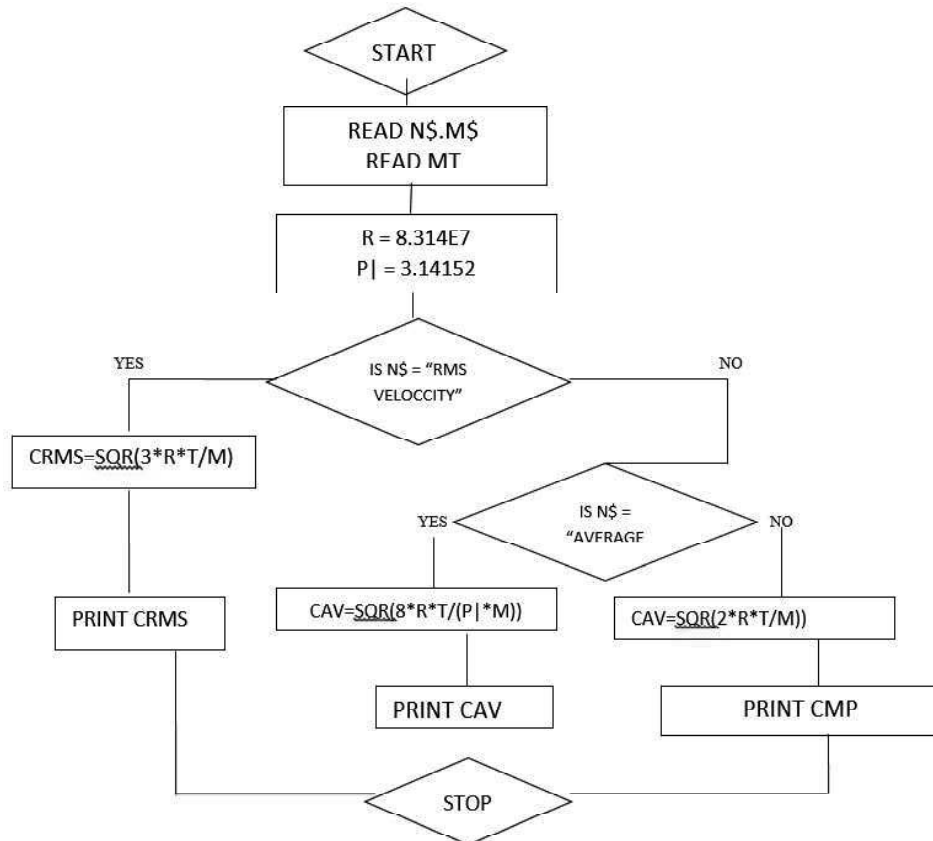
If N\$ is not equal to the string "AVERAGE VELOCITY" go to statement number 85 to calculate CMP using the statement.

$$CMP = SQR(2 * R * T/M)$$

Print CMP in cm/sec units.

Step 4: Stop.

Flow-Chart



NOTES

Program Listing and Output

```
10 REM PROGRAM TO CALUCLATE THE RMS.AVERAGE
20 REM OR MOST PROBABLE VELOCITY OF A GAS
30 REM ILLUSTRATION OF THE USE OF TWO
40 REM IF-TEHN STATEMENTS
50 READ N$
55 READ M,T
60 R = 8.314E7
65 P| = 3.14152
70 IF N$ = "RMS VELCOCCITY" THEN 130
80 IF N$ = "AVERAGE VELCOCCITY" THEN 100
90 PRINT "THE MOST PROBABLE VELOCITY OF ";M$" IS =
   ";CMP;"CM/SEC"
95 GO TO 180
100 CAV = SQR(8*R*T/(P|M))
110 PRINT "THE AVERAGE VELOCITY OF ";M$ IS =";CAV;"CM/SEC"
120 GO TO 180
130 CRMS = SQR (3*R*T/M)
```

NOTES

```
140 PRINT "THE RMS VELOCITY OF ";M$ IS = ";CMP;"CM/SEC"  
150 DATA RMS VELOCITY  
160 DATA OXYGEN  
170 DATA 32,300  
180 END
```

Output:

THE RMS VELOCITY OF OXYGEN IS = 48356.1 CM/SEC

Check Your Progress

1. What is the need of computer languages?
2. Write the Vander Waal's equation for real gas.
3. What is lattice energy?
4. Write the formula for root mean square velocity of a gas.

3.14 ANSWERS TO 'CHECK YOUR PROGRESS'

1. Computer programming languages allow us to give instructions to a computer in a language the computer understands. Just as many human-based languages exist, there are an array of computer programming languages that programmers can use to communicate with a computer.
2. Van der Waal's Equation for Real Gas:
$$(P + a * n^2 / V^2) * (V - n * b) = n R T$$
where, average attraction between particles (a) = 1.360,
volume excluded by a mole of particles (b) = 0.03186,
Universal Gas constant (R) = 8.314
3. The energy of the crystal lattice of an ionic compound is the energy released when ions come together from infinite separation to form a crystal.
4. The root mean square velocity of a gas is given by the formula,
$$C_{RMS} = (3RT/M)^{1/2}$$

3.15 SUMMARY

- Computer programming languages allow us to give instructions to a computer in a language the computer understands. Just as many human-based languages exist, there are an array of computer programming languages that programmers can use to communicate with a computer.
- Computer languages allow computers to quickly and efficiently process large and complex swaths of information. For example, if a person is given a list of randomized numbers ranging from one to ten thousand and is asked to place

them in ascending order, chances are that it will take a sizable amount of time and include some errors.

- Despite notational differences, contemporary computer languages provide many of the same programming structures. These include basic control structures and data structures. The former provide the means to express algorithms, and the latter provide ways to organize information.
- The three basic control structures in virtually every procedural languages are: sequence, conditional and iterative.
- The energy of the crystal lattice of an ionic compound is the energy released when ions come together from infinite separation to form a crystal.
- This energy can be calculated using the Born-Lande equation which is given below:

$$U_0 = (N_0 AZ^+ Z^- e^2) (1/r_0) (1-(1/n))$$

NOTES

3.16 KEY TERMS

- **Computer Programming:** It the process of creating an executable computer program to perform a specific task.
- **Lattice Energy:** It is the energy required to dissociate one mole of an ionic compound to its constituent ions in the gaseous state.
- **Radioactive Decay:** It is the process by which an unstable atomic nucleus loses energy by radiation.

3.17 SELF-ASSESSMENT QUESTIONS AND EXERCISES

Short-Answer Questions

1. State about the rate constant of a first order reaction.
2. What is pH titration?
3. State about the atomic mass unit in MeV.
4. Define the term thermodynamic parameters.
5. What is radioactive decay?
6. What is ionic mobilities?

