Programming Practices and Techniques

Are you ☐☐☐☐☐☐☐☐☐ with Onlinevarsity.com?	R G E T R E E S I D
Did you ☐☐☐☐☐☐☐☐ this ebook?	D O O W L N A D
Do you have a ☐☐☐☐☐ copy of this ebook?	L G A E L
Are you a victim of ☐☐☐☐☐☐?	P C I A R Y

Answers: REGISTERED, DOWNLOAD, LEGAL, PIRACY

Download a **LEGAL** copy of this ebook which you can say is **mine**

because **PIRACY** is a **crime**

Programming Practices and Techniques Learner's Guide

APTECH LIMITED
Contact E-mail: ov-support@onlinevarsity.com

Edition 2 - 2013

Aptech®
Unleash your potential

Aptech™
COMPUTER EDUCATION
Unleash your potential

Aptech™
COMPUTER EDUCATION
Unleash your potential

Dear Learner,

We congratulate you on your decision to pursue an Aptech Worldwide course.

Aptech Ltd. designs its courses using a sound instructional design model – from conceptualization to execution, incorporating the following key aspects:

- Scanning the user system and needs assessment

Needs assessment is carried out to find the educational and training needs of the learner

Technology trends are regularly scanned and tracked by core teams at Aptech Ltd. TAG* analyzes these on a monthly basis to understand the emerging technology training needs for the Industry.

An annual Industry Recruitment Profile Survey# is conducted during August - October to understand the technologies that Industries would be adapting in the next 2 to 3 years. An analysis of these trends & recruitment needs is then carried out to understand the skill requirements for different roles & career opportunities.

The skill requirements are then mapped with the learner profile (user system) to derive the Learning objectives for the different roles.

- Needs analysis and design of curriculum

The Learning objectives are then analyzed and translated into learning tasks. Each learning task or activity is analyzed in terms of knowledge, skills and attitudes that are required to perform that task. Teachers and domain experts do this jointly. These are then grouped in clusters to form the subjects to be covered by the curriculum.

In addition, the society, the teachers, and the industry expect certain knowledge and skills that are related to abilities such as *learning-to-learn, thinking, adaptability, problem solving, positive attitude etc*. These competencies would cover both cognitive and affective domains.

A precedence diagram for the subjects is drawn where the prerequisites for each subject are graphically illustrated. The number of levels in this diagram is determined by the duration of the course in terms of number of semesters etc. Using the precedence diagram and the time duration for each subject, the curriculum is organized.

- Design & development of instructional materials

The content outlines are developed by including additional topics that are required for the completion of the domain and for the logical development of the competencies identified. Evaluation strategy and scheme is developed for the subject. The topics are arranged/organized in a meaningful sequence.

The detailed instructional material – Training aids, Learner material, reference material, project guidelines, etc.- are then developed. Rigorous quality checks are conducted at every stage.

- Strategies for delivery of instruction

  Careful consideration is given for the integral development of abilities like thinking, problem solving, learning-to-learn etc. by selecting appropriate instructional strategies (training methodology), instructional activities and instructional materials.

  The area of IT is fast changing and nebulous. Hence considerable flexibility is provided in the instructional process by specially including creative activities with group interaction between the students and the trainer. The positive aspects of Web based learning –acquiring information, organizing information and acting on the basis of insufficient information are some of the aspects, which are incorporated, in the instructional process.

- Assessment of learning

  The learning is assessed through different modes – tests, assignments & projects. The assessment system is designed to evaluate the level of knowledge & skills as defined by the learning objectives.

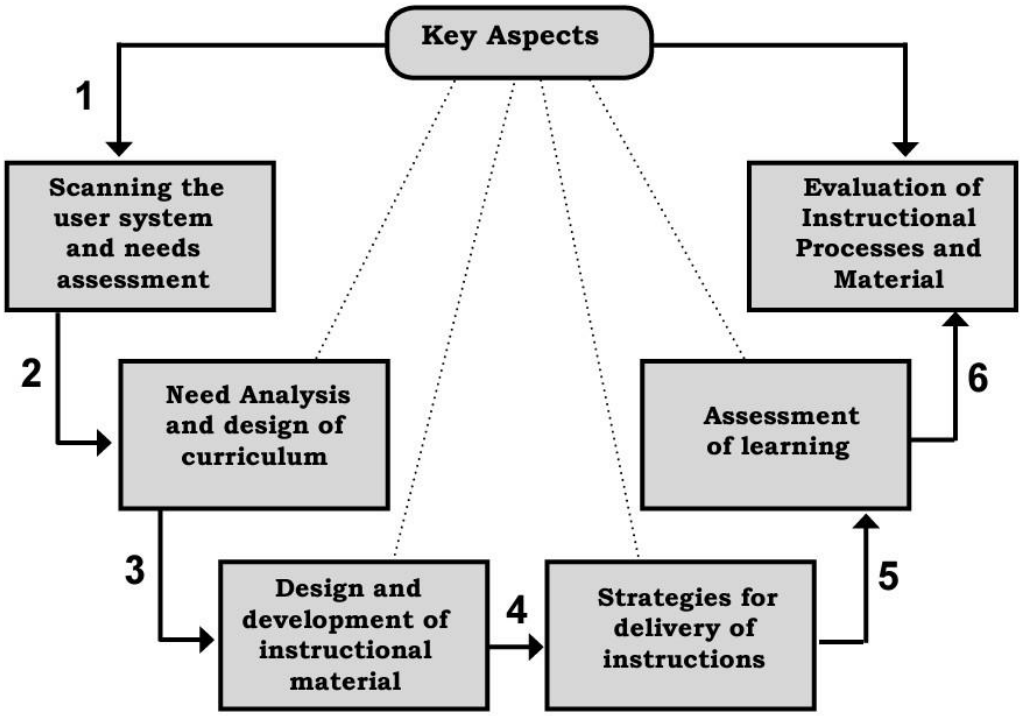
- Evaluation of instructional process and instructional materials

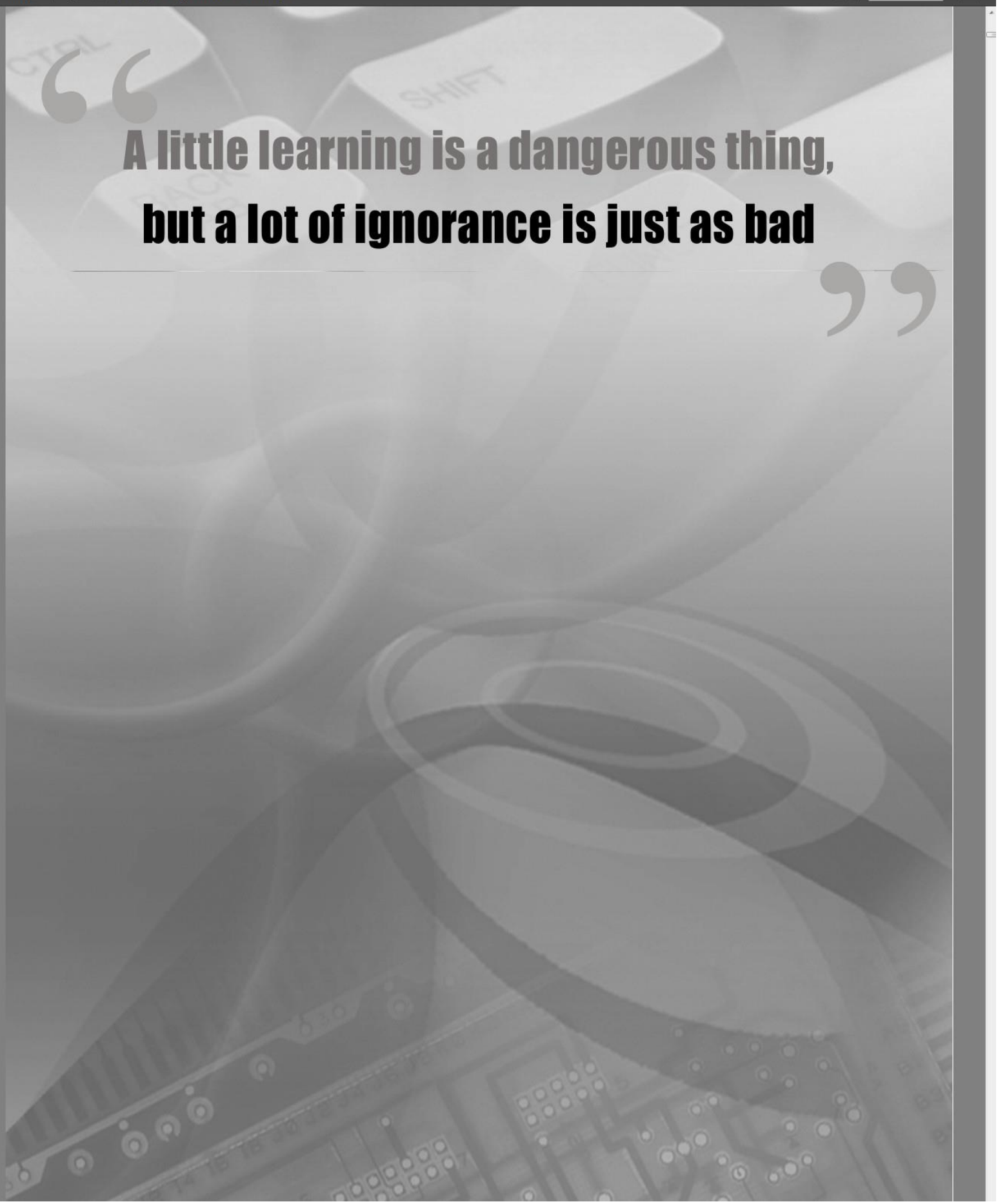
  The instructional process is backed by an elaborate monitoring system to evaluate - on-time delivery, understanding of a subject module, ability of the instructor to impart learning. As an integral part of this process, we request you to kindly send us your feedback in the reply pre-paid form appended at the end of each module.

*TAG – Technology & Academics Group comprises of members from Aptech Ltd., professors from reputed Academic Institutions, Senior Managers from Industry, Technical gurus from Software Majors & representatives from regulatory organizations/forums.

Technology heads of Aptech Ltd. meet on a monthly basis to share and evaluate the technology trends. The group interfaces with the representatives of the TAG thrice a year to review and validate the technology and academic directions and endeavors of Aptech Ltd.

Aptech New Products Design Model




A little learning is a dangerous thing,
but a lot of ignorance is just as bad

Preface

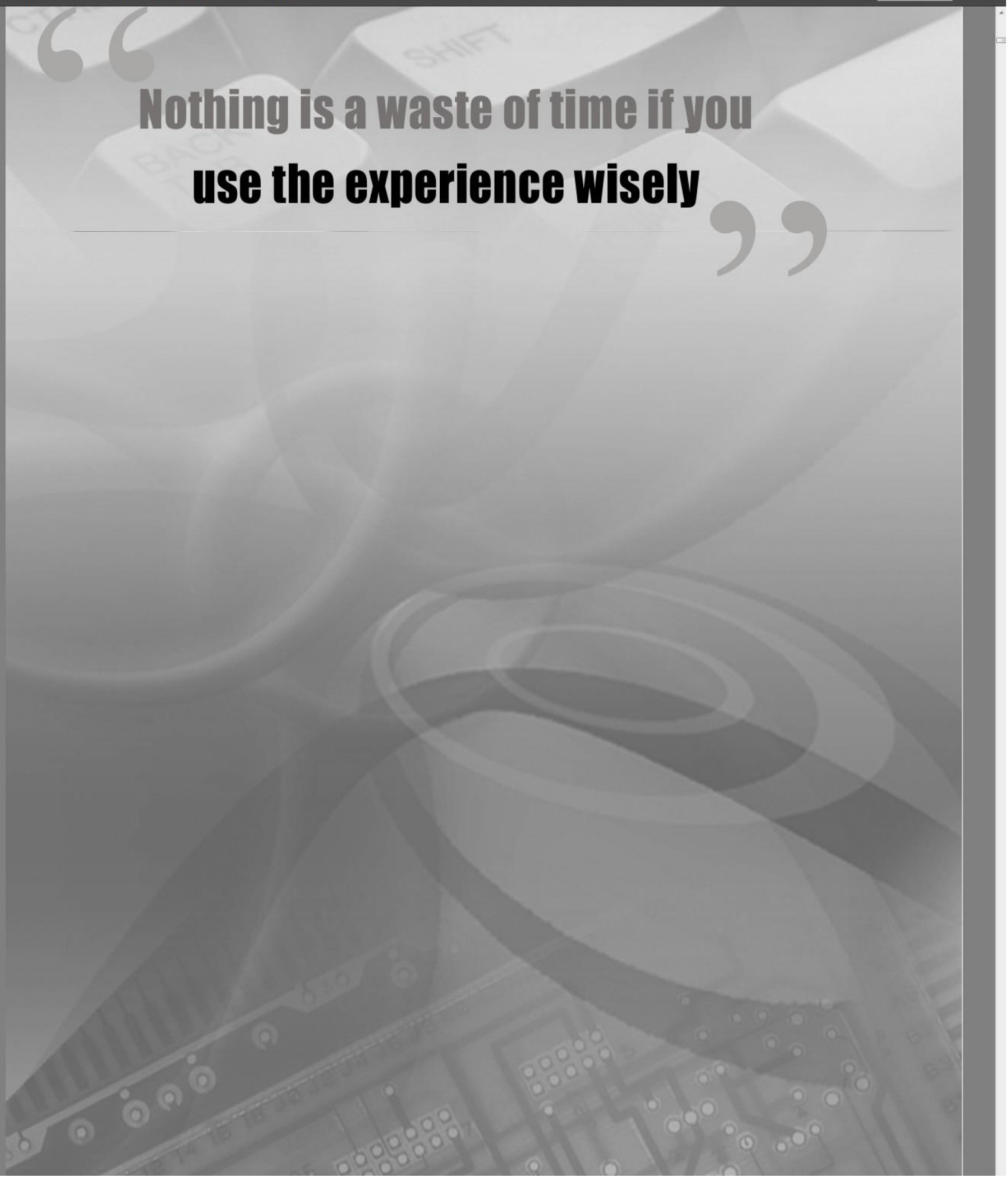
Computers have made our life easy. Computers can complete different tasks for us, but computer by themselves are not intelligent. They have to be instructed or programmed to perform the tasks. Over the years, several programming languages have been developed to help the programmers to get the computer to perform the required tasks. While the programming languages have been varied in terms of the keywords they use, the basic approach to writing a program has remained more or less the same.

Keeping this in mind, we have designed this book to equip you with the skills required to build the skeleton of a program using algorithms. After this, the programmer needs to build the 'flesh' around it using the particular syntax of a specific language. In this book, we will be discussing the basic approach to write a program. This is done by using an algorithm which is written independently of any programming language. In addition, the fundamentals of programming are also specified in the book. After reading this book, you will be able to develop algorithms, which can be implemented using any programming language. Therefore, this book provides pre-requisites for learning any programming language.

The knowledge and information in this book is the result of the concentrated effort of the Design Team, which is continuously striving to bring to you the latest, the best and the most relevant subject matter in Information Technology. As a part of Aptech's quality drive, this team does intensive research and curriculum enrichment to keep it in line with industry trends and learner requirements.

We will be glad to receive your suggestions.

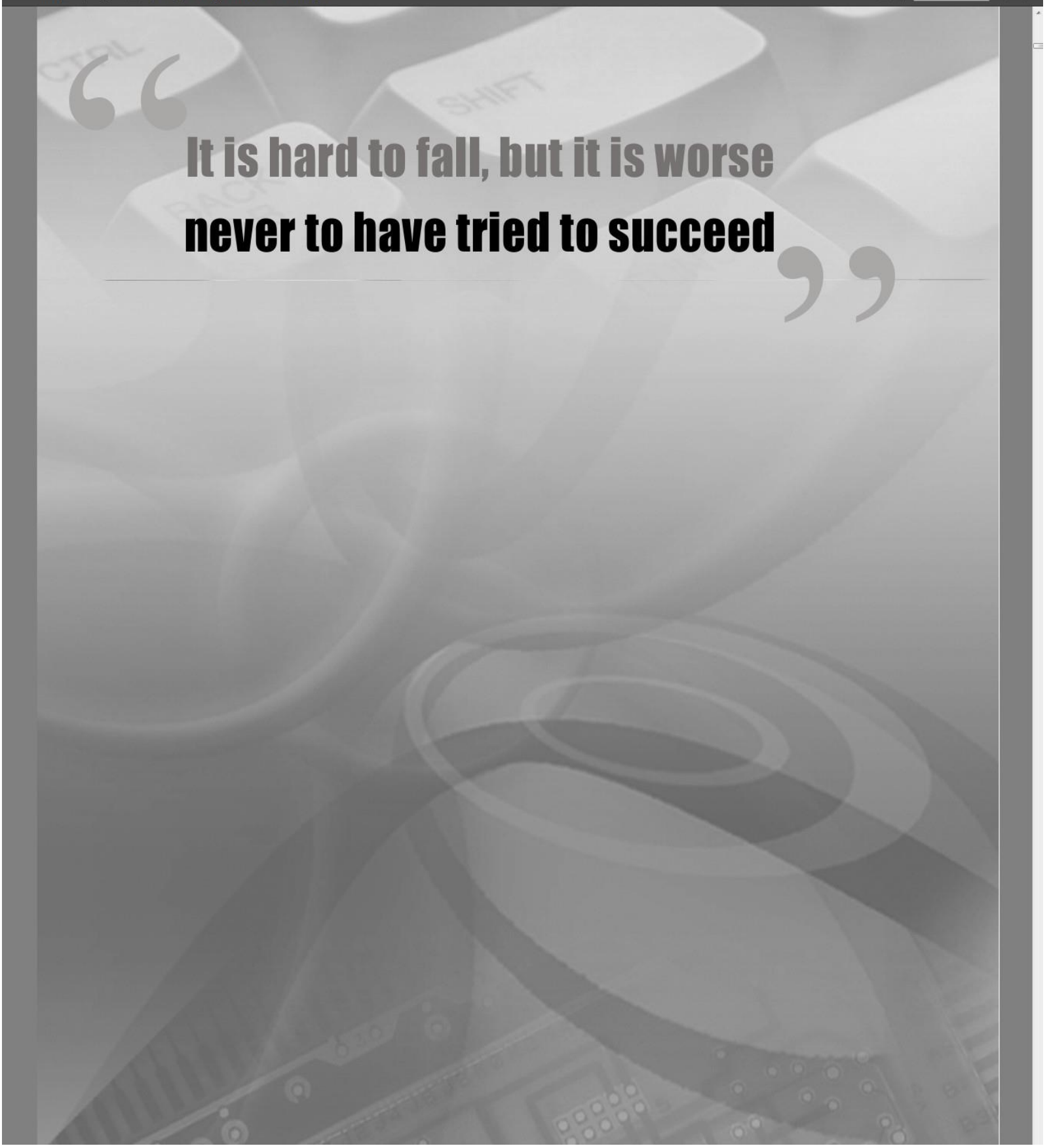
Design Team


Nothing is a waste of time if you
use the experience wisely

Aptech COMPUTER EDUCATION Unleash your potential

Table of Contents

Sessions


It is hard to fall, but it is worse
never to have tried to succeed

Session - 1

Introduction to Computers

Welcome to the Session, **Introduction to Computers**.

The session will discuss the different hardware and software parts of a computer. It will also discuss the startup process of a computer and the latest technologies in computer.

In this Session, you will learn to:

- Define a computer system
- Describe parts of a computer
- Explain the computer system startup process
- List new technologies in computer

1.1 Introduction

A computer can be defined as a system that responds to a set of instructions in a well-defined manner. In other words, it can be defined as a device that manipulates the data based on a program. A program is defined as a prerecorded list of instruction.

The computer system consists of different parts namely, Central Processing Unit (CPU), Motherboard, Random Access Memory (RAM), hard disk, main cabinet with all the peripheral devices connected to the motherboard, and an Operating System (OS). It is known as a system because all the components work together to achieve a common goal.

The computer system is classified into two types namely, workstations (clients) and servers. Workstations or client machines are either a high-end system used by a single person or a system that accesses a service made available by a server. The client machine can be a user's laptop or desktop computer. Servers are always present in a network and share its data, resources, and applications with multiple users. The servers range from simple desktop computers to mainframes.

The capacity of a computer system depends on the number of task, number of users sharing the system concurrently, the type of task performed, and the amount of data that requires to be stored.

1.2 Types of Computers

Computers can be classified based on its size and processing power. The different types of computers are as follows:

- Personal Computers
- WorkStation
- MiniComputer
- SuperComputer

1.2.1 Personal Computers or Microcomputers

Personal Computers (PC) are relatively small as compared to supercomputers and mainframes. They are inexpensive and designed for individual users. They are used in business, homes, and schools for surfing Internet, playing games, listening to music, and so on.

They come in different forms such as desktop computers, laptop computers, tablet PCs, Personal Digital Assistants (PDAs), and so on.

➔ Desktop Computers

These computers are not portable and are designed to fit on desk. They are much smaller in size, easy to use, and cheap as compared to other types of PCs. They are most commonly used in offices, homes, cyber café, and so on.

Figure 1.1 shows a desktop computer.

Figure 1.1: Desktop Computer

➔ Laptop Computers

Laptop computers are light-weight and portable. All the functionalities of desktop computers are integrated into laptop. The physical size of a laptop is the primary factor that determines the cost of a laptop. Older laptops were heavier as compared to modern laptops. The lighter and smaller a laptop is, the higher is its price. Laptops that are smaller in size are known as notebooks. Figure 1.2 shows a laptop.

Figure 1.2: Laptop

➔ Tablet PCs

Tablet PCs are similar to notebook PC and have a touch screen or pen enabled interface. It has an onscreen virtual keyboard. They are very useful and popular for writing notes in the fields of law, education, and medicine. This type of computer offers mobility for a user who may not have enough space to work with a desktop, laptop, or notebook PC. Figure 1.3 shows a Tablet PC.

Figure 1.3: Tablet PC

➔ Handheld Computers or PDA

A PDA also known as a palmtop computer or a pocket PC, is a handheld miniature computer. They are smaller and lighter in weight and can be carried in a pocket. They are not powerful like desktops or laptops. These devices hold data such as schedules, notes, appointments, address book, and many more. Modern PDAs support applications such as word processors and image viewers. Figure 1.4 shows a PDA.

Figure 1.4: PDA

Mobile phones that are integrated with PDA functionality are called Smartphones. Figure 1.5 shows Smartphone devices.

Figure 1.5: Smartphones

1.2.2 Workstation

A workstation can be defined as a type of computer that is mainly used for technical and scientific applications, desktop publishing, and so on. They are used by a single person and are connected to a local area network (LAN). It has a moderate computing power and a high quality graphic capability.

Workstations have built-in network support, mass storage device, graphical user interface, and so on. There are some workstations which are without disk drive and are known as diskless workstation. Like personal computers they are single user and can be used as a stand-alone system. In networking, workstation refers to any computers that are linked to a LAN.

1.2.3 MiniComputer

Minicomputer is a multi-user computer and supports hundreds of users concurrently. They are smaller to mainframe computers as far as speed, performance, and storage capacity is concerned. When compared with mainframes they are less expensive. The category of minicomputers falls between the category of workstations and mainframes.

1.2.4 SuperComputer

SuperComputers are fast computers that can execute trillions of instructions per second. They are the best and most expensive computers. They are used by applications that require intensive numerical computations such as weather forecasting, nuclear energy research, oil and gas exploration, and so on.

Mainframe is a large computer that can support hundreds or thousands of users simultaneously. The difference between a supercomputer and mainframe is that supercomputer executes few programmes as fast as possible, whereas mainframe executes many programs simultaneously. Mainframes are used in banking, airline, and railways.

1.3 Various Components of Computer System

There are different components in a computer, some of them are additional components which are not required very often.

The different mandatory components of a computer system are as follows:

- **Processor** - The main component of a computer that executes all processes and instructions supplied by memory unit. It is the brain of the computer and stores information about the OS. The processing speed of the processor is measured in megahertz. The processor is also called as CPU. Figure 1.6 shows a processor.

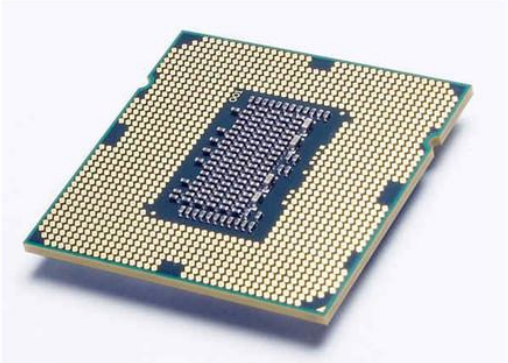


Figure 1.6: Processor

- **Random Access Memory (RAM)** - This area in the computer stores all the instructions (processes) and information of system. Since it is a volatile memory, the data stored in the RAM is not permanent; it is erased when the computer is turned off. Memory is measured in bits and bytes. It is responsible for overall system performance and speed. Figure 1.7 shows a RAM.

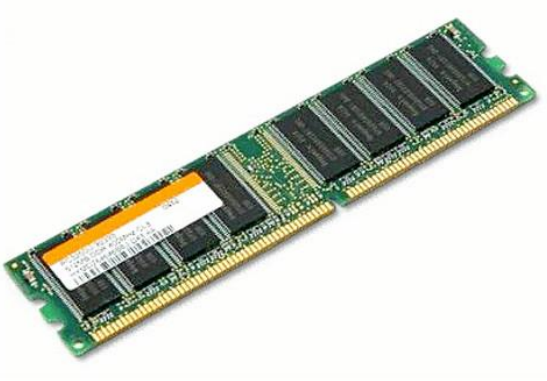


Figure 1.7: RAM

➔ **Read Only Memory (ROM)** - ROM is a memory chip, which contains in-built programs written into it by manufacturers. ROM, being a non-volatile memory, retains the data stored in it. The data stored in it can be read but cannot be altered. Figure 1.8 shows a ROM.

Figure 1.8: ROM

➔ **Hard Disk Drive (HDD)** - A HDD, also known as a hard drive is a non-volatile storage device that stores data or programs. It is a non-volatile memory because it retains information, unless the information itself is deleted or overwritten. It can store large amount of data and is used to transfer data back and forth between the disk and the memory. It is measured in gigabytes. Figure 1.9 shows a HDD.

Figure 1.9: Hard Disk Drive

➔ **Monitor** - The most commonly used output device is the monitor. It is used to display a variety of information. Monitor comprises a display device, a circuitry, and an enclosure. The important characteristics of a monitor are size and clarity.

The different types of monitors are as follows:

- **Cathode Ray Tube (CRT) monitor** - CRT monitors are square-shaped, heavy boxes, typically placed on a desktop. Figure 1.10 shows a CRT monitor.

Figure 1.10: CRT Monitor

- **Liquid Crystal Display (LCD) monitor** - LCD monitors are flat portable monitors. These monitors are thinner and lighter when compared to CRT monitors. LCD monitors are capable of displaying images with more color and better clarity. They require less power to run. Figure 1.11 shows a LCD monitor.

Figure 1.11: LCD Monitor

- **Plasma monitor** - Plasma monitors are less commonly used with computers due to their high operating temperatures, high power consumption, and fixed resolution. They are mostly used with TVs. Figure 1.12 shows a plasma monitor.

Figure 1.12: Plasma Monitor

- **Touchscreen monitor** - Touchscreen monitors provide a new way of interacting with your computer with a touch-sensitive screen. This allows users to interact directly with the application on screen and does not require a mouse or a keyboard. It detects the presence and location of touch within the display area. It can also detect other objects such as stylus. Figure 1.13 shows a touchscreen monitor.

Figure 1.13: Touchscreen Monitor

- **Organic Light Emitting Display (OLED) monitor -** OLED computer monitors are much thinner and brighter than LCD or Plasma screens. OLED monitors can also be placed on transparent surfaces, such as glass, allowing the user to see through them when not active. Figure 1.14 shows an OLED monitor.

Figure 1.14: OLED Monitor

➔ **Keyboard** - The keyboard is one of the most common input devices. Keyboard convert numbers, letters, and other special characters into digital signals, which the system understands. These digital signals are then processed by the system. Figure 1.15 shows a keyboard.

Figure 1.15: Keyboard

➔ **Mouse** - A mouse is an input device that controls a pointer, which is displayed on the monitor. It generally has two buttons, on the left and right sides. It is also available with a scroll wheel in the middle that is used to scroll through the window. The mouse is used to click and drag objects on the graphical interface as well as to select or activate options.

Figure 1.16 shows a mouse.

Figure 1.16: Mouse

➔ **Software** - Software is a generic term for organized collections of computer data and instructions. Any software during installation is stored by default in the hard disks in program files. There are two types of software namely, application and system software. Application software performs a specific task such as word processing. System software represents the OS and the utility programs. It is designed to operate and control the hardware as well as act as an interface for the application software.

1.4 Computer Software and its Types

A computer is defined as an electronic device, and similar to other electronic devices it also needs to be instructed to perform a desired job. Hence, a sequence of instructions is specified to a computer to solve a problem. These sequence of instructions are written in a language, which can be comprehended by a computer and they are called a computer program. The program controls the processing of the computer, and it performs specifically as the program instructs. A set of computer programs, procedures, and associated documents (flowcharts, manuals, and so on) describes the program. They also show how the program can be executed. A software package is a collection of programs, whose objective is to accomplish a particular task. For example, a word-processing package can have programs for text editing, text formatting, drawing graphics, spell check, and so on. Hence, a computer system has multiple software packages for each job.

There are two types of software namely, system software and application software. They are discussed in detail.

1.4.1 System Software

System software comprises several highly complex programs that work together in order to make all the components of the PC work together as a single unit. For example, it enables input devices to accept user input, output devices to display the output, and other devices to be involved in processing the data in different ways. Without the system software, the computer cannot operate as a single unit.

System software includes the following:

- Device drivers
- Operating systems
- Servers
- Utilities

1.4.2 Application Software

Application software is the software that enables the end-user to perform specific tasks, such as word processing, image manipulation, and so on. Application software falls into the following categories:

- Business software
- Computer games
- Medical software
- Military software
- Photo-editing software
- Spreadsheets
- Word processors
- Decision making software

1.5 Computer System Boot Process

In order to boot a computer successfully, its Basic Input Output System (BIOS), operating system and hardware components must work properly. Failure of any one of these three elements will likely result in a failed boot sequence.