Long-Answer Questions

1. Discuss about the various elements of programming with the help of examples.
2. Briefly explain the molecular weight giving appropriate example.
3. Analyze the Debye-Huckel law theory with the help of example.
4. Briefly explain the van der Waal's equation with the help of appropriate program.

5. Describe the various types of application in chemical kinetics.
6. Analyze the evaluation of lattice energy with the help of example.
7. Briefly explain the pH titration with the help of appropriate examples.

NOTES

3.18 FURTHER READING

Dey, Pradip and Manas Ghosh. *Computer Fundamentals and Programming in C*. New Delhi: Oxford Higher Education, 2006.

Bronson, Gary J. *A First Book of ANSI C*, 3rd edition. California: Thomson, Brooks Cole, 2000.

Kanetkar, Yashwant. *Understanding Pointers in C*. New Delhi: BPB Publication, 2001.

Kanetkar, Yashwant. *Let us C*. New Delhi: BPB Publication, 1999.

Kernighan, Brian W. and Dennis Ritchie. *C Programming Language*, 2nd edition. New Jersey: Prentice Hall, 1988.

Foster, W. D. and L. S. Foster. *C by Discovery*. Boston: Addison-Wesley, 2005.

Kanetkar, Yashwant. *Working with C*. New Delhi: BPB Publication, 2003.

Horton, Ivor. *Instant C Programming*. New Jersey: Wrox Press (John Willey & Sons), 1995.

Lawlor, Steven C. *The Art of Programming Computer Science with 'C'*. New Jersey: West Publishing Company, 1996.

UNIT 4 USE OF COMPUTER PROGRAMMES

NOTES

Structure

- 4.0 Introduction
- 4.1 Objectives
- 4.2 Standard Programmes and Packages
 - 4.2.1 Programming Process
- 4.3 Execution of Linear Regression
- 4.4 Numerical Integration and Differentiation
- 4.5 Monte Carlo and Molecular Dynamics
- 4.6 Introduction to Software Packages
 - 4.6.1 MATLAB
 - 4.6.2 EasyPlot
 - 4.6.3 FoxPro
- 4.7 Answers to 'Check Your Progress'
- 4.8 Summary
- 4.9 Key Terms
- 4.10 Self-Assessment Questions and Exercises
- 4.11 Further Reading

4.0 INTRODUCTION

We have to provide the computer with a set of instructions to solve the problem. This set of instructions is called a program. In other words, the specification of the sequence of computational steps in a particular programming language is termed as a program. The special computer understandable notation for the specification of such a sequence of computation steps is referred to as programming language. So, the task of developing programs is called programming and the person engaged in programming activity is known as programmer. A computer is extremely potent tool in the hands of modern man. It is capable of solving complex problems with unimaginable ease at incredible speed. To those who are not so much initiated into the world of computers are bound to look at it with awe. The extent and range of the works that computers can carry out do make them seem like magic box indeed.

In this unit, you will learn about the standard programmes and packages, execution of linear regression, X, Y Plot, numerical integration and differentiation, Monte Carlo and molecular dynamics.

4.1 OBJECTIVES

After going through this unit, you will be able to:

- Understand the significance of standard programmes and packages
- Describe the execution of linear regression

NOTES

- Explain the significance of X, Y plot
- Discuss the numerical integration and differentiation equations
- Define the Monte Carlo and molecular dynamics
- Elaborate on the programmes with data preferably
- Explain the MATLAB, EasyPlot, FoxPro

4.2 STANDARD PROGRAMMES AND PACKAGES

Why Programming?

You may already have used software, perhaps for word processing or spreadsheets, to solve problems. Perhaps now you are curious to learn how programmers write software. A program is a set of step-by-step instructions that directs the computer to do the tasks you want it to do and produce the results you want. There are at least three good reasons for learning programming:

- Programming helps you understand computers. The computer is only a tool. If you learn how to write simple programs, you will gain more knowledge about how a computer works.
- Writing a few simple programs increases your confidence level. Many people find great personal satisfaction in creating a set of instructions that solve a problem.
- Learning programming lets you find out quickly whether you like programming and whether you have the analytical turn of mind programmers need. Even if you decide that programming is not for you, understanding the process certainly will increase your appreciation of what programmers and computers can do.

A set of rules that provides a way of telling a computer what operations to perform is called a programming language. There is not, however, just one programming language; there are many. In this chapter you will learn about controlling a computer through the process of programming. You may even discover that you might want to become a programmer.

In general, the programmer's job is to convert problem solutions into instructions for the computer. That is, the programmer prepares the instructions of a computer program and runs those instructions on the computer, tests the program to see if it is working properly, and makes corrections (if required) to the program. The programmer also writes a report on the program. These activities are all done for the purpose of helping a user to fulfil its need, such as paying employees, billing customers, or admitting students to college.

The programming activities just described could be done, perhaps, as solo activities, but a programmer typically interacts with a variety of people. For example,

if a program is part of a system of several programs, the programmer coordinates with other programmers to make sure that the programs fit together well. If you were a programmer, you might also have coordination meetings with users, managers, systems analysts, and with peers who evaluate your work-just as you evaluate theirs.

4.2.1 Programming Process

It is the process of designing and creating the programs to perform a user specific task. There are five phases in the programming process.

1. Defining the problem
2. Planning the solution
3. Coding the program
4. Testing the program
5. Documenting the program

Defining the Problem

Suppose that, as a programmer, you are contacted because your services are needed. You meet with users from the client organization to analyze the problem, or you meet with a systems analyst who outlines the project. Specifically, the task of defining the problem consists of identifying what it is you know (input-given data), and what it is you want to obtain (output-the result). Eventually, you produce a written agreement that, among other things, specifies the kind of input, processing, and output required. This is not a simple process.

Planning the Solution

Two common ways of planning the solution of a problem are to draw a flowchart and to write pseudocode, or possibly both. Essentially, a flowchart is a pictorial representation of a step-by-step solution to a problem. It consists of arrows representing the direction the program takes and boxes and other symbols representing actions. It is a map of what your program is going to do and how it is going to do it. The American National Standards Institute (ANSI) has developed a standard set of flowchart symbols. Figure 4.1 shows the symbols and how they might be used in a simple flowchart of a common everyday act-preparing a letter for mailing.

Pseudocode is an English-like non-standard language that lets you state your solution with more precision than you can in plain English but with less precision than required when using a formal programming language. Pseudocode permits you to focus on the program logic without having to be concerned just yet about the precise syntax of a particular programming language. However, pseudocode is not executable on the computer. We will illustrate these later in this chapter, when we focus on language examples.