As soon as the computer power is turned on, the processor initializes itself and gets the first instructions from the BIOS ROM. BIOS is a microchip that stores instructions such as Power On Self Test (POST) in a predetermined memory address. POST begins by checking the BIOS chip, testing the CMOS RAM, then initializing the processor, checking the inventoried hardware devices (such as the video card), secondary storage devices such as the hard drives and floppy drives, the ports and other hardware devices such as the keyboard and mouse, to ensure they are functioning properly.

Once the POST determines that all components are functioning properly and the processor has successfully been initialized, then the BIOS look for an OS and its boot sequence in the Master Boot Record (MBR). The MBR shows the BIOS where to find the OS as well as the subsequent program file that will initialize the OS.

Once the OS is initialized, the files are copied into memory by the BIOS and the OS takes over the control of the boot process. After the OS is started, it checks the availability of the system memory, and then loads the device drivers of the peripheral devices that it needs to control, such as printer, scanner, optical drive, mouse, and keyboard. Figure 1.17 shows boot device order.

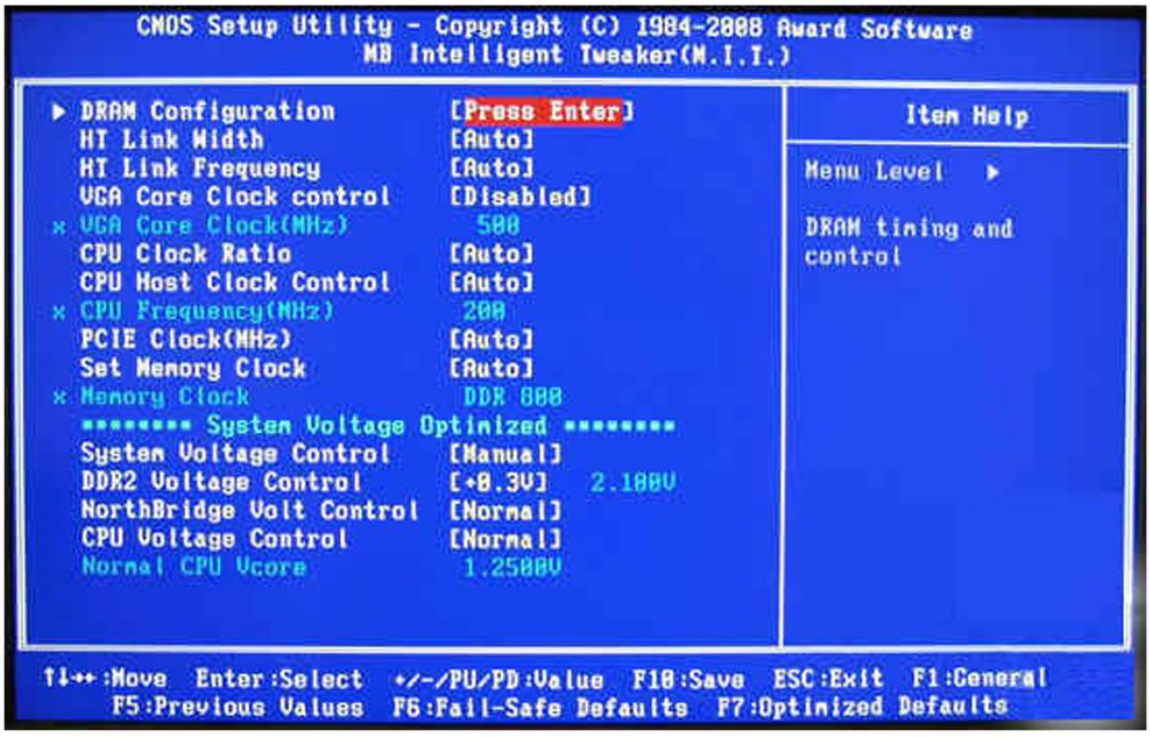


Figure 1.17: CMOS Boot Device Order

1.6 New Technologies in Computer

The important characteristics of this technology era are transformation, transmission, and dominance of information. The information society offer great preference to new technologies, specifically to those dedicated to information.

Modern technology has revolutionized the communication methods. Other than radio, telephone, satellite communication, cellular technology and wireless Internet, in present day, two people can communicate with a help of computer when they are in remote places. Technology is all about building connections between people on opposite sides of the globe, thus bringing people from different cultures and backgrounds together.

Internet has a large information base. New technologies have made it possible for this information to reach far and wide. Online education has helped in rendering knowledge to students living in their home or remote location. Introduction of new technologies helped machines to execute difficult and recurring tasks, hence the laborious work of the people has reduced. Automation of processes has increased efficiency and speed. Rapid performance of tasks has reduced human effort and time.

1.6.1 Cloud Computing

Cloud computing evolved from a concept called virtualization. Virtualization is the process of creating a virtual version of an OS, a server, or network resources. Using virtualization, you can host multiple Operating Systems at the same time on a single machine.

A traditional application server may have just 5-10% utilization, whereas virtualized servers can reach 50-80% utilization. By hosting more virtualized instances on fewer physical servers, you can lower costs for hardware acquisition, maintenance, energy and cooling system usage.

Cloud computing supports more features compared to virtualization. Cloud computing supports Green IT by consuming low energy. If few computers perform the same amount of work, then the utilization increases and the cost of assembling those computers are saved. Cloud computing saves a lot of energy as a number of systems are consolidated into a single machine, thus saving power. This is especially useful for data centers.

Cloud computing is an approach enabling convenient and on-demand access through the Internet to resources such as networks, servers, storage, applications, and services.

Figure 1.18 shows cloud computing infrastructure.

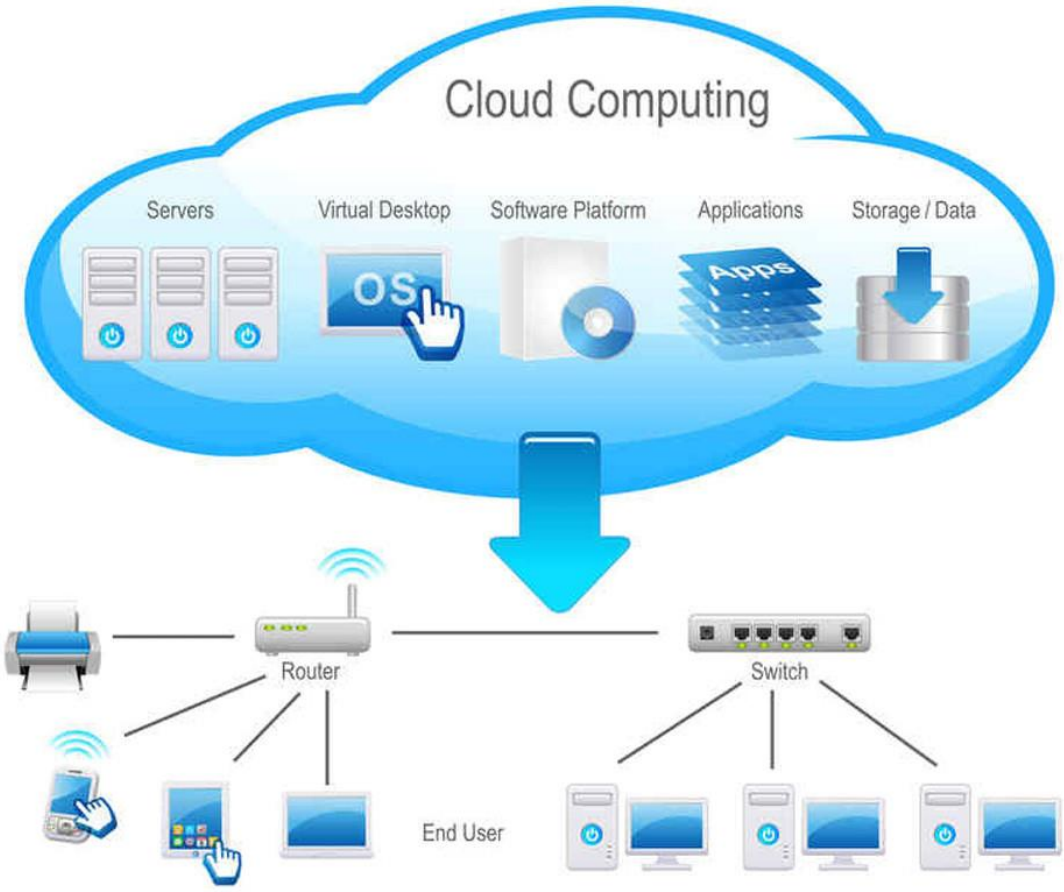


Figure 1.18: Cloud Computing Infrastructure

Several types of clouds have emerged during the premature stages of a major paradigm shift from single organization local data center to big organization remote data centers.

Cloud computing is divided into three major cloud structures and they are as follows:

➔ **Public Cloud** - A public cloud is a cloud that can be accessed by anyone using the Internet. A vendor constructs, configures, manages, and monitors the backbone of the public cloud in their data centers. The client manages their normal resources that operate within the cloud (data, apps, and so on), and provide appropriate remuneration for the services used. In public cloud there is no long term commitment with any particular vendor cloud, the client can discontinue the services at any time.

Most of the clients are interested in using their private data centers due to the lack of security in sharing the data to public cloud utilities.

However, the attraction of the instant IT resource access provided by the cloud technology has increased the usage of private clouds.

- **Private Cloud** - A private cloud is cloud technology which uses a private data center to which only one organization has access. In this way, the organization still maintains its own data center and staff, but IT resources within the cloud are available on-demand according to the requirement of the organization's employees.

  Private clouds can work within a private segment of a public cloud. By this, an organization obtains the outsourcing benefits of a public cloud with a level of security and privacy similar to a private setting.

- **Hybrid Cloud** - Hybrid clouds are a combination of private and public clouds. When an application in a private cloud experiences abnormally high demand, the application extends to use resources present in a public cloud (or private segment of a public cloud) and this type of scaling is called as cloud bursting.

The different types of services provided in cloud computing platform are as follows:

- **Infrastructure as a service (IaaS)** - In this model, computers and other resources are provided to the users.
- **Platform as a service (PaaS)** - In this model, OS, program execution environment, database, and Web server are provided as a service.
- **Software as a service (SaaS)** - In this model, cloud providers install and operate the application software in the cloud and the users access it.
- **Storage as a service (STaaS)** - In this model, large service providers rents out storage infrastructures.
- **Security as a service (SECaaS)** - In this model, large service providers integrates into the corporate infrastructure's security services.
- **Data as a service (DaaS)** - In this model, data is provided to the user as and when required irrespective of geographic location or organization separation between the provider and the consumer.
- **Test Environment as a service (TEaaS)** - In this model, software and its data are hosted to be accessed by users using a Web browser over the Internet.
- **Desktop as a service (DaaS)** - In this model, desktop is virtualized.
- **API as a service (APIaaS)** - It enables the creation and hosting of APIs.

1.6.2 Grid Computing

Grid computing can be defined as an interconnected computer system where the machines utilize the same resources collectively for solving a problem or reaching a common goal. Grid computing usually consists of one main computer that distributes information and tasks to a group of network computers to accomplish a common goal.

Advantages of grid computing are as follows:

- Large six figure Symmetric Multiprocessing (SMP) servers for application processing are not required. Results can then be concatenated and evaluated upon job completion.
- Idle resources can be utilized much more efficiently by distributing jobs to idle servers or idle desktops. Policies instruct jobs to be sent to lightly loaded server or to servers that has appropriate CPU characteristics for specific applications.
- A grid environment has modular structure and does not have single points of failure. Even on failure there are backup servers or desktops that can complete the job efficiently. Jobs restart automatically when there is a failure.
- Systems can be upgraded without scheduling downtime.
- Jobs are executed in parallel speeding performance. Grid environments are most suitable for jobs that can be divided into smaller chunks so that they can be executed in parallel on many nodes.

Disadvantages of grid computing are as follows:

- Large SMP are still used when applications that require high memory do not take advantage of Message Passing Interface (MPI).
- A fast connection in between the computer resources (gigabit Ethernet at least) is required.
- Some applications are required to be fine-tuned to take full advantage of the new model.

1.6.3 Utility Computing

In utility computing, the computing resources and infrastructure is provided to the customer by the service provider. The customer is charged according to the service usage. This type of computing is a part of service provisioning model.

Advantages of utility computing are as follows:

1. The client is not required to buy all the hardware, software and licenses, instead, the client depends on the utility computing company to provide these services.

2. In a large company, the files used by employees in one department of a company might be incompatible with the software used by employees in another department. Utility computing gives companies the option to subscribe to a single service and use the same software suite for the entire client organization.

Disadvantages of utility computing are as follows:

1. If a utility computing company has financial difficulty or has frequent equipment problems, clients could get discontinued from the services even after paying.
2. Utility computing systems are always targets for hackers. A hacker can access services without paying and sneaking around and exploring client files.

1.7 Check Your Progress

1. Desktop computers are not portable and are designed to fit on _____.

(A)	docking system	(C)	network
(B)	desk	(D)	None of these

2. Match the following:

Parts of a Computer		Description	
(a)	RAM	1.	Is a non-volatile memory.
(b)	Monitor	2.	Executes all processes and instructions provided by memory unit.
(c)	Processor	3.	Is used to display a variety of information.
(d)	ROM	4.	Is a volatile memory.

(A)	a-1, b-2, c-3, d-4	(C)	a-4, b-3, c-2, d-1
(B)	a-2, b-3, c-4, d-1	(D)	a-4, b-2, c-3, d-1

3. Which of the following component interact directly with the application on screen without the need for a mouse or keyboard?

(A)	CRT monitor	(C)	Touch screen monitor
(B)	Plasma monitor	(D)	OLED monitors

4. A traditional application server may have just ____ utilization, whereas virtualized servers can reach _____ utilization.

(A)	60-80%	(C)	6-10%
(B)	5-10%	(D)	50-80%

5. Which of the following environment has a modular structure and does not have single points of failure?

(A)	Grid Computing	(C)	Utility Computing
(B)	Cloud Computing	(D)	None of these

1.7.1 Answers

1.	B
2.	C
3.	C
4.	B, D
5.	A

- The computer consists of different parts namely, Central Processing Unit (CPU), Motherboard, Random Access Memory (RAM), hard disk, main cabinet with all the peripheral devices connected to the motherboard, and an operating system (OS).
- A workstation can be defined as a type of computer that is mainly used for technical and scientific applications, desktop publishing, and so on.
- Booting is the process of loading the operating system when the user switches ON the computer system.
- Cloud computing is an approach enabling convenient and on-demand access through the Internet to resources such as networks, servers, storage, applications, and services.
- Grid computing usually consists of one main computer that distributes information and tasks to a group of network computers to accomplish a common goal.
- Utility computing is a service-provisioning model. In this model, the service provider makes computing resources and infrastructure management available to the customer as needed, and charges them for specific usage rather than a specific rate.

Session - 2

Introduction to Programming Tools

Welcome to the Session, **Introduction to Programming Tools**.

This session explains the evolution of computer languages and the aspects of programming. It also discusses data storage in computers. Finally, the session lists the different modeling tools available for program designing.

In this Session, you will learn to:

- Explain data storage in computers
- Describe the evolution of computer languages
- Identify aspects of programming
- List the different modeling tools for program designing

2.1 Introduction

Computers can perform different functions and calculations, but a computer cannot take a decision as a human being. Computers have to be instructed or programmed to perform certain task according to the requirement. For this purpose, several programming languages have been developed to help programmers write code to meet user's requirement. Though there are various programming languages that are developed, the basic approach to write the programming code is almost similar for all programming tools. For example, consider the code written using C, BASIC, and Java as shown in figure 2.1.

C

```
#include <stdio.h>

void main()
{
char name[15];
printf("Enter Name: ");
scanf("%s", &name);
printf("Hello %s ", name);
}
```

BASIC

```
10 Input name$
20 Print "Hello "; name$
30 END
```

JAVA

```
Class Hello
{
public static void main(String args[])
  {
   String name;
   System.in.readln(name);
   System.out.println("Hello  " + name);
   }
}
```

Figure 2.1: Code Written Using C, BASIC, and Java

All the three snippets of code achieve the same task that of accepting the user's name and displaying a welcome message. However, the syntax differs from one language to the other, the logic or the steps followed for each program is the same.

2.2 Computer Languages

For the purpose of communication between the different components and programs of a computer, computer languages were developed. These languages provided instructions that helped computer and the programs to communicate with each other.

Some examples of computer languages are C, C++, Java, and so on. Languages are broadly categorized into three types as shown in figure 2.2.

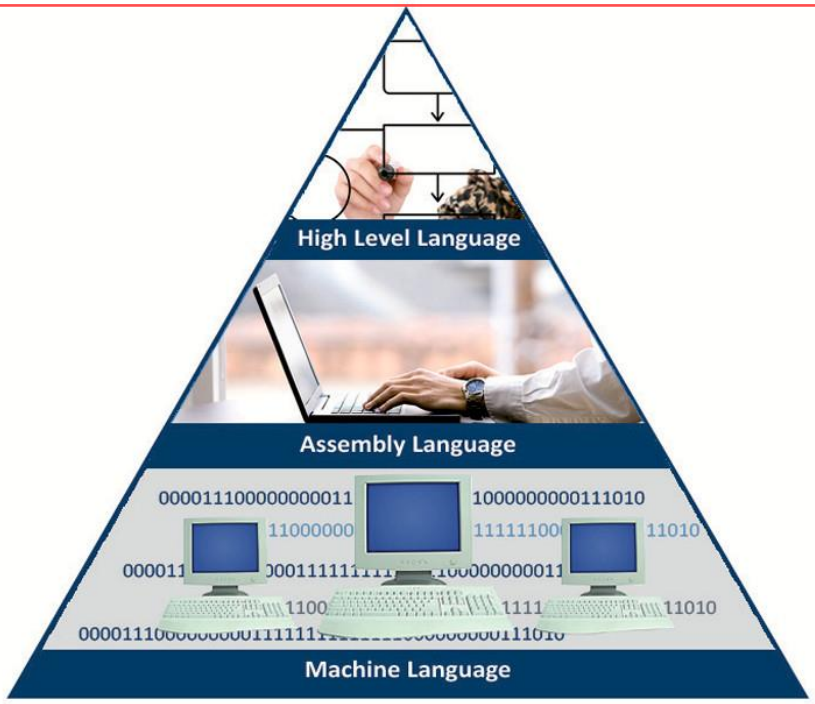


Figure 2.2: Computer Languages

➔ **Machine Language** – This is one of the earliest invented languages. Machine language is basic in nature and is cumbersome and difficult to understand. As the name suggest it was developed for communication between the components of the machine. Machine language was machine specific and was not compatible with different types of machines. For example, machine language for IBM computers was different from Apple computers. Machine language used strings of 0s and 1s to represent instructions and hence, it is also known as binary language. This language was used for writing instructions which could easily be understood by machines and need not be translated.

The main advantage of this language was that the code could be executed very fast and efficiently as the CPU could directly execute it. The drawback of this language was that it was difficult to learn and edit when an error takes place. Additionally, if any instruction was required to be added in the middle, then all the remaining instructions were required to be removed and place made in the memory to accommodate the new instructions. Programs written in machine language were not required to be translated for the computer to understand the instructions.

- **Assembly Language** – This language is second-generation language. It was introduced after the machine language, as machine language was very difficult to understand. To overcome the drawbacks of machine language, the Assembly language was developed. It used cryptic English like phrases for writing codes that represented strings of numbers.

  The language used many Mnemonics that were easy for the programmers to understand and had some specific meaning. Mnemonic was used to represent low level machine operation or opcode. Some opcode required operands as part of the instruction. For example, in the statement ADD A, B, ADD is the Mnemonic used for adding the operands A and B. Each assembly language was specific to the architecture of the computer. For execution of the code, the language was required to be translated into machine language using a translator known as assembler. In other words, the assembler creates object code by translating the mnemonics into opcodes and by resolving the symbolic names for memory locations. The process of converting the assembly language to machine language is known as assembly or assembling the code.

  This language became popular with programmers who were trying to create a communication interface between the computer and programs because of English like words. The disadvantage of this language was that it was required to be converted into machine language as it used certain human readable words. Like machine language, assembly language is also machine specific.

- **High Level Language** – Though assembly language used English words, they were difficult to learn. To address this issue and to make it easier to write programs, high level languages were introduced. Instructions could be provided in an easier way using these high level languages. FORTRAN was one of the first languages introduced by IBM in 1957 for solving scientific and engineering problems. Some other high level languages that were introduced later were COBOL, PASCAL, C, and so on.

  High level language is more powerful when compared with machine and assembly language. It allows the user to work in a more English like environment. High level programming languages are further categorized into three groups namely, third generation, fourth generation, and fifth generation languages. Each of these generations are more powerful.

- **Third Generation Languages** – Is also known as 3GLS. They use English like phrases and are much easier to use. They are also portable. It means that the object code can be translated to be used in different systems. Example of 3GLs are Fortran, Cobol, C, Basic, Pascal, and so on.

- **Fourth Generation Languages** - Is also known as 4GLS. They are easier to use and has text based environment or a visual environment. In visual environment, the programmers work with graphic tools. Example of 4GLs are Visual Basic, Visual Age, and so on.

- **Fifth Generation Languages** - Exists as a debatable issue in programming community. They use artificial intelligence to create software and are very difficult to develop.

2.3 Data Storage

The data for processing is stored in the memory or primary storage. Memory is classified into two types, namely, primary storage and secondary storage.

2.3.1 Primary Storage

Primary storage is a temporary storage. RAM is known as the primary storage or the main memory. It is a volatile memory as the data stored in it is lost when the power is switched off. Secondary storage consists of other memory devices such as hard disk drives, optical disc drives, and so on. Secondary storage devices are non-volatile. Non-volatile means the data stored in the memory is not erased even after the power is switched off.

Memory is measured in terms of bytes. A byte consists of 8 bits (Binary Digits). A byte can normally store one character. The characters can be any one of the following:

- Any number from 0 to 9
- Capital letters from A to Z
- Small letters from a to z
- Special characters such as . , + - % ' " ! and so on

The memory capacity of a microcomputer is 640 kilobytes, which means that 655360 bytes (640 * 1024) of data can be stored.

Assume a byte is a small box, which contains eight switches kept next to each other as shown in figure 2.3.

Figure 2.3: Byte

Each switch has two positions, ON or OFF. When the switch is ON the values are as follows:

- 1st switch - 1 i.e. 2^0
- 2nd switch - 2 i.e. 2^1
- 3rd switch - 4 i.e. 2^2

- 4th switch - 8 i.e. 2^3
- 5th switch - 16 i.e. 2^4
- 6th switch - 32 i.e. 2^5
- 7th switch - 64 i.e. 2^6
- 8th switch - 128 i.e. 2^7

If 1 is required to be stored, the first switch will be ON and the remaining switches will be OFF. If 9 is required to be stored, then the first and the fourth switch will be ON and the remaining switches will be OFF. When all switches are ON adding all the values will result in 255. Thus, a byte can store any numbers ranging from 0 to 255 as each of the characters are represented by a number.

There are different types of codes used for storing characters in memory such as American Standard Code for Information Interchange (ASCII), Binary Coded Decimal (BCD), and Extended Binary Coded Decimal Interchange Code (EBCDIC).

In most computers, ASCII system is used for storing data in a byte. It has associated a number with each character that is each alphabet has a number associated with it. For example, '0' is represented by the number 48, A is represented by 65.

Data is stored in the memory. CPU constantly reads data from the memory and executes them as and when required. Memory is divided into four parts as shown in figure 2.4.

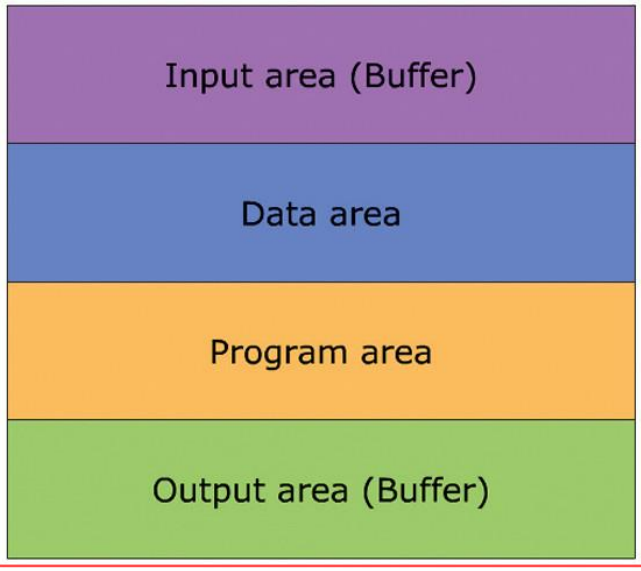


Figure 2.4: Parts of RAM

Data read from the input devices are stored in the input buffer and data to be displayed or output is stored in the output buffer. Variables that are required for a short time is stored in the data area and the steps of a program are stored in the program area.

2.3.2 Secondary Storage

Secondary storage is different from primary storage. It is a non-volatile memory and is not directly accessible by the CPU. The computer swaps data in and out of the memory to the secondary storage using the Input/Output channels. Secondary storage devices used in a computer are as follows:

➔ **Hard disk drives**

Hard disk drives are devices that store digital data of a computer for easy retrieval later. The hard disk drive is the largest storage device in a computer. It can store a large amount of data. A hard disk contains rotating discs called platters that are used to store data. This data is read using a magnetic head. Most files of a computer are stored in a hard disk drive. Figure 2.5 shows a hard disk with magnetic head and platters.

Figure 2.5: Hard Disk

➔ **Optical disk drives**

Optical disk drives use laser lights or electromagnetic waves to read and write data from optical discs. These optical discs include Compact Disc (CD), Digital Versatile Disc (DVD), and Blu-ray Disc (BD). These discs store data but not as much as hard disk drive. Optical discs are used because of its easy portability option.

➔ **Flash memory**

Flash memory is a memory chip that is designed to store data that can be easily erased and new data can be added. This type of chip is commonly used in memory cards and USB pen drives.

➔ **Other storage**

The other storage devices which are not commonly used these days include floppy disks, magnetic tapes, punch cards, zip drives, and so on.

2.4 Working of Computer Systems

Data is defined as a collection of numbers, texts, or symbols that is used as input to a computer, and then processed as required. In other words, data is represented as raw, unorganized facts that need to be processed. Data has no meaning of its own. When this data is processed into organized collection of useful and meaningful data it is known as information.

Computers are used to process the raw data and provide meaningful information to the user after processing the data. Thus, computers can be referred as an information processing system. They are not intelligent as human beings and can process the data according to the instructions that have been fed into the computers. These instructions, which are provided, to the computers are known as programs.

2.4.1 Input-Process-Output

As discussed, a computer is an electronic device that accepts the raw data, processes the data according to the instruction provided, and generates the output. Thus, it can be said that a computer can only complete its task by going through the following three stages. These stages are input, process, and output as shown in figure 2.6.

Figure 2.6: Stages

Input – In this stage, the computer accepts or receives the data required by the program to produce the required output. Input devices are used to receive the data and keyboard is the standard input device.

Processing – The data received from the input devices is processed in the CPU. The action performed on the data depends on the instruction provided by the user of the data. The action can be arithmetic or logical calculation, editing, and so on. In this stage, the output produced will depend on the input provided.

Output – Output is the result generated after processing of the data. The processed data is displayed to the user on output devices. Monitor or screen or Visual Display Unit (VDU) is the standard output device. The output may be in the form of text, sound, image, document, and so on.

2.4.2 Hardware Components

A general purpose computer system consists of the following hardware components:

- Input/Output Device
- Control Unit
- Memory

These units are connected by wires known as buses.

- **Input/Output (I/O) Device**

  I/O devices are used to exchange information. These devices are known as peripheral devices and are connected to the computer by wires.

- **Control Unit**

  The CPU consists of three sub-units, namely, Control Unit (CU), Arithmetic Logic Unit (ALU), and Memory. The CU controls and coordinates by sending signals and commands to various parts of the computer system to prepare and process the data. The ALU processes the data using arithmetic and logical operation. The processed data is then sent to main memory, which stores it temporarily before the next command is executed. The data is then sent to the secondary storage devices after secondary storage device had received command from the CU.

- **Memory**

  Memory can be viewed as cells into which data are placed. As discussed earlier, each cell is numbered and can store a single number. In modern computers each cell store binary numbers.

2.4.3 Input Devices

A hardware device that is used to receive data from the computer user is known as an input device. Different types of data such as audio, video, text, and so on can be provided by the user. The commonly used input devices are as follows:

- **Keyboard** – A keyboard is an input device mainly used for typing text. A keyboard is like typewriters with the keys being used are in QWERTY format. A keyboard can be used to insert alphabet, numbers, and symbols in a document or invoke a particular function of the computer. The keyboard consists of the following keys:
  - Alphabet Keys (A, B, C, and so on)
  - Numeric Keys (1, 2, *, %, #, and so on)

- Function Keys (F1, F2, F3, and so on)
- Special Keys (Ctrl, Alt, Home, Page Up, and so on)

➔ **Mouse** – A mouse is a pointing device that can be used to perform various functions such as opening and closing a program, selecting the text, and also opening additional functions using the right-click button. The movement of a mouse is detected by moving the mouse on a flat surface. Movement of the mouse is echoed on the screen by the movement of a pointer. There are different types of mouse that are available such as wireless mouse, optical mouse, wheel mouse, and so on. A mouse is generally connected on the PS/2 port or the USB port.

➔ **Microphone** – A microphone is an audio input device. Microphone is also called as Mic or mike. It converts sound into an electrical signal. Generally, a mic is attached with headphones which are output devices.

➔ **Scanner** – A scanner is an input device that scans images, printed text or an object and converts it into digital image. A scanner is usually connected to the USB or parallel port.

➔ **Webcam** – A Webcam is a camera attached to a computer. It is used to capture still pictures and videos and feed its images in real-time to computer using a Wi-Fi or an USB port. It is mostly used for establishing video links and thus, enables the computer to behave as a video conference station.

Other input devices include barcode readers, digital cameras, fingerprint scanners, trackballs, touchpads, joysticks, and so on.

2.4.4 Output Devices

A hardware device that is used to display the processed data to the computer user is known as an output device. The output can be in both tangible and non-tangible format. The commonly used output devices are as follows:

➔ **Monitor** – A monitor is a visual display unit that can be used to view the graphical output. This output can be text, images, videos, and so on. There are different types of monitors available such as CRT monitors, flat screen LED and LCD monitors, touch screen monitors, and so on.

➔ **Printer** – A printer helps to generate a hard copy of the output on paper. Printers can be used to print both text and images. The different types of printers include dot matrix printer, inkjet printer, and laser printer. Printers are generally connected on the parallel port or USB.