NOTES

NOTES

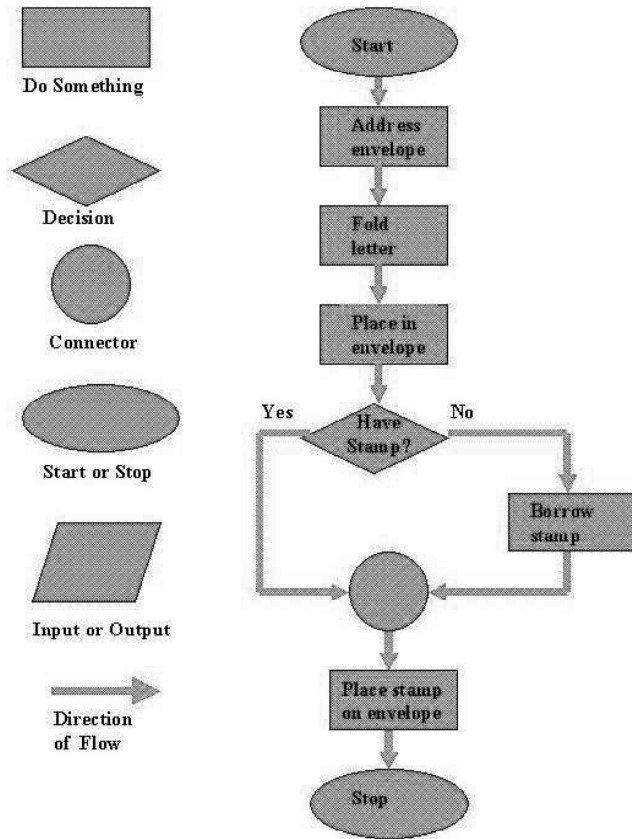


Fig. 4.1 Flow Chart Symbols and Flow Chart for Mailing Letter

Coding the Program

As a programmer, your next step is to code the program to express your solution in a programming language. You will apply the logic from the flowchart or pseudo code to a programming language. As we know that a programming language is a set of rules that provides a way of instructing the computer what operations to perform. There are many programming languages like BASIC, COBOL, Pascal, FORTRAN, and C. You may find yourself working with one or more of these.

Although programming languages operate grammatically, somewhat like the English language, they are much more precise. To get your program to work, you have to follow exactly the rules/ syntax of the language. Of course, using the language correctly is no guarantee that your program will work, any more than speaking grammatically correct English means you know what you are talking about. The point is that correct use of the language is the required first step.

Testing the Program

This phase identifies the errors that occurs during the execution of the program. You have to debug the error for successful compilation and running the program. Some experts insist that a well-designed program can be written correctly the first time. In fact, they assert that there are mathematical ways to prove that a program is correct. However, the imperfections of the world are still with us, so most programmers get used to the idea that their newly written programs probably have a few errors. This is a bit discouraging at first, since programmers tend to be precise,

Careful, detail-oriented people who take pride in their work. Still, there are many opportunities to introduce mistakes into programs, and you, just as those who have gone before you, will probably find several of them.

Documenting the Program

Documentation is a written detailed description of the programming cycle and specific facts about the program. Typical program documentation materials include the origin and nature of the problem, a brief narrative description of the program, logic tools such as flowcharts and pseudocode, data-record descriptions, program listings, and testing results. Comments in the program itself are also considered an essential part of documentation. Many programmers document as they code. In a broader sense, program documentation can be part of the documentation for an entire system. The wise programmer continues to document the program throughout its design, development, and testing. Documentation is needed to supplement human memory and to help organize program planning. Also, documentation is critical to communicate with others who have an interest in the program, especially other programmers who may be part of a programming team. And, since turnover is high in the computer industry, written documentation is needed so that those who come after you can make any necessary modifications in the program or track down any errors that you missed.

NOTES

4.3 EXECUTION OF LINEAR REGRESSION

Machine Learning, being a subset of Artificial Intelligence (AI), has been playing a dominant role in our daily lives. Data science engineers and developers working in various domains are widely using machine learning algorithms to make their tasks simpler and life easier. For example, certain machine learning algorithms enable Google Maps to find the fastest route to our destinations, allow Tesla to make driverless cars, help Amazon to generate almost 35% of their annual income, Accu Weather to get the weather forecast of 3.5 million locations weeks in advance, Facebook to automatically detect faces and suggest tags and so on.

In statistics and machine learning, linear regression is one of the most popular and well understood algorithms. Most data science enthusiasts and machine learning fanatics begin their journey with linear regression algorithms. In this article, we will look into how linear regression algorithm works and how it can be efficiently used in your machine learning projects to build better models.

Linear Regression is one of the machine learning algorithms where the result is predicted by the use of known parameters which are correlated with the output. It is used to predict values within a continuous range rather than trying to classify them into categories. The known parameters are used to make a continuous and constant slope which is used to predict the unknown or the result.

Linear regression attempts to model the relationship between two variables by fitting a linear equation to observed data. One variable is considered to be an explanatory variable, and the other is considered to be a dependent variable. For example, we might want to relate the weights of individuals to their heights using a linear regression model.

NOTES

Before attempting to fit a linear model to observed data, we should first determine whether or not there is a relationship between the variables of interest. This does not necessarily imply that one variable causes the other (for example, higher SAT scores do not cause higher college grades), but that there is some significant association between the two variables. A scatterplot can be a helpful tool in determining the strength of the relationship between two variables. If there appears to be no association between the proposed explanatory and dependent variables (i.e., the scatterplot does not indicate any increasing or decreasing trends), then fitting a linear regression model to the data probably will not provide a useful model. A valuable numerical measure of association between two variables is the correlation coefficient, which is a value between -1 and 1 indicating the strength of the association of the observed data for the two variables.

A linear regression line has an equation of the form $Y = a + bX$, where X is the explanatory variable and Y is the dependent variable. The slope of the line is b , and a is the intercept (the value of y when $x = 0$).

Linear regression was the first type of regression analysis to be studied rigorously, and to be used extensively in practical applications. This is because models which depend linearly on their unknown parameters are easier to fit than models which are non-linearly related to their parameters and because the statistical properties of the resulting estimators are easier to determine.

Linear regression has many practical uses. Most applications fall into one of the following two broad categories:

- If the goal is prediction, forecasting, or error reduction, linear regression can be used to fit a predictive model to an observed data set of values of the response and explanatory variables. After developing such a model, if additional values of the explanatory variables are collected without an accompanying response value, the fitted model can be used to make a prediction of the response.
- If the goal is to explain variation in the response variable that can be attributed to variation in the explanatory variables, linear regression analysis can be applied to quantify the strength of the relationship between the response and the explanatory variables, and in particular to determine whether some explanatory variables may have no linear relationship with the response at all, or to identify which subsets of explanatory variables may contain redundant information about the response.

Linear regression models are often fitted using the least squares approach, but they may also be fitted in other ways, such as by minimizing the “lack of fit” in some other norm (as with least absolute deviations regression), or by minimizing a penalized version of the least squares cost function as in ridge regression (L^2 -

norm penalty) and lasso (L^1 -norm penalty). Conversely, the least squares approach can be used to fit models that are not linear models. Thus, although the terms ‘least squares’ and ‘linear model’ are closely linked, they are not synonymous.