Dot matrix printers are rarely used nowadays and form an image by placing a series of dots. The quality of the image depends upon the number of dots per inch. The print outs are of low quality when compared with inkjet or laser printers. Inkjet printers are the most commonly used printer and are mostly used at homes. It prints by spraying streams of quick-drying ink on paper. Inks are stored in disposable cartridges and these cartridges increases the overall cost of the printer.

Laser printer uses laser technology to print images on paper. They are normally used in places where printing has to be done in large quantity.

- **Plotter** - A plotter is a device similar to a printer, which is used for drawing large scale images. Plotters are used to print posters, CAD drawings, and so on. They use a pen, pencil, or a marker to draw an image. Plotters can be flat-bed or drum plotters. In flat-bed plotter a moving arm moves the pen over the paper rather than the paper moving under the arm. On the other hand, in a drum plotter a drum revolver is used to move the paper while the pen does the printing.

- **Speakers** – Speakers provide output in the form of audio to a computer user.

- **Projectors** – A projector is an output device that projects a large version of the screen output on a flat surface. Projectors are generally used for meetings, conferences, and presentations.

2.5 Aspects of Programming

To write a program, two important aspects need to be considered at the start. These two aspects of programming are shown in figure 2.7.

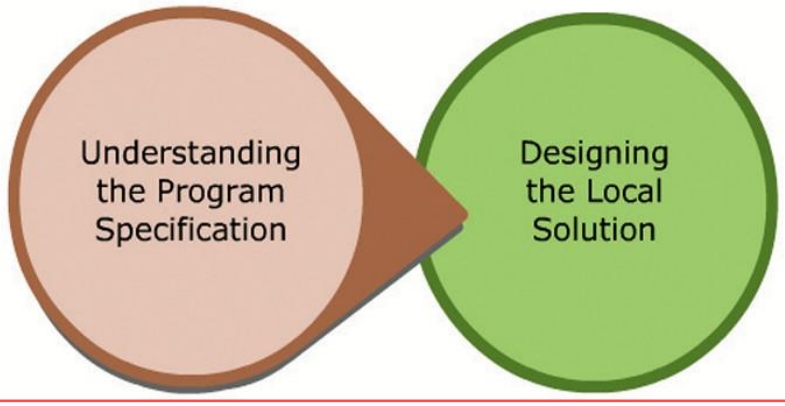


Figure 2.7: Aspects of Programming

1. Understanding the program specifications
2. Designing the logical solution

Program specifications list the available inputs, required outputs, and the other rules to be followed to get the required output from the given inputs. After the program specifications are received, the best logical solution needs to be designed. For designing the logical solution, different modeling tools are utilized. These modeling tools use algorithms to design the logical solution. An algorithm is a systematic procedure or instructions to solve a given problem or get the desired result.

2.5.1 Modeling Tools

To design the logical solution, two modeling tools are used. The tools used for this purpose are as follows:

1. **Flowchart** – A flowchart is a diagrammatic representation of an algorithm.
2. **Pseudocodes** – A pseudocode is a representation of an algorithm in a form that can easily be translated into programming statements.

Both these methods are used to specify a set of steps that need to be performed in order to arrive at the solution. A pseudocode and a flowchart represent the steps that need to be followed to achieve the solution. The actual implementation of the steps is done by the programmers when they write the code, using some language, to perform the specified steps.

Note - Modeling tools will be covered in detail in the later sessions.

2.5.2 Differentiate Between Flowcharts and Pseudocodes

Flowcharts and pseudocodes are two different ways of showing a program design in an easier manner. Both have their own advantages and disadvantages. The differences between flowcharts and pseudocodes using different parameters for differentiation are as follows:

- **Type** – Flowcharts use only diagrams for displaying the program design, whereas pseudocodes are text based.
- **Representation** – Flowcharts are pictorial representation of the logic, whereas pseudocodes are sentence-like representation of code.
- **Focus** – Flowcharts focus on the complete module of the software, whereas pseudocodes are used to mainly focus on the sub-sections of a module.
- **Layout** – Graphical layout is used by flowcharts and text-based layouts are used by pseudocodes.
- **Structure** – The structure of flowcharts are based on symbols and shapes, whereas pseudocodes use linear text-based structure.
- **Benefits** – Flowcharts are beneficial for smaller concepts and pseudocodes are beneficial for larger programming concepts.

2.6 Check Your Progress

1. How many characters can a byte store?

(A)	One	(C)	Four
(B)	Two	(D)	Ten

2. _____ is also known as primary storage.

(A)	ROM	(C)	Hard disk
(B)	RAM	(D)	Floppy

3. Which of the following section of the RAM stores variables for a short time?

(A)	Input area	(C)	Program area
(B)	Data area	(D)	Output area

4. _____ was one of the first languages introduced by IBM in 1957.

(A)	FORTRAN	(C)	PASCAL
(B)	COBOL	(D)	BASIC

5. Which of the following options are the two modeling tools?

(A)	Flowcharts	(C)	Pseudocodes
(B)	Programming	(D)	Assembly language

2.6.1 Answers

1.	A
2.	B
3.	B
4.	A
5.	A, C

- Languages are broadly categorized into three types namely, machine language, assembly language, and high level language.
- Machine language uses strings of 0s and 1s to represent instructions. Assembly language uses cryptic English like phrases for writing codes that represent strings of numbers. High level languages use English words.
- There are different types of codes for storing characters in memory such as American Standard Code for Information Interchange (ASCII), Binary Coded Decimal (BCD), and Extended Binary Coded Decimal Interchange Code (EBCDIC).
- The data for processing is stored in the memory or primary storage. Memory is measured in bytes.
- A computer is an electronic device that accepts the raw data, processes the data according to the instruction provided, and generates the output.
- The two aspects of programming are understanding the program specifications and designing the logical solution.
- The tools used to design the logical solution are flowcharts and pseudocodes. A flowchart is a diagrammatic representation of an algorithm. A pseudocode is a representation of an algorithm in a form that can easily be translated into programming statements.


Practice is the best
of all instructors

Session - 3
Flowcharts and Pseudocodes

Welcome to the Session, **Flowcharts and Pseudocodes**.

This session explains the modeling tool used by programmers to resolve problems. Finally, the session also explains the flowchart and pseudocode modeling tool used by a programmer.

In this Session, you will learn to:

- Explain algorithm
- Explain flowcharts
- Explain pseudocodes

Session 3

Flowcharts and Pseudocodes

Aptech COMPUTER EDUCATION Unleash your potential

3.1 Introduction

A programmer's job is to write instructions to achieve a goal or solution for a given problem. In other words, programmers write program to meet the user's requirement. A professional programmer does not sit down at a computer and start typing the code. A programmer follows a program development lifecycle to solve a problem. The program development lifecycle consists of the following steps namely, understand the problem, plan the logic, code the program, test the program, put the program into production, and maintain the program.

In programming, a programmer faces different problems while writing the code. To solve these problems, a correct understanding of the problem is required. After understanding the problem, the steps required to overcome the problem is required to be worked upon.

3.2 Overview of Programming Tools

The core of any programming process is planning the program's logic. In this stage, the programmer decides the steps to be included in the program and how to order them. Thus, at the fundamental level the programmer has to plan and write down the solution to a problem in a particular manner and this is known as algorithm. The programmer uses different tools for constructing the algorithm. The two tools used by the programmer are namely, flowchart and pseudocode. Both these tools help the programmer to write the steps of the program in English like language and allow the programmer not to concentrate on the syntax of any programming language. Planning the logic involves careful thinking of all the possible data that a program might accept and how to handle the data in different scenarios.

3.3 Algorithm

Consider a scenario where a student wants to go to the cafeteria from the classroom. The cafeteria is located in the basement. With these instructions, the first requirement is to work out the steps involved to perform the actual task of reaching the cafeteria. Hence, the steps required will be as follows:

Step 1: Leave the classroom

Step 2: Head towards the staircase

Step 3: Go down to the basement

Step 4: Head for the cafeteria

The procedure lists a concise and clear set of executable steps that need to be performed in order to solve the problem at hand. Such a set of steps is called an algorithm.

An algorithm can, therefore, be defined as a procedure, formula, or recipe for solving a problem. It consists of steps that help to arrive at a solution. It can also be defined as a step-by-step sequence of well-defined and unambiguous instructions to solve a problem in a finite number of steps.

In order to solve a problem, first you must understand the problem clearly. Next, gather any relevant information that is required. Then, the developer needs to process these bits of information. Finally, you need to arrive at the solution to the problem.

The algorithm thus represents a set of steps listed in simple language. It is very likely that though the steps written by two people may be similar, the language used to express these steps may be different. Therefore, it is necessary to have some standard method of writing algorithms so that it is easily understood by everyone. In programming, algorithms are written using two standard methods namely, flowcharts and pseudocodes.

Algorithm is an ordered set of instructions. The algorithm should have the ability to alter the order of execution of the instructions. Control structure is used to alter the order of execution of instructions. The three types of statement constructs that an algorithm can have are as follows:

- **Sequential** – where the instructions are executed one after the other.
- **Conditional** – where a condition is presented and depending on the result of evaluation of the condition, the next instruction is executed.
- **Iteration** – where a series of instructions are executed repeatedly until the condition controlling the execution of the instruction.

3.4 Flowcharts

A flowchart is a graphical representation of an algorithm. It charts the logical flow of instructions or activities in a process. Flowcharts help programmers to view how the statements in a program are interrelated. Each activity in a flowchart is depicted using symbols. It uses a combination of blocks and arrows to represent actions and sequence.

For example, the flowchart for displaying a message 'Hello' is shown in figure 3.1.

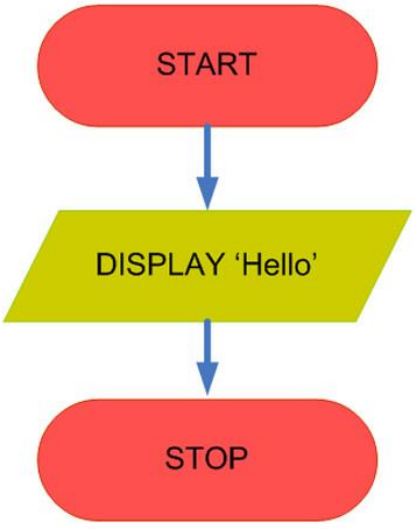


Figure 3.1: Flowchart

A flowchart begins with the START or BEGIN keyword, and ends with the END and STOP keyword. The DISPLAY keyword is used to display some value to the user. Figure 3.2 shows the different symbols used in flowcharts and their relevance.

Symbol	Description
	Marks the start or end of the program and are represented as rounded rectangles
	Is used for any process, function, or action and is represented as a rectangle
	Used for accepting or displaying (Input/Output) instructions and is represented as a parallelogram
	Used to represent decision making and branching statements and is represented as diamond
	Is used to represent the exit to and entry from another part of the same flowchart and is known as Connectors
	Used to show the flow of control and is known as Flow Line

Figure 3.2: Different Symbols in Flowchart

Consider a program accepting two numbers from the user and displaying the sum of the two numbers using a third variable. The flowchart for this example is shown in figure 3.3.

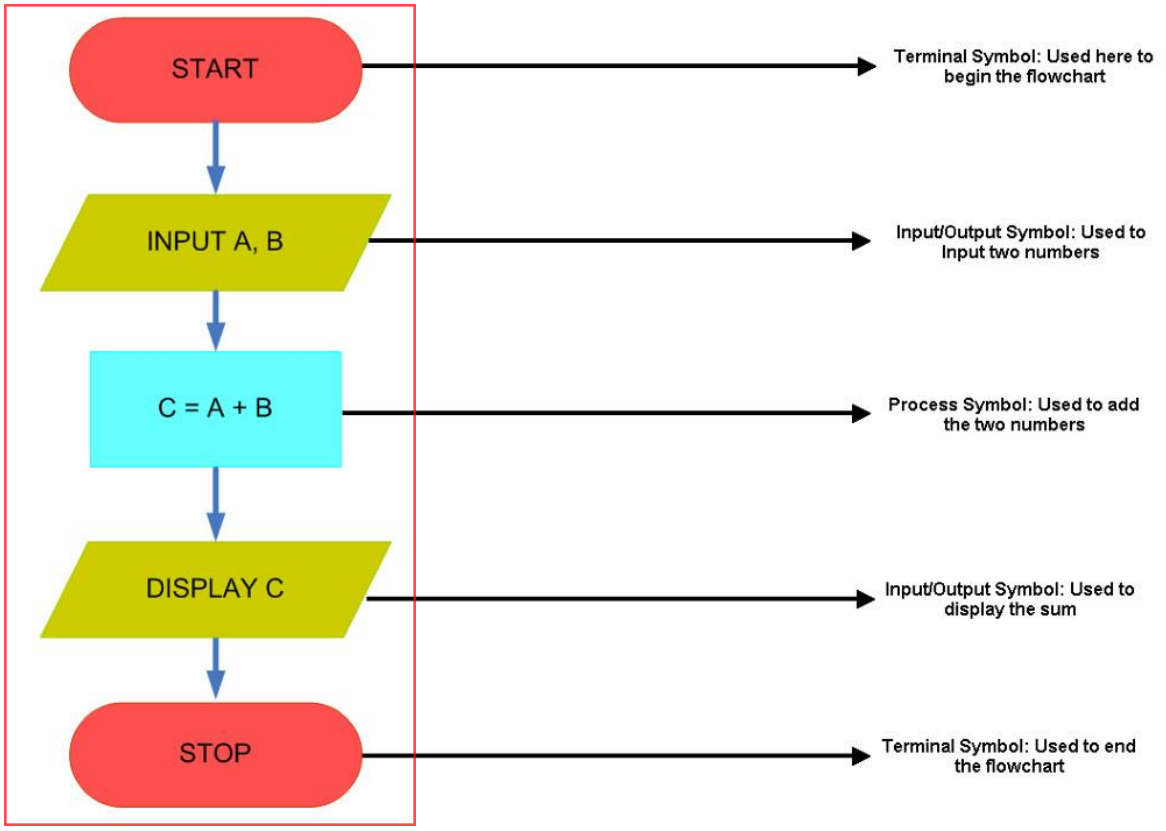


Figure 3.3: Example of Flowchart

Here, the step in which the values of the two variable are added and assigned to a third variable is considered to be a process and is represented by a rectangle. Sometimes, flowcharts span across several pages. In such cases, symbols called connectors are used to indicate the location of the joins. These connectors help to identify the flow across the pages.

It is essential to design a flowchart as a tool for easy understanding of the logic, as the programmer would use it as a base algorithm to write a code for the program. Some of the essential points to be considered while drawing a flowchart are as follows:

- Initially concentrate on the logic of the problem and draw out the main path of the flowchart
- Maintain consistent level of detail for a flowchart
- Must not contain minute details. Only the essential and meaningful steps need to be represented
- Common and easy to understand words should be used
- Consistent usage of variable names
- Flow should be from left to right and top to bottom
- Must have only one START and one STOP point
- Should be simple

There are many advantages of using flowcharts. Some of the advantages of using flowcharts are as follows:

- **Easy to understand** – It is easy to understand as it is a pictorial representation of the logic
- **Effective analysis of problem** – Macro flowchart charts the main logic of the system and can be further broken down for detailed study and analysis
- **Effective joining of different parts** – Each programmer is responsible for designing a part of the large software systems and thus helps to link the various parts of the system
- **Ease in coding** – Flowchart acts as a roadmap and thus helps the programmer to develop an error-free code very fast
- **Systematic debugging** – Flowchart helps to detect, locate, and remove errors from programs in a systematic manner
- **Systematic testing** – Flowcharts ensures in correct selection of data for conforming with the working of the logic

Besides the advantages mentioned, flowcharts have some disadvantages. They are as follows:

- **Time consuming** – Flowcharts take more time to draw
- **Difficult to change** – Redrawing a flowchart is tedious as it uses symbols to display the program logic. Many companies use tools to draw a flowchart
- **No standards** – Flowcharts does not follow any standard as far as the amount of detail that should be included

3.4.1 Best Practices for Drawing Flowcharts

The best practices for drawing flowcharts are as follows:

- The direction of the arrow flow should be to one side either from top to bottom or from left to right.
- Standard symbols must be used in a flowchart so that it can be understood by all.
- All the symbols in a flowchart must be named appropriately.
- Use the connector symbol for complex flowcharts.
- The size of the symbols used in a flowchart must be consistent.

3.5 Pseudocodes

The word pseudo means false. As the name suggest, pseudocode is not the actual code. It is a method of algorithm writing which uses a certain standard set of words which makes it resemble a code. However, pseudocode cannot be complied or executed as a code.

Consider the following example for displaying a 'Hello' message. The pseudocode will be as follows:

Example 1:

```
BEGIN
    DISPLAY 'Hello'
END
```

Similar to a flowchart, each pseudocode must start with the word **BEGIN** or **START**, and end with **END** or **STOP**. The statements between **START** and **END** are English phrases and indented to make the word **START** and **END** stand out. To display some value, the word **DISPLAY**, **WRITE**, or **PRINT** is used. If the value to be displayed is a constant value, it is enclosed within quotes. Similarly, to accept a value from the user, the word **INPUT** or **READ** is used.

As pseudocode is a planning analysis tool. It is flexible and is used for planning program logic. Each statement specifies a single instruction and is written on a separate line. Each statement needs to be indented to show hierarchy. These statements can be directly translated into computer programming language instructions. When the pseudocode is converted to a programming language it loses its flexibility as it requires the programmer to follow the syntax of that specific programming language.

Consider the pseudocode for accepting two numbers from a user and displaying the sum of the two numbers.

Example 2:

```
BEGIN
    INPUT A, B
    DISPLAY A + B
END
```

In this pseudocode, the user enters two values, which are stored in memory and can be accessed as A and B respectively. Such named locations in memory are called variables. The next step in the pseudocode displays the sum of the values present in variables A and B.

However, the pseudocode can be modified to store the sum of variables in a third variable, and then display the value stored in this third variable as shown in the example 3.

Example 3:

```
BEGIN
    INPUT A, B
    C = A + B
    DISPLAY C
END
```

A set of instructions or steps in a pseudocode is collectively called a construct. There are three types of programming constructs namely, sequence, selection, and iteration constructs.

Note - Programming constructs will be covered in detail in the later sessions.

Some of the rules to be followed while writing pseudocodes are as follows:

- The pseudocode must be easy to understand by all and not just the programmer. The activities/ steps should be described in simple statements.
- The variables mentioned in the pseudocode must be self-descriptive. Avoid using abbreviations and shortened versions of words in the pseudocode.
- The pseudocode must not contain actual programming code but should have only logical steps to show how to operate a code.

Some of the advantages of using pseudocode are as follows:

- **Easy to create** – It is easy to create as it uses English like languages
- **No symbols** – Does not use any special symbols to display the program flow
- **No specific syntax** – Syntax is not specific
- **Easy to translate** – Each statement can be translated to any high-level language
- **Reduces time** – It helps to reduce time in coding, testing, and debugging systems

Besides the advantages mentioned, pseudocodes have some disadvantages. They are as follows:

- **Lack of standards** – Pseudocode is unstructured and there is no standardized way to write a pseudocode.
- **Do not focus on big picture** – The focus of a pseudocode is on a certain detailed part of the software program. A programmer may not consider the entire software package when writing a pseudocode.

3.5.1 Best Practices for Writing Pseudocodes

The best practices for writing pseudocodes are as follows:

- The vocabulary used to write a pseudocode must be simple to understand and should be on the lines of structured programming.
- Each statement in the pseudocode must be written in a separate line.
- The keywords, procedure names, and module names must be capitalized in pseudocode.
- Use descriptive names that can be understood by non-technical persons also.
- Maintain consistency when using generic terms such as DISPLAY, PRINT, END, and so on.
- Indent the statements as required to show logical hierarchy.

3.6 Check Your Progress

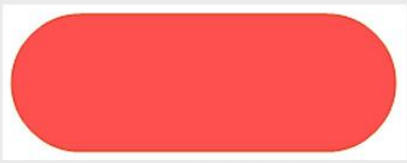
1. Algorithm is a set of steps listed in _____ language.

(A)	Simple	(C)	Complex
(B)	Symbolic	(D)	Unknown

2. Which of the following options are the two types of algorithms that are used?

(A)	Flowcharts	(C)	Pseudocodes
(B)	Programming	(D)	Testing

3. Which of the following symbol marks the start or end of the program?

(A)		(C)	
(B)		(D)	

4. To display some value, the word _____ or _____ is used in pseudocode.

(A)	DISPLAY	(C)	TYPE
(B)	WRITE	(D)	REVEAL

5. What is the function of connectors in a flowchart?

(A)	To begin the flowchart	(C)	To indicate the location of joins
(B)	To display the sum	(D)	To terminate the flowchart

3.6.1 Answers

1.	A
2.	A, C
3.	A
4.	A, B
5.	C

Session 3

Flowcharts and Pseudocodes

- An algorithm can be defined as a procedure, formula, or recipe for solving a problem. It consists of a step of steps that help to arrive at a solution.
- A flowchart is a graphical representation of an algorithm. It charts the flow of instructions or activities in a process. Each activity in a flowchart is depicted using symbols.
- A flowchart begins with the START or BEGIN keyword, and ends with the END and STOP keyword.
- The DISPLAY keyword is used to display some value to the user in a flowchart.
- Pseudocode is not actual code. It is a method of algorithm writing which uses a certain standard set of words which makes it resemble a code.
- Each pseudocode must start with the word BEGIN or START, and end with END or STOP.
- The word DISPLAY, PRINT, or WRITE is used to display some value in pseudocode.

Session - 4

Selection Constructs

Welcome to the Session, **Selection Constructs**.

This session explains the IF and IF...ELSE statements. This session also explains the multiple selection statements, nested IF...ELSE statements, and case conditions.

In this Session, you will learn to:

- Explain IF statement
- Explain IF...ELSE selection construct
- Explain multiple selection statements
- Explain nested IF...ELSE statements
- Explain case construct

Session 4

Selection Constructs

4.1 Introduction

A programmer's job is to write instructions to achieve a goal or solution for a given problem. When developing a program, a programmer may come across a condition in the program, where the path of execution can branch into two or more options. Such constructs are referred to as programming, selection, conditional, or branching constructs.

4.2 IF Statement

The **IF** construct is a basic selection construct. Consider an example where the customer is given a discount if purchases of over $100 are made. Here, each time a customer is billed, a part of the code has to check to see if the bill amount exceeds $100. If it does exceed the amount, then it must deduct 10% of the total amount, otherwise nothing must be deducted. The pseudocode for the scenario will be as follows:

```
IF customer purchases items worth more than $100
    Give 10% discount
```

The construct used here is to illustrate the process of taking a decision using an **IF** statement. The general form of an **IF** statement or construct is as follows:

```
IF condition
    Statements → Body of the IF Construct
END IF
```

An **IF** construct begins with **IF** followed by the condition. If the condition evaluates to True, then the control is passed to the statements within its body. If the condition returns as False, the statements within the body are not executed, and the program flow continues with the next statement after the **END IF**. The **IF** construct must always end with an **END IF**, as it marks the end of the construct.

Example 1 uses the **IF** construct to find whether a number is even.

Example 1:

```
BEGIN
    INPUT number
    rem = number MOD 2
    IF rem=0
        Display "Number is even"
    END IF
END
```

The code accepts a number from the user, performs the MOD operation on it, and checks to see if the remainder is zero. If it is, then it displays a message, otherwise it just exits.

A flowchart for the pseudocode is shown in figure 4.1.

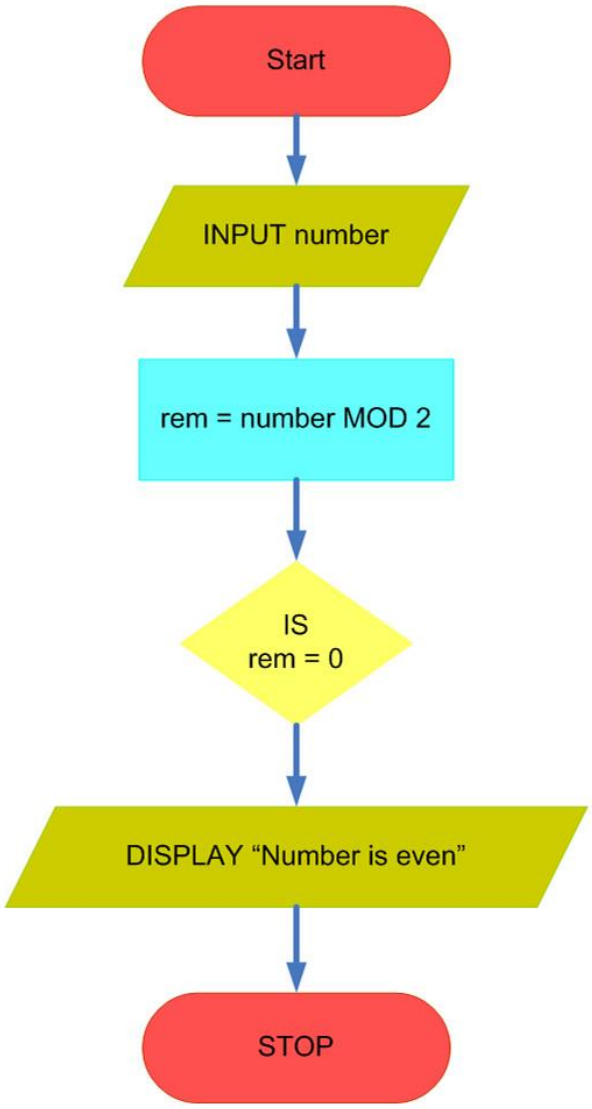


Figure 4.1: Flowchart of the IF Pseudocode

The syntax for the **IF** statement in C language is as follows:

```
if (condition)
        {
        Statements;
        }
```

Example 2 shows the pseudocode that would be written in C.

Example 2:

```
/*A C program using the IF construct */
#include <stdio.h>
void main ()
{
int number, rem;
printf ("Please enter a number: ");
scanf ("%d", &number);
rem=number%2;
if(rem==0)
     {
     printf("Even Number");
     }
}
```

4.3 IF...ELSE Statement

In example 2, the even numbers are identified. For easier understanding, it would be better to show that the number is odd, when the value received is not an even number. This can be specified in an easy way by using the **IF...ELSE** statement. The **IF...ELSE** statement enables a programmer to make a single comparison and execute the steps depending on whether the result of the comparison is True or False.

The general form of the **IF...ELSE** statement is as follows:

```
IF condition
  Statement set1
ELSE
  Statement set2
END IF
```

The syntax for the **IF...ELSE** construct in C language is given as follows:

```
if(condition)
        {
        statement set1;
        }
else
        {
        statement set2;
        }
```

If the result of the condition is True, statement set1 is executed, otherwise statement set2 is executed, but never both. So, a more efficient code for the even number is shown in example 3.

Example 3:

```
BEGIN
INPUT number
rem=number MOD 2
IF rem=0
       DISPLAY "Even Number"
ELSE
  DISPLAY "Odd Number"
END IF
END
```

The flowchart for the pseudocode is shown in figure 4.2.

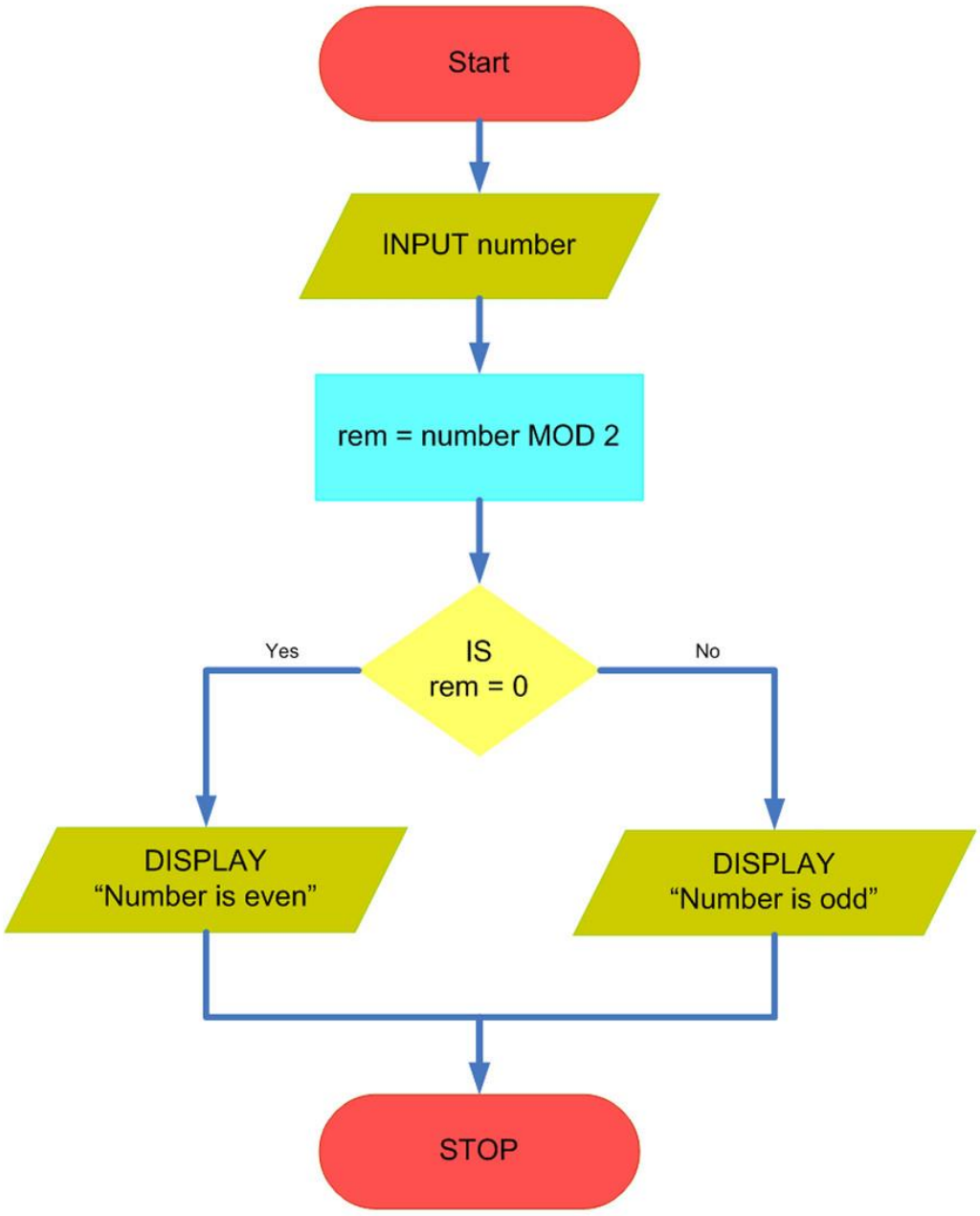


Figure 4.2: Flowchart of the IF...ELSE Pseudocode

4.4 Multiple Selection Statements

The **IF...ELSE** construct helps to reduce the complexity of a program and enhances its efficiency. It also helps to improve the readability of the code. The **IF** statement is generally used for single condition based problems. For more than one condition, the **AND** statement can be used in conjunction with the **IF** statement.