When to use Linear Regression?

Linear Regression’s power lies in its simplicity, which means that it can be used to solve problems across various fields. At first, the data collected from the observations need to be collected and plotted along a line. If the difference between the predicted value and the result is almost the same, we can use linear regression for the problem.

Assumptions in Linear Regression

If you are planning to use linear regression for your problem then there are some assumptions you need to consider:

- The relation between the dependent and independent variables should be almost linear.
- The data is homoscedastic, meaning the variance between the results should not be too much.
- The results obtained from an observation should not be influenced by the results obtained from the previous observation.
- The residuals should be normally distributed. This assumption means that the probability density function of the residual values is normally distributed at each independent value.

You can determine whether your data meets these conditions by plotting it and then doing a bit of digging into its structure. Few properties of Regression Line are as follows:

- Regression passes through the mean of independent variable (x) as well as mean of the dependent variable (y).
- Regression line minimizes the sum of ‘Square of Residuals’. That’s why the method of Linear Regression is known as “Ordinary Least Square (OLS)”. We will discuss more in detail about Ordinary Least Square later on.
- B_1 explains the change in Y with a change in x by one unit. In other words, if we increase the value of ‘ x ’ it will result in a change in value of Y .

Finding a Linear Regression Line

Suppose, we want to predict ‘ y ’ from ‘ x ’ given in the following table and assume they are correlated as $y = B_0 + B_1x$.

NOTES

NOTES

x	y	Predicted 'y'
1	2	B_0+B_1*1
2	1	B_0+B_1*2
3	3	B_0+B_1*3
4	6	B_0+B_1*4
5	9	B_0+B_1*5
6	11	B_0+B_1*6
7	13	B_0+B_1*7
8	15	B_0+B_1*8
9	17	B_0+B_1*9
10	20	B_0+B_1*10

Where,

Std. Dev. of x	3.02765
Std. Dev. of y	6.617317
Mean of x	5.5
Mean of y	9.7
Correlation between x & y	0.989938

If the Residual Sum of Square (RSS) is differentiated with respect to B_0 & B_1 and the results equated to zero, we get the following equation:

$$B_1 = \text{Correlation} * (\text{Std. Dev. of } y / \text{Std. Dev. of } x)$$

$$B_0 = \text{Mean}(Y) - B_1 * \text{Mean}(X)$$

Putting values from table 1 into the above equations,

$$B_1 = 2.64$$

$$B_0 = -2.2$$

Hence, the least regression equation will become:

$$Y = -2.2 + 2.64*x$$

x	Y - Actual	Y - Predicted
1	2	0.44
2	1	3.08
3	3	5.72
4	6	8.36
5	9	11
6	11	13.64
7	13	16.28
8	15	18.92
9	17	21.56
10	20	24.2

As there are only 10 data points, the result is not too accurate but if we see the correlation between the predicted and actual line, it has turned out to be very high; both the lines are moving almost together.

4.4 NUMERICAL INTEGRATION AND DIFFERENTIATION

NOTES

Differentiation and integration are basic mathematical operations with a wide range of applications in many areas of science. It is therefore important to have good methods to compute and manipulate derivatives and integrals. You probably learnt the basic rules of differentiation and integration in school. These are important, and most derivatives can be computed this way. Integration however, is different, and most integrals cannot be determined with symbolic methods like the ones you learnt in school. Another complication is the fact that in practical applications a function is only known at a few points. For example, we may find the position of a car every minute via a GPS (Global Positioning System) unit, and we want to compute its speed. If the position is known as a continuous function of time, we can find the speed by differentiating this function. But when the position is only known at isolated times, this is not possible. The same applies to integrals. The solution, both when it comes to integrals that cannot be determined by the usual methods, and functions that are only known at isolated points, is to use approximate methods of differentiation and integration. In our context, these are going to be numerical methods. We are going to present a number of methods for doing numerical integration and differentiation, but more importantly, we are going to present a general strategy for deriving such methods. In this way you will not only have a number of methods available to you, but you will also be able to develop new methods, tailored to special situations that you may encounter.

We use the same general strategy for deriving both numerical integration and numerical differentiation methods. The basic idea is to evaluate a function at a few points, find the polynomial that interpolates the function at these points, and use the derivative or integral of the polynomial as an approximation to the function. This technique also allows us to keep track of the so-called truncation error, the mathematical error committed by integrating or differentiating the polynomial instead of the function itself. However, when it comes to round off error, we have to treat differentiation and integration differently: Numerical integration is very insensitive to round-off errors, while numerical differentiation behaves in the opposite way; it is very sensitive to round-off errors.

A simple method for numerical differentiation we start by studying numerical differentiation. We first introduce the simplest method, derive its error, and its sensitivity to round-off errors. The procedure used here for deriving the method and analyzing the error is used over again in later sections to derive and analyze additional methods. Let us first make it clear what numerical differentiation is.

Let f be a given function that is only known at a number of isolated points. The problem of numerical differentiation is to compute an approximation to the derivative f' off by suitable combinations of the known values off. A typical example

NOTES

is that f is given by a computer program (more specifically a function, procedure or method, depending on your choice of programming language), and you can call the program with a floating-point argument x and receive back a floating-point approximation of $f(x)$. The challenge is to compute an approximation to $f'(a)$ for some real number a when the only aid we have at our disposal is the program to compute values. The basic idea Since we are going to compute derivatives, we must be clear about they are defined. Recall that $f'(a)$ is defined by $f'(a) = \lim_{h \rightarrow 0} \frac{f(a+h) - f(a)}{h}$. In the following we will assume that this limit exists; i.e., that f is differentiable. We immediately have a natural approximation to $f'(a)$; we simply pick a positive h and use the approximation $f'(a) \approx \frac{f(a+h) - f(a)}{h}$.

Note that this corresponds to approximating f by the straight line p_1 that interpolates f at a and $a-h$, and then using $p_1'(a)$ as an approximation to $f'(a)$. The derivative off at a can be approximated by $f'(a) \approx \frac{f(a+h) - f(a)}{h}$. In a practical situation, the number a would be given, and we would have to locate the two nearest values a_1 and a_2 to the left and right of a such that $f(a_1)$ and $f(a_2)$ can be found. Then we would use the approximation $f'(a) \approx \frac{f(a_2) - f(a_1)}{a_2 - a_1}$.

Scalar Ordinary Differential Equations

Ordinary differential equations (ODEs) written in the abstract form is as follows:

$$u'(t) = f(u(t), t). \quad (1)$$

There is an infinite number of solutions to such an equation, so to make the solution $u(t)$ unique, we must also specify an initial condition

$$u(0) = U_0. \quad (2)$$

Given $f(u, t)$ and U_0 , our task is to compute $u(t)$.