Consider a scenario to classify a supplier as a Most Valuable Supplier (MVS). For this, the organization must check that the supplier has been with them for the last 10 years and has done a total business of more than $500000. These two conditions must be satisfied to consider a supplier as a MVS. Example 4 shows the pseudocode for this scenario.

Example 4:

```
BEGIN
INPUT YearsWithUs
INPUT BizDone
IF YearsWithUs >= 10 AND BizDone >= 500000
     DISPLAY "Classified as an MVS"
ELSE
     DISPLAY "A little more effort required"
END IF
END
```

Note - The expression used for **AND** in C language is **&&**.

Example 5 shows the pseudocode that would be written in C.

Example 5:

```
/* C snippet depicting the AND operator in IF */
if(YearsWithUs >= 10 && BizDone >= 500000)
     {
     printf("Classified as an MVS");
     }
else
     {
     printf("A little more effort required");
     }
```

In real life situations, there can be plenty of conditions that a programmer may need to check. The **AND** operator can be used to connect the conditions conveniently.

Consider an example where the organization decides to give the supplier the MVS status when either of the two conditions is fulfilled. That is, either if a supplier has been with them for 10 years or has done a business for more than $500000. To fulfill this condition, the **AND** operator will be replaced by the OR operator in the pseudocode.

4.5 Nested IF...ELSE Statements

Another way to combine two conditions without using the **AND** operator, is by using nested **IF**...**ELSE** statements. A nested **IF** is an **IF** statement inside another **IF** statement. Consider the same example to recognize the MVS status of a supplier. The pseudocode for this is shown in example 6.

Example 6:

```
BEGIN
INPUT YearsWithUS
INPUT BizDone
IF YearsWithUs >= 10
     IF BizDone >= 500000
          DISPLAY "Classified as an MVS"
     ELSE
          DISPLAY "A little more effort required"
     END IF
ELSE
     DISPLAY "A little more effort required"
END IF
END
```

The code performs the same function of selecting the supplier without using the **AND** statement. However, there is an **IF** (BizDone is more than $500000) within another **IF** (YearsWithUs is more than 10). The first **IF** statement checks to see if the supplier has been with the organization for more than 10 years. If this returns as False, it rejects it as a MVS, else it enters the **IF** statement to check if the BizDone is more than $500000. If the second statement is True, then the supplier is classified as MVS, else it displays a message rejecting it as an MVS.

The flowchart for the pseudocode is shown in figure 4.3.

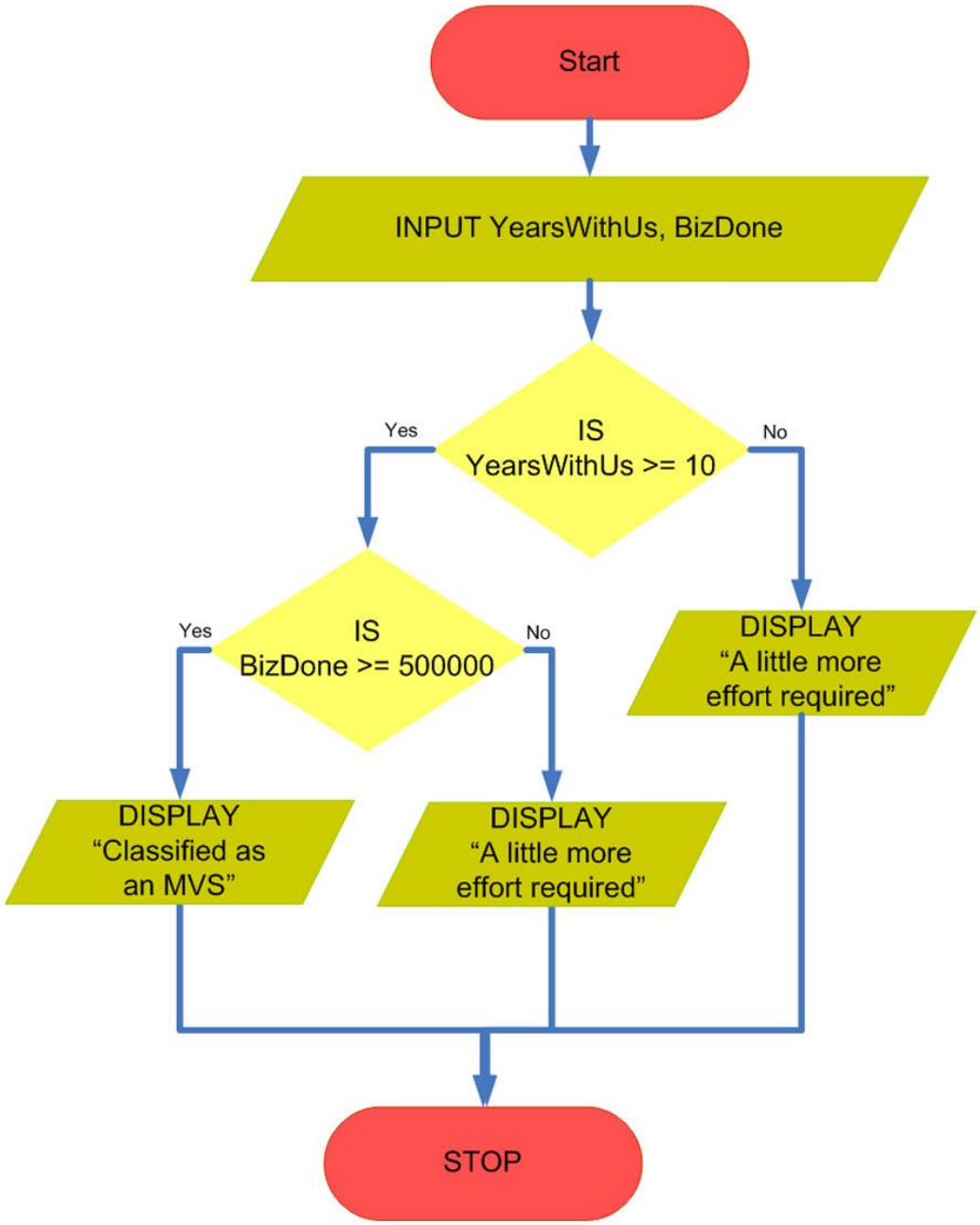


Figure 4.3: Flowchart of the Nested IF...ELSE Pseudocode

As seen in the flowchart, the MVS rejection statement has to be written twice. The programmer will also have to write more code, and the complier has to evaluate two **IF** statements, thus taking more time. This does not mean that nested **IF** statements are inefficient. It depends on the situation where they are used. Sometimes using the **AND** operator is better, and in other cases nested **IF** statements are more efficient.

Consider an example where nested **IF**s will prove more efficient than using the **AND** operator. A company allots basic salary to its employees based on the criteria specified in table 4.1.

Grade	Experience	Salary
E	2	2000
E	3	3000
M	2	3000
M	3	4000

Table 4.1: Salary to Employees

Therefore, for example, an employee with grade E and two years of experience would receive a salary of $2000, and an employee with the same grade but 3 years of experience would get a salary of $3000.

The pseudocode for the problem using the **AND** operator is shown in example 7.

Example 7:

```
BEGIN
INPUT grade
INPUT exp
IF grade = "E" AND exp = 2 THEN
     salary = 2000
ELSE IF grade = "E" AND exp = 3 THEN
     salary = 3000
ELSE IF grade = "M" AND exp = 2 THEN
     salary = 3000
ELSE IF grade = "M" AND exp = 3 THEN
     salary = 4000
END IF
END
```

The first **IF** checks the employee's grade and his experience. If the grade is E and experience is 2 years, he is allotted a salary of $2000, else if his grade is E, but he has got 3 years of experience, then his salary will be $3000.

If both the conditions evaluate to false, then the next **ELSE IF** statement similarly checks the employee's

grade as well as experience and allots the salary.

Suppose the grade of an employee is E and he has got 2 years of experience. His salary is calculated in the first **IF**. The **ELSE IF** part of after the first **IF** is not executed. However, it goes to the second **ELSE IF** and checks the condition which ofcourse would be false, so it checks the next **ELSE IF** clause. Therefore, it unnecessarily goes through these steps.

If a company had about 15 grade levels, the program would have to go through all of their **IF** and **ELSE** clauses. This would be time consuming and would waste the system resources. Hence, this would not be considered as an efficient way to write a code.

Consider the same example pseudocode which is modified by using nested **IF** statements is shown in example 8.

Example 8:

```
BEGIN
INPUT grade
INPUT exp
IF grade = "E"
     IF exp = 2
          salary = 2000
     ELSE
          IF exp = 3
               salary = 3000
          END IF
     END IF
ELSE
     IF grade = "M"
          IF exp = 2
               salary = 3000
          ELSE
               IF exp = 3
                    salary = 4000
               END IF
          END IF
     END IF
END IF
```

If an employee has a grade of E and has two years of experience, then his salary is calculated to be $2000 in the first step of the first **IF** statement. After this, the code is exited as it does not need to execute any of the **ELSE** clauses. Therefore, it does not go through unnecessary **IF**s. Hence, the code is more efficient and the program runs faster.

4.6 Case Conditions

Though the nested **IF** code is more efficient, it is also more difficult to write. Therefore, a programmer needs to maintain the efficiency and also must write a manageable code. For this, programming languages provide a special construct which is called the **DO CASE...END CASE** construct. This construct is used when a variable is to be successively compared against different values.

Consider a scenario to re-write the nested **IF** codes to include the **DO CASE**. Example 9 shows the method to allocate the salary to its employees based on the criteria based on the earlier example.

Example 9:

```
BEGIN
INPUT grade
INPUT exp
DO CASE (grade)
CASE 'E'
     IF exp = 2
          salary = 2000
     ELSE
          IF exp = 3
               salary = 3000
          END IF
     END IF
CASE 'M'
     IF exp = 2
          salary = 3000
     ELSE
          IF exp = 3
               salary = 4000
          END IF
     END IF
OTHERWISE:
     DISPLAY "Invalid Grade Entered!"
END CASE
END
```

If the operator does not match with any of the case statements, then the statements in the OTHERWISE section get executed. If the user enters a grade other than E or M, he gets a message that an invalid grade was entered.

The **DO CASE** is known as 'Switch Case' in C. The syntax in C will be as follows:

```
switch (expression)
{
    case const-expr:
        statement set;
        break;
    case const-expr:
        statement set;
        break;
    default
        statement set;
}
```

The usage in C is very much similar to the pseudocode's **DO CASE** statement. However, before starting a next **CASE** statement, the earlier statement must be concluded by using a break statement. The break statement breaks out of the switch case construct and continues execution at the instruction following this construct.

The default statement in C syntax ensures that the chunk of code within the block is executed whenever there is no matching case for the expression passed. Therefore, it can be mapped to the **OTHERWISE** statement in the pseudocode.

Example 10 shows the pseudocode written for C language.

Example 10:

```
/* C program using the SWITCH CASE construct */
#include <stdio.h>
void main()
{
char grade;
int exp, salary;
printf("Enter your Grade: ");
scanf("%c", &grade);
printf("Enter your experience (in years): ");
scanf("%d", &exp);
```

```
switch(grade)
{
case 'E' :
     if(exp==2)
     {
     salary=2000;
     printf("Salary is %d",salary);
     }
     else
          {
          if(exp==3)
          {
          salary=3000;
          printf("Salary is %d",salary);
          }
          }
          printf("Years of Experience is Incorrect");
     break;
case 'M' :
     if(exp==2)
     {
     salary=3000;
     printf("Salary is %d",salary);
     }
     else
          {
          if(exp==3)
          {
          salary=4000;
```

```
            printf("Salary is %d",salary);
            }
            }
            printf("Years of Experience is Incorrect");
      break;
default:
      printf("Invalid Grade Entered!");
}
}
```

4.7 Check Your Progress

1. The _____ construct is a basic selection construct.

(A)	IF	(C)	AND
(B)	OR	(D)	DO CASE

2. Identify the correct form of the IF...ELSE statement.

(A)	IF
	  Statement set1
	ELSE
	  Statement set2
	END IF
		(C)	IF condition
			  Statement set1
			ELSE
			  Statement set2

(B)	IF condition
	  Statement set1
	  Statement set2
	END IF
		(D)	IF condition
			  Statement set1
			ELSE
			  Statement set2
			END IF

3. Which of the following are a part of multiple selection statements?

(A)	AND	(C)	OR
(B)	NOR	(D)	CAN

4. The _____ statement can be mapped to the **OTHERWISE** statement in the pseudocode.

(A)	Default	(C)	Applied
(B)	IF	(D)	AND

5. The _____ construct is used when a variable is to be successively compared against different values.

(A)	`IF`	(C)	`IF ELSE`
(B)	`DO CASE`	(D)	`AND OR`

4.7.1 Answers

1.	A
2.	D
3.	A, C
4.	A
5.	B

Session 4

Selection Constructs

- Conditions in a program where the path of execution may branch into two or more options are referred to as programming, selection, conditional, or branching constructs.
- The basic selection construct is an **IF** construct.
- The **IF...ELSE** construct enables the programmer to make a single comparison and execute the steps depending on whether the result of the comparison is True or False.
- The AND statement can be used in conjunction with the IF statement for more than one condition.
- A nested **IF** is an **IF** statement inside another **IF** statement.
- The **DO CASE** construct is used when a variable is to be successively compared against different values.
- The break statement breaks out of the switch case construct and continues execution at the instruction following this construct.


Action may not always bring happiness,
but there is no happiness without action

Session - 5

Operators

Welcome to the Session, **Operators**.

This session explains operators and lists the different types of operators. It also explains the different types of operators, namely, arithmetic, relational, and logical. Finally, this session helps to identify the precedence of different operators in an expression.

In this Session, you will learn to:

- Define operators
- List the different types of operators
- Describe the use of arithmetical operators
- Describe the use of relational operators to make comparisons
- Explain the process of associating selections with logical operators
- Identify the precedence of operators in an expression

5.1 Introduction

All programming languages provide some mechanism using which a programmer can perform various operations on data stored in variables. The operations can be arithmetic such as addition, division, or even comparison where one variable is compared to another variable. These kinds of operations are performed using operators.

5.2 Operators

Operators are a set of symbols that help to manipulate or perform some sort of function on data. For example, when two numbers are added, the + symbol is used, which is the addition operator.

Consider the following statement:

A + B

This statement would add the contents of the two variables A and B. The + symbol in the statement is called the operator and the operation performed is addition. As A and B are the variables on whom the operation is performed, they are called operands. The combination of both the operator and the operands is called an expression. So the expression in this case is the entire statement A + B.

Operators can be classified into three types. These three types are as follows:

- Arithmetic Operators
- Relational Operators
- Logical Operators

5.3 Using Arithmetic Operators

Mathematical calculations are always required to be performed by computers especially in the field of commerce and scientific research. Computers have to perform these mathematical calculations accurately and cannot afford to make mistakes. For this purpose, arithmetic operators are used by computers.