At first sight, (1) is only a first-order differential equation, since only u' and not higher-order derivatives like u'' are present in the equation. However, equations with higher-order derivatives can also be written on the abstract form (1) by introducing auxiliary variables and interpreting u and f as vector functions. This rewrite of the original equation leads to a system of first-order differential equations and will be treated in the section Systems of ordinary differential equations. The bottom line is that a very large family of differential equations can be written as (1). Forthcoming examples will provide evidence.

We shall first assume that $u(t)$ is a *scalar function*, meaning that it has one number as value, which can be represented as a `float` object in Python. We then refer to (1) as a *scalar differential equation*. The counterpart *vector function* means that u is a vector of scalar functions and the equation is known as a *system of ODEs* (also known as a *vector ODE*). The value of a vector function is a list or array in a program. Systems of ODEs are treated in the section Systems of ordinary differential equations.

Examples on right-hand-side functions

To write a specific differential equation on the form (1) we need to identify what the off function is. The equation will be:

$$y^2 y' = x, y(0) = Y,$$

with $y(x)$ as the unknown function. First, we need to introduce u and t as new symbols: $u=y$, $t=x$. This gives the equivalent equation $u^2 u' = t$ and the initial condition $u(0) = Y$. Second, the quantity u^2 must be isolated on the left-hand side of the equation in order to bring the equation on the form (1). Dividing by u^2 gives

$$u' = tu^{-2}.$$

This fits the form (1), and the $f(u,t)$ function is simply the formula involving u and t on the right-hand side:

$$f(u,t) = tu^{-2}.$$

The t parameter is very often absent on the right-hand side such that f involves u only.

Let us list some common scalar differential equations and their corresponding f functions. Exponential growth of money or populations is governed by

$$u' = \alpha u, \tag{3}$$

where $\alpha > 0$ is a given constant expressing the growth rate of u . In this case,

$$f(u,t) = \alpha u. \tag{4}$$

A related model is the logistic ODE for growth of a population under limited resources:

$$u' = \alpha u(1 - u/R), \tag{5}$$

Where $\alpha > 0$ is the initial growth rate and R is the maximum possible value of u . The corresponding f is

$$f(u,t) = \alpha u(1 - u/R). \tag{6}$$

Radioactive decay of a substance has the model

$$u' = -\alpha u, \tag{7}$$

Where $\alpha > 0$ is the rate of decay of u . Here,

$$f(u,t) = -\alpha u. \tag{8}$$

A body falling in a fluid can be modeled by

$$u' + b|u|u = g, \tag{9}$$

where $b > 0$ models the fluid resistance, g is the acceleration of gravity, and u is the body's velocity. By solving for u' we find

$$f(u,t) = -b|u|u + g. \tag{10}$$

Finally, Newton's law of cooling is the ODE

$$u' = -h(u - s), \tag{11}$$

where u is the temperature of a body, $h > 0$ is a proportionality constant, normally to be estimated from experiments, and s is the temperature of the surroundings. Obviously,

$$f(u,t) = -h(u - s). \tag{12}$$

NOTES

NOTES

Forward Euler Scheme

Our task now is to define numerical methods for solving equations of the form (1). The simplest such method is the Forward Euler scheme. Equation (1) is to be solved for $t \in (0, T]$, and we seek the solution u at discrete time points.

$$t_i = i\Delta t, \quad i=1, 2, \dots, n.$$

Clearly, $t_n = n\Delta t = T$, determining the number of points n as $T/\Delta t$. The corresponding values $u(t_i)$ are often abbreviated as u_i , just for notational simplicity.

Equation (1) is to be fulfilled at all-time points $t \in (0, T]$. However, when we solve (1) numerically, we only require the equation to be satisfied at the discrete time points t_1, t_2, \dots, t_n . That is,

$$u'(t_k) = f(u(t_k), t_k),$$

for $k=1, \dots, n$. The fundamental idea of the Forward Euler scheme is to approximate $u'(t_k)$ by a one-sided, forward difference:

$$u'(t_k) \approx \frac{u(t_{k+1}) - u(t_k)}{\Delta t} = \frac{u_{k+1} - u_k}{\Delta t}.$$

This removes the derivative and leaves us with the equation

$$\frac{u_{k+1} - u_k}{\Delta t} = f(u_k, t_k).$$

We assume that u_k is already computed, so that the only unknown in this equation is u_{k+1} , which we can solve for:

$$u_{k+1} = u_k + \Delta t f(u_k, t_k). \tag{13}$$

This is the Forward Euler scheme for a scalar first-order differential equation $u' = f(u, t)$.

Equation (13) has a recursive nature. We start with the initial condition, $u_0 = U_0$, and compute u_1 as

$$u_1 = u_0 + \Delta t f(u_0, t_0).$$

Then we can continue with

$$u_2 = u_1 + \Delta t f(u_1, t_1),$$

and then with u_3 and so forth. This recursive nature of the method also demonstrates that we *must* have an initial condition – otherwise the method cannot start.

Function Implementation

The next task is to write a general piece of code that implements the forward Euler scheme (13). The complete original (continuous) mathematical problem is stated as:

$$u' = f(u, t), \quad t \in (0, T], \quad u(0) = U_0, \tag{14}$$

while the discrete numerical problem reads

$$u(0) = U_0, \tag{15}$$

$$u_{k+1} = u_k + \Delta t f(u_k, t_k), \quad t_k = k\Delta t, \quad k=1, \dots, n, \quad n=T/\Delta t \tag{16}$$

We see that the input data to the numerical problem consist of f , U_0 , T , and Δt or n . The output consists of u_1, u_2, \dots, u_n and the corresponding set of time points t_1, t_2, \dots, t_n .

Let us implement the Forward Euler method in a function `ForwardEuler` that takes f , U_0 , T , and n as input, and that returns u_0, \dots, u_n and t_0, \dots, t_n .

```
import numpy as np
def ForwardEuler(f, U0, T, n):
    """Solve u' = f(u,t), u(0) = U0, with n steps until t = T."""
    t = np.zeros(n+1)
    u = np.zeros(n+1) # u[k] is the solution at time t[k]
    u[0] = U0
    t[0] = 0
    dt = T/float(n)
    for k in range(n):
        t[k+1] = t[k] + dt
        u[k+1] = u[k] + dt*f(u[k], t[k])
    return u, t
```

Note: the close correspondence between the implementation and the mathematical specification of the problem is to be solved. The argument f to the `ForwardEuler` function must be a Python function $f(u, t)$ implementing the $f(u, t)$ function in the differential equation. In fact, f is the definition of the equation to be solved. For example, we may solve $u' = u$ for $t \in (0, 3)$, with $u(0) = 1$, and $\Delta t = 0.1$ by the following code utilizing the `ForwardEuler` function:

```
def f(u, t):
    return u
u, t = ForwardEuler(f, U0=1, T=4, n=20)
```

With the u and t arrays, we can easily plot the solution or perform data analysis on the numbers.

4.5 MONTE CARLO AND MOLECULAR DYNAMICS

Monte Carlo algorithm is a randomized algorithm whose output may be incorrect with a certain (typically small) probability. Two examples of such algorithms are Karger–Stein algorithm and Monte Carlo algorithm for minimum Feedback arc set.

The name refers to the grand casino in the Principality of Monaco at Monte Carlo, which is well-known around the world as an icon of gambling. The term ‘Monte Carlo’ was first introduced in 1947 by Nicholas Metropolis.

NOTES

NOTES

Las Vegas algorithms are a dual of Monte Carlo algorithms that never return an incorrect answer. However, they may make random choices as part of their working. As a result, the time taken might vary between runs even with the same input. If there is a procedure for verifying whether the answer given by a Monte Carlo algorithm is correct, and the probability of a correct answer is bounded above zero, then with probability one running the algorithm repeatedly while testing the answers will eventually give a correct answer. Whether this process is a Las Vegas algorithm depends on whether halting with probability one is considered to satisfy the definition.

One-sided vs Two-sided error

The answer returned by a deterministic algorithm is always expected to be correct, this is not the case for Monte Carlo algorithms. For decision problems, these algorithms are generally classified as either false-biased or true-biased. A false-biased Monte Carlo algorithm is always correct when it returns false; a true-biased algorithm is always correct when it returns true. While this describes algorithms with one-sided errors, others might have no bias; these are said to have two-sided errors. The answer they provide (either true or false) will be incorrect, or correct, with some bounded probability.

For instance, the Solovay–Strassen primality test is used to determine whether a given number is a prime number. It always answers true for prime number inputs; for composite inputs, it answers false with probability at least $1/2$ and true with probability less than $1/2$. Thus, false answers from the algorithm are certain to be correct, whereas the true answers remain uncertain; this is said to be a $1/2$ -correct false-biased algorithm.

Amplification

For a Monte Carlo algorithm with one-sided errors, the failure probability can be reduced (and the success probability amplified) by running the algorithm k times. Consider again the Solovay Strassen algorithm which is $1/2$ -correct false-biased. One may run this algorithm multiple times returning a false answer if it reaches a false response within k iterations, and otherwise returning true. Thus, if the number is prime then the answer is always correct, and if the number is composite then the answer is correct with probability at least $1 - (1/2)^k = 1 - 2^{-k}$.

For Monte Carlo decision algorithms with two-sided error, the failure probability may again be reduced by running the algorithm k times and returning the majority function of the answers.

Complexity classes

The complexity class BPP describes decision problems that can be solved by polynomial-time Monte Carlo algorithms with a bounded probability of two-sided errors, and the complexity class RP describes problems that can be solved by a Monte Carlo algorithm with a bounded probability of one-sided error: if the correct answer is false, the algorithm always says so, but it may answer false incorrectly for some instances where the correct answer is true. In contrast, the complexity class ZPP describes problems solvable by polynomial expected time Las Vegas algorithms. $ZPP \subseteq RP \subseteq BPP$, but it is not known whether any of these complexity

classes is distinct from each other; that is, Monte Carlo algorithms may have more computational power than Las Vegas algorithms, but this has not been proven. Another complexity class, PP, describes decision problems with a polynomial-time Monte Carlo algorithm that is more accurate than flipping a coin but where the error probability cannot necessarily be bounded away from 1/2.

NOTES

Check Your Progress

1. Define term programming.
2. What do you understand by the term linear regression?
3. State about the differentiation and integration.

4.6 INTRODUCTION TO SOFTWARE PACKAGES

4.6.1 MATLAB

MATLAB (An abbreviation of “MATrix LABoratory”) is a proprietary multi-paradigm programming language and numeric computing environment developed by MathWorks. It allows matrix manipulations, plotting of functions and data, implementation of algorithms, creation of user interfaces, and interfacing with programs written in other languages.

Although, MATLAB is intended primarily for numeric computing, an optional toolbox uses the MuPAD symbolic engine allowing access to symbolic computing abilities. An additional package, Simulink, adds graphical multi-domain simulation and model-based design for dynamic and embedded systems.

Commercial Development

MATLAB was first released as a commercial product in 1984 at the Automatic Control Conference in Las Vegas. MathWorks, Inc. was founded to develop the software and the MATLAB programming language was released. The first MATLAB sale was the following year, when Nick Trefethen from the Massachusetts Institute of Technology bought ten copies. By the end of the 1980s, several hundred copies of MATLAB had been sold to universities for student use. The software was popularized largely thanks to toolboxes created by experts in various fields for performing specialized mathematical tasks. Many of the toolboxes were developed as a result of Stanford students that used MATLAB in academia, then brought the software with them to the private sector. Over time, MATLAB was re-written for early operating systems created by Digital Equipment Corporation, VAX, Sun Microsystems, and for Unix PCs. Version 3 was released in 1987. The first MATLAB compiler was developed by Stephen C. Johnson in the 1990s.

In 2000, MathWorks added a FORTRAN-based library for linear algebra in MATLAB 6, replacing the software’s original LINPACK and EISPACK subroutines that were in C. MATLAB’s Parallel Computing Toolbox was released at the 2004 Supercomputing Conference and support for graphics processing units

(GPUs) was added to it in 2010. As of 2020, MATLAB has more than 4 million users worldwide. MATLAB users come from various backgrounds of engineering, science, and economics.

NOTES

SYNTAX

The MATLAB application is built around the MATLAB programming language. Common usage of the MATLAB application involves using the “Command Window” as an interactive mathematical shell or executing text files containing MATLAB code.

Variables

Variables are defined using the assignment operator, =. MATLAB is a weakly typed programming language because types are implicitly converted. It is an inferred typed language because variables can be assigned without declaring their type, except if they are to be treated as symbolic objects, and their type can change. Values can come from constants, from computation involving values of other variables, or from the output of a function. Consider the example given below:

```
>> x = 17
x =
    17

>> x = 'hat'
x =
    hat

>> x = [3*4, pi/2]
x =
    12.0000    1.5708

>> y = 3*sin(x)
y =
   -1.6097    3.0000
```

Vectors and matrices

A simple array is defined using the colon syntax: *initial:increment:terminator*. For instance:

```
>> array = 1:2:9
array =
    1    3    5    7    9
```

Defines a variable named `array` (or assigns a new value to an existing variable with the name `array`) consisting of the values 1, 3, 5, 7, and 9. The array starts at 1 (the *initial* value), increments with each step from the previous value by 2 (the *increment* value), and stops once it reaches (or is about to exceed)

9 (the *terminator* value). The *increment* value can actually be left out of this syntax (along with one of the colons), to use a default value of 1.

```
>> ari = 1:5
ari =
1 2 3 4 5
```

Assigns to the variable named `ari` an array with the values 1, 2, 3, 4, and 5, since the default value of 1 is used as the increment.

Indexing is one-based,^[37] which is the usual convention for matrices in mathematics, unlike zero-based indexing commonly used in other programming languages such as C, C++, and Java.

Matrices can be defined by separating the elements of a row with blank space or comma and using a semicolon to terminate each row. The list of elements should be surrounded by square brackets `[]`. Parentheses `()` are used to access elements and subarrays (they are also used to denote a function argument list).

```
>> A = [16 3 2 13; 5 10 11 8; 9 6 7 12; 4 15 14 1]
A =
16 3 2 13
5 10 11 8
9 6 7 12
4 15 14 1
```

```
>> A(2,3)
ans =
11
```

Sets of indices can be specified by expressions such as `2 : 4`, which evaluates to `[2, 3, 4]`. For example, a submatrix taken from rows 2 through 4 and columns 3 through 4 can be written as:

```
>> A(2:4,3:4)
ans =
11 8
7 12
14 1
```

A square identity matrix of size n can be generated using the function `eye`, and matrices of any size with zeros or ones can be generated with the functions `zeros` and `ones`, respectively.

```
>> eye(3,3)
ans =
1 0 0
0 1 0
0 0 1
```

NOTES

NOTES

```
>> zeros(2,3)
```

```
ans =
```

```
0 0 0
```

```
0 0 0
```

```
>> ones(2,3)
```

```
ans =
```

```
1 1 1
```

```
1 1 1
```

Transposing a vector or a matrix is done either by the function `transpose` or by adding dot-prime after the matrix (without the dot, prime will perform conjugate transpose for complex arrays). For example:

```
>> A = [1 ; 2], B = A.', C = transpose(A)
```

```
A =
```

```
1 2
```

```
B =
```

```
1 2
```

```
C =
```

```
1 2
```

```
>> D = [0 3 ; 1 5], D.'
```

```
D =
```

```
0 3
```

```
1 5
```

```
ans =
```

```
0 1
```

```
3 5
```

Most functions accept arrays as input and operate element-wise on each element. For example, `mod(2*J, n)` will multiply every element in `J` by 2, and then reduce each element modulo `n`. MATLAB does include standard `for` and `while` loops, but (as in other similar applications such as R), using the vectorized notation is encouraged and is often faster to execute. The following code, excerpted from the function `magic.m`, creates a magic square `M` for odd values of `n` (MATLAB function `meshgrid` is used here to generate square matrices `I` and `J` containing `1:n`):

```
[J,I]=meshgrid(1:n);
```

```
A = mod(I + J - (n + 3) / 2, n);
```

```
B = mod(I + 2 * J - 2, n);
```

```
M = n * A + B + 1;
```


Classes and Object-Oriented Programming

MATLAB supports object-oriented programming including classes, inheritance, virtual dispatch, packages, pass-by-value semantics, and pass-by-reference semantics. However, the syntax and calling conventions are significantly different from other languages. MATLAB has value classes and reference classes, depending on whether the class has *handle* as a super-class (for reference classes) or not (for value classes).

Method call behavior is different between value and reference classes. For example, a call to a method:

```
object.method();
```

It can alter any member of *object* only if *object* is an instance of a reference class. Otherwise value class methods must return a new instance if it needs to modify the object.

An example of a simple class is provided below:

```
classdef Hello
    methods
    function greet(obj)
        disp('Hello!')
    end
    end
end
```

When stored into a file named `hello.m`, this can be executed with the following commands:

```
>> x = Hello();
>> x.greet();
Hello!
```

4.6.2 EasyPlot

EasyPlot for MapInfo is a simple-to-use utility that allows for easy and professional MapInfo Printing, simply customize templates to your requirements, either for the production of high quality printed maps or for bulk production of batch processed maps or images. An example program to create a simple plot is given below.

```
import matplotlib.pyplot as plt
import numpy as np

# Data for plotting
t = np.arange(0.0, 2.0, 0.01)
s = 1 + np.sin(2 * np.pi * t)
```

NOTES

NOTES

```
fig, ax = plt.subplots()
ax.plot(t, s)

ax.set(xlabel='time (s)', ylabel='voltage (mV)',
       title='About as simple as it gets, folks')
ax.grid()

fig.savefig("test.png")
plt.show()
```

4.6.3 FoxPro

FoxPro is programming language as well as it develops an application development environment. It is also a database manager. It has different tools which helps in development and design of applications in very easy way. It also provides platform for new databases and to perform new tasks on that databases. In earlier times, this language is used to be most powerful and most used language and database manager. But with the time, many new languages, database, programming and many software's were developed. Nowadays, this language is not used much but still it is most powerful language and database management system. It has various versions- DBASE I, DBASE II, DBASE III, DBASE III+, DBASE IV, FOXBASE, FOXBASE +, FOXPRO. The version 2.5 of FoxPro and versions released after it is can worked with Windows and DOS operating system. Visual Foxpro, the successor to Foxpro programming, is often used for application development.

Features of FoxPro

Following are some of the features of FoxPro.

- 1) It acts as interactive database manager.
- 2) It acts as programming language.
- 3) It provides application development environment.
- 4) It can be used to create database.
- 5) It can be used to Add/Remove data from database.
- 6) It is used to manipulate existing data.

Advantages

- 1) It supports the object-oriented programming syntax.
- 2) It provides visual tools.
- 3) It also supports multimedia database.
- 4) It is inexpensive.

Disadvantages

- 1) It is not applicable for low-level programming.

- 2) It is not suitable for users or programmers *with* little or no programming experience.
- 3) It has character mode only.

Data Types used in FoxPro

(i) Character

A-Z, a-z, or any arithmetic operation, maximum of 250 characters can be stored and default width for the same is 10.

(ii) Number

It is used to store numerical values, can store decimal values not exceeding 20.

(iii) Float

It is used to store floating values. Default width is 10.

(iv) Date

It is used to store the date. By default, its width is 10 characters.

(v) Logical

It is used to store the Boolean Values like true/false and yes/no.

(vi) Memo

It is special type of data-type in which large amount of text, images, sound, audio, video etc. can be stored. It is powerful data-type because when we store any data, it stores the data in different file, whose files extension is .fpt.

(vii) General

It has same characteristics as Memo and with that it can handle and store much larger data in word processing systems. The advantage of this data-type is that, we can increase the size and can store large amount of data.

Visual FoxPro originated as a member of the class of languages commonly referred to as “x Base” languages, which have syntax based on the dBase programming language. Other members of the xBase language family include Clipper and Recital (database).

Visual FoxPro, commonly abbreviated as VFP, is tightly integrated with its own relational database engine, which extends FoxPro’s xBase capabilities to support SQL query and data manipulation. Unlike most database management systems, Visual FoxPro is a full-featured, dynamic programming language that does not require the use of an additional general-purpose programming environment. It can be used to write not just traditional “fat client” applications, but also middleware and web applications.

In late 2002, it was demonstrated that Visual FoxPro can run on Linux under the Wine Windows compatibility suite. In 2003, this led to complaints by Microsoft: it was claimed that the deployment of runtime FoxPro code on non-Windows machines violates the End User License Agreement.

Visual FoxPro had a rapid rise and fall in popularity as measured by the TIOBE Programming Community Index. In December 2005, VFP broke into

NOTES

the top 20 for the first time. In June 2006 it peaked at position 12, making it (at the time) a “B” language. As of October 2019, Visual FoxPro holds position 51 on the TIOBE index.

NOTES

In March 2007, Microsoft announced that there will be no VFP 10, thus making VFP9 (released to manufacturing on December 17, 2004) the last commercial VFP release from Microsoft. Service Pack 2 for Microsoft Visual FoxPro 9.0 was released on October 16, 2007. The support of Version 9 ended on January 13, 2015.

Check Your Progress

4. Write the use of MATLAB.
5. How the variables are defined in MATLAB?
6. What is FoxPro?

4.7 ANSWERS TO ‘CHECK YOUR PROGRESS

1. Programming is the process of designing and creating the programs to perform a user specific task.
2. Linear regression is one of the machine learning algorithms where the result is predicted by the use of known parameters which are correlated with the output.
3. Differentiation and integration are basic mathematical operations with a wide range of applications in many areas of science. It is therefore important to have good methods to compute and manipulate derivatives and integrals.
4. MATLAB allows matrix manipulations, plotting of functions and data, implementation of algorithms, creation of user interfaces, and interfacing with programs written in other languages.
5. Variables are defined using the assignment operator, =. MATLAB is a weakly typed programming language because types are implicitly converted. It is an inferred typed language because variables can be assigned without declaring their type, except if they are to be treated as symbolic objects, and that their type can change.
6. FoxPro is programming language as well as it develops an application development environment. It is also Database Manager. It is also called Relational Database Management System because of its various features.

4.8 SUMMARY

- In general, the programmer’s job is to convert problem solutions into instructions for the computer. That is, the programmer prepares the instructions of a computer program and runs those instructions on the computer, tests the program to see if it is working properly, and makes corrections to the program.

- In statistics and machine learning, linear regression is one of the most popular and well understood algorithms. Most data science enthusiasts and machine learning fanatics begin their journey with linear regression algorithms.
- Linear Regression is one of the machine learning algorithms where the result is predicted by the use of known parameters which are correlated with the output.
- Linear regression attempts to model the relationship between two variables by fitting a linear equation to observed data. One variable is considered to be an explanatory variable, and the other is considered to be a dependent variable.
- Differentiation and integration are basic mathematical operations with a wide range of applications in many areas of science. It is therefore important to have good methods to compute and manipulate derivatives and integrals.
- Monte Carlo algorithm is a randomized algorithm whose output may be incorrect with a certain (typically small) probability. Two examples of such algorithms are Karger–Stein algorithm and Monte Carlo algorithm for minimum Feedback arc set.
- (An abbreviation of “MATrix LABoratory”) is a proprietary multi-paradigm programming language and numeric computing environment developed by MathWorks.
- MATLAB allows matrix manipulations, plotting of functions and data, implementation of algorithms, creation of user interfaces, and interfacing with programs written in other languages.
- The MATLAB application is built around the MATLAB programming language. Common usage of the MATLAB application involves using the “Command Window” as an interactive mathematical shell or executing text files containing MATLAB code.
- Variables are defined using the assignment operator, =. MATLAB is a weakly typed programming language because types are implicitly converted. It is an inferred typed language because variables can be assigned without declaring their type, except if they are to be treated as symbolic objects, and that their type can change.
- EasyPlot for MapInfo is a simple-to-use utility that allows for easy and professional MapInfo Printing, simply customize templates to your requirements, either for the production of high quality printed maps or for bulk production of batch processed maps or images.
- FoxPro is programming language as well as it develops an application development environment. It is also database manager. It is also called Relational Database Management System (RDBMS) because of its various features.

NOTES

NOTES

4.9 KEY TERMS

- **Linear Regression:** It is a linear approach for modelling the relationship between a scalar response and one or more explanatory variables.
- **MATLAB:** It allows matrix manipulations, plotting of data and functions, creation of user interfaces, implementation of algorithms and interfacing with programs written in other languages.
- **FoxPro:** It is text-based procedurally oriented programming language as well as it develops an application development environment.

4.10 SELF-ASSESSMENT QUESTIONS AND EXERCISES

Short-Answer Questions

1. How will you define programming?
2. What are the different phases of programming?
3. Write the use of linear regression.
4. State about the complexity classes in Monte Carlo algorithm.
5. What are the uses of MATLAB?
6. What are the different functions used in vector and matrices in MATLAB?
7. Write the advantages and disadvantages of FoxPro.

Long-Answer Questions

1. Describe the process of programming along with its phases.
2. Briefly explain the linear regression giving appropriate examples.
3. Discuss briefly how to find a line linear regression with the help of appropriate example.
4. Describe briefly scalar ordinary differential equations with the help of example.
5. Analyze the Monte Carlo algorithm giving appropriate examples.
6. What do you mean by EASYPLOT? Discuss.
7. Discuss about the FoxPro and its features.

4.11 FURTHER READING

Dey, Pradip and Manas Ghosh. *Computer Fundamentals and Programming in C*. New Delhi: Oxford Higher Education, 2006.

Bronson, Gary J. *A First Book of ANSI C*, 3rd edition. California: Thomson, Brooks Cole, 2000.

Kanetkar, Yashwant. *Understanding Pointers in C*. New Delhi: BPB Publication, 2001.

Kanetkar, Yashwant. *Let us C*. New Delhi: BPB Publication, 1999.

Kernighan, Brian W. and Dennis Ritchie. *C Programming Language*, 2nd edition. New Jersey: Prentice Hall, 1988.

Foster, W. D. and L. S. Foster. *C by Discovery*. Boston: Addison-Wesley, 2005.

Kanetkar, Yashwant. *Working with C*. New Delhi: BPB Publication, 2003.

Horton, Ivor. *Instant C Programming*. New Jersey: Wrox Press (John Willey & Sons), 1995.

Lawlor, Steven C. *The Art of Programming Computer Science with 'C'*. New Jersey: West Publishing Company, 1996.

NOTES