Arithmetic operators help to manipulate numeric data and perform common arithmetic operation on the data. Table 5.1 shows a list of arithmetic operators that are common to most programming languages. The last column in the table also lists the equivalent of the arithmetic operator in C language.

Operator	Description	Example	Result	C Equivalent
+	Addition	9 + 2	11	+
-	Subtraction	9 - 2	7	-
/	Division	9/2	4.5	/
*	Multiplication	9 * 2	18	*
^	Exponentiation	9^2	81	^

Operator	Description	Example	Result	C Equivalent
MOD	Modulus	9 MOD 2	1	%
-	Negation	-9	-9	-

Table 5.1: List of Arithmetic Operators

The negation operator requires only a single operand. All other operators require two operands. Hence, the negation operator is also known as a unary operator. The other operators are known as binary operators.

Apart from these operators, certain languages such as C and Java provide two different types of operators called the Increment operator (++) and the Decrement operator (--). These operators also require only a single operand and hence, they also fall under the unary operator category.

Note - Unary operators use only one operand to perform different kind of operations. Binary operators use two operands to perform operations.

Consider the following example to explain increment operator:

Example 1:

```
BEGIN
     A = 5
     A = A + 1
     DISPLAY A
END
```

The pseudocode first assigns 5 to variable A, and then in the next step adds 1 to it. In other words, it increments its own value by 1. In the end, it displays the value of A, which would be 6.

The following example shows the code being rewritten using the increment operator.

Example 2:

```
BEGIN
     A = 5
     A++
     DISPLAY A
END
```

In line 3, the increment operator ++ is used. This operator increments the value stored in variable A by 1. Instead of using A = A + 1, the expression used is A++. Though increment operators may not seem like a huge advantage to the programmer, it is very helpful when using loops. Loops will be discussed in the next session.

The C syntax for using the increment operator is shown in example 3.

Example 3:

```
/* Program depicting the use of the increment operator */
     #include <stdio.h>
     void main ()
     {
     a = 5;
     a++;
     printf("Value of a : %d", a);
     }
```

The decrement operator is similar to the increment operator except that it decrements the value of the variable by 1.

5.3.1 Precedence between Arithmetic Operators

Consider example 4 to understand the precedence of arithmetic operators.

Example 4:

```
BEGIN
     DISPLAY 4+3*6/2-5+5
END
```

Table 5.2 shows the order in which the arithmetic operators precede over other arithmetic operators.

Precedence	Operator	Description
1	++	Increment
2	--	Decrement
3	*, /, MOD	Multiplication, Division, Modulus
4	+, -	Addition, Subtraction

Table 5.2: Precedence of Arithmetic Operators

Note that some of the operators have the same precedence. In this case, the equation with all the symbols having equal precedence would be evaluated from left to right. The steps to solve the equation are as follows:

The equation is 4+3*6/2-5+5

The equation is 4+3*6/2-5+5

1. From the different operators used, * and / have the highest precedence.

   4+**[3*6/2]**-5+5

2. However, since * and / have the same precedence, they are evaluated from left to right.

   4+**[18/2]**-5+5

   4+9-5+5

3. + and – have the same precedence. So they are evaluated from left to right.

   [4+9]-5+5

4. **[13-5]**+5

5. **8+5**

6. **13**

5.4 Using Relational Operators

Relational operators compare two or more values or expressions and always return either 'True' or 'False'. For example, 2 is greater than 9 would return 'False'.

Relational operators are binary operators. Table 5.3 shows a list of relational operators common to most languages.

Operator	Description	Example	Result	C Equivalent
<	Less than	2<9	True	<
<=	Less than or Equal to	2<=9	True	<=
>	Greater than	2>9	False	>
>=	Greater than or Equal to	2>=9	False	>=
=	Equal to	2=9	False	==
<>	Not Equal to	2<>9	True	<>

Table 5.3: List of Relational Operators

In C language, == is used as the operator to check the equality of two variables. The = operator which is used as a relational operator in general pseudocode, is only used in C to assign values to variables.

5.4.1 Precedence between Relational Operators

Some arithmetic operators have precedence over other arithmetic operators. There is no such precedence among relational operators. Therefore, they are always evaluated from left to right.

5.5 Using Logical Operators

Logical operators are used in situations where multiple conditions need to be satisfied. Logical operators combine the results of several comparisons, as required, to present a single answer. Logical operators also return the results in either 'True' or 'False'.

Table 5.4 shows a list of logical operators.

Operator	Description	C Equivalent
AND	Result is 'True' only when both conditions are 'True'	&&
OR	Result is 'True' when either of the two conditions is 'True'	||
NOT	Operates on a single value and converts 'True' to 'False' and vice-versa	!

Table 5.4: List of Logical Operators

The AND Operator

Consider a scenario, to classify a student into Grade B, the score should be more than 80 marks but less than 90. There are two conditions that must be fulfilled in the given statement. First, check if the student has scored more than 80. At the same time, also ensure that the student has scored less than 90. Only if both the conditions are True, then the student is classified as Grade B. Therefore, if the student gets 95, while the first condition is met, the second condition is not met. So the student cannot get a Grade B. In such scenarios where both the conditions are to be considered, the **AND** operator is used. Example 5 shows the pseudocode for the scenario.

Example 5:

```
BEGIN
IF score > 80 AND score < 90
   DISPLAY "Student is in Grade B"
END
```

The OR Operator

Consider a scenario, where a shopkeeper offers his customers a 10% discount if the customers either buy 5 items from his shop or if the total bill exceeds $100. The **AND** operator cannot be used here, because even if the customer fulfils one of the conditions, the discount has to be provided. In such a scenario, the **OR** operator is used.

First, check if the customer has bought more than 5 items. If this condition is met, the student is eligible for the discount, regardless of the second condition. However, if the customer bought fewer than 5 items, check if the bill amount exceeds $100. If this condition is met, the customer gets a discount. However, if both the conditions are not met, there is no discount. Therefore, the **OR** statement evaluates to True even if one of the condition is True. Example 6 shows the pseudocode for the scenario.

Example 6:

```
BEGIN
IF itemcount > 5 OR amount > 100
   DISPLAY "Customer gets a discount"
END
```

The NOT Operator

The **NOT** operator is a unary operator. This means that the **NOT** operator requires a single operand. The **NOT** operator is used to perform logical negation on its operand. This operator is used to turn True to False and vice-versa. Therefore, if an expression returns True, preceding it with a **NOT** operator returns False. Table 5.5 shows the results of using the **NOT** operator.

Condition	NOT Condition
False	True
True	False

Table 5.5 Results of Using the NOT Operator

Consider a scenario, where to receive a driver's license, a person's age must be over 18 years. Example 7 shows the pseudocode for the scenario.

Example 7:

```
BEGIN
If NOT age>18
 DISPLAY "Ineligible for Driver's license"
END
```

5.5.1 Precedence between Logical Operators

Similar to arithmetic operators, logical operators also have precedence. Table 5.6 shows the precedence order for logical operators.

Precedence	Operator
1	**NOT**
2	**AND**
3	**OR**

Table 5.6: Precedence of Logical Operators

If there is more than one operator of the same precedence in the equation, then they get evaluated from right to left. Consider the expression given in example 8.

Example 8:

```
Expression: 1>3 OR 0<1 AND NOT 5>8 AND 25<100
Evaluation: False OR True AND NOT False AND True
```

The steps using which the condition is evaluated are as follows:

1. As **NOT** has the highest precedence, it is evaluated first.

   False **OR** True **AND** [**NOT** False] **AND** True

2. Next, **AND** is the operator of highest precedence, and there is more than one **AND** in the expression, hence it is evaluated from right to left.

   False **OR** True **AND** **[**True **AND** True**]**

3. True **AND** True in this step results in True.

   False **OR** **[**True **AND** True**]**

4. False **OR** True results in True as there is one True in this equation.

   [False **OR** True**]**

5. True

5.6 Precedence of Operators in an Expression

When an expression uses more than one type of operator, then the order of precedence has to be established between the different types of operators. Table 5.7 shows the precedence among the different types of operators.

Precedence	Type of Operator
1	Arithmetic
2	Relational
3	Logical

Table 5.7: Precedence Among the Different Types of Operators

If an equation involves all three types of operators, then the arithmetic operators are evaluated first, followed by the relational operators, and then the logical operators. After this, the precedence of operators within a particular type of operator is evaluated.

Consider example 9 to understand the precedence among the different types of operators.

Example 9:

```
Expression:2*3+4/2>3 AND 3<5 OR 10<9
```

The steps for evaluation are as follows:

1. **[2*3+4/2]** > 3 AND 3<5 OR 10<9

   First the arithmetic operators are evaluated. The order of precedence for the evaluation of arithmetic operators is shown in table 5.2.

2. **[[2*3]+[4/2]]** > 3 AND 3<5 OR 10<9

3. **[6+2]** > 3 AND 3<5 OR 10<9

4. **[8 > 3]** AND [3<5] OR [10<9]

   Next, the relational operators are evaluated from left to right.

5. `[True AND True] OR False`

   Finally, the logical operators are evaluated. The AND operator takes precedence over the OR operator.

6. `True OR False`

7. `True`

5.6.1 The Parenthesis

The compiler sets the precedence rules. Sometime for certain formulas, the programmer may need to override the precedence rules. These rules can be overridden with the help of parenthesis. Using parenthesis, the programmer can specify the part of the equation that needs to be solved first. Consider example 10 which shows the usage of parenthesis.

Example 10:

```
Expression:4+5*(7-1)
```

The steps to solve the expression are as follows:

1. According to the precedence rules, the * has higher precedence over any other operator in the equation, but as 7-1 is enclosed in parenthesis, it is evaluated first.

   ```
   4+5*6
   ```

2. As there are no other parentheses, the * operator is evaluated as per the precedence rules.

   ```
   4+30
   ```

3. `34`

5.7 Check Your Progress

1. _____ are a set of symbols that help to manipulate or perform some sort of function on data.

(A)	Operators	(C)	Expressions
(B)	Operands	(D)	Rationales

2. Which of the following is the C equivalent symbol for MOD?

(A)	*	(C)	%
(B)	^	(D)	-

3. Which of the following has the first precedence among the arithmetic operators?

(A)	Increment	(C)	Multiplication
(B)	Decrement	(D)	Modulus

4. Match the following:

Operator		C Equivalent	
(A)	AND	1.	||
(B)	OR	2.	!
(C)	MOD	3.	%
(D)	NOT	4.	&&

(A)	a-3, b-2, c-1, d-4	(C)	a-4, b-1, c-2, d-4
(B)	a-2, b-4, c-3, d-1	(D)	a-1, b-3, c-4, d-2

5. Which of the following will be evaluated in the expression 6/3 > 3?

(A)	6	(C)	3 >3
(B)	6/3	(D)	3

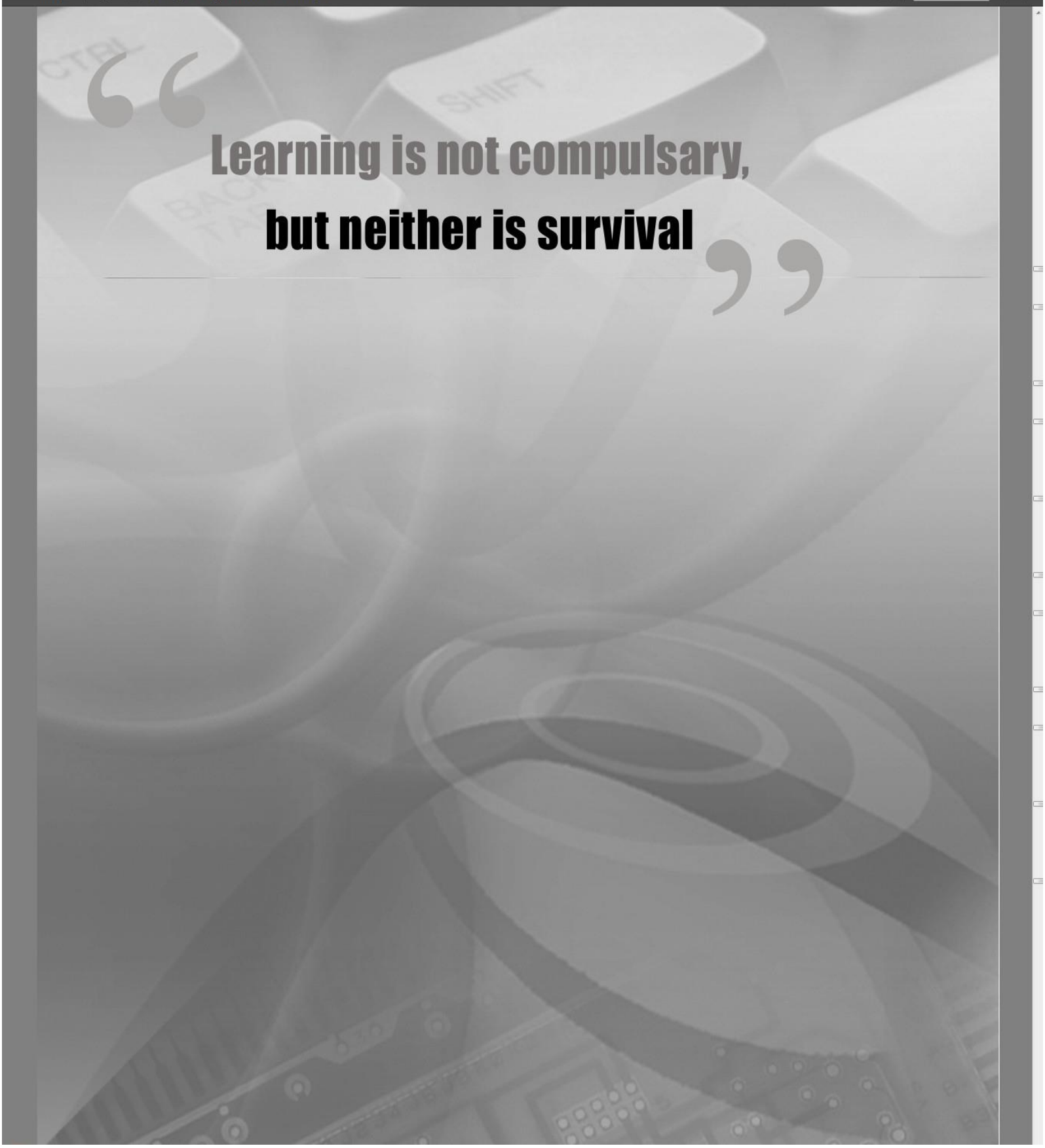
5.7.1 Answers

1.	A
2.	C
3.	A
4.	B
5.	B

Session 5

Operators

- Operators are a set of symbols that help to manipulate or perform some sort of function on data.
- Operators can be classified into three types, namely, arithmetic, relational, and logical operators.
- Arithmetic operators help to manipulate numeric data and perform common arithmetic operation on the data.
- Relational operators compare two or more values or expressions and always return either 'True' or 'False'.
- Logical operators are used in situations where multiple conditions need to be satisfied.
- When an equation involves all three types of operators, then the arithmetic operators are evaluated first, followed by the relational operators, and then the logical operators.
- The programmer can specify the part of the equation that needs to be solved first by using parenthesis.


Learning is not compulsary,
but neither is survival

Session - 6

Iteration Constructs

Welcome to the Session, **Iteration Constructs**.

This session explains the concept of looping constructs. It also lists the different types of loops. Finally, the session explains nested loops.

At the end of this session, you will be able to:

- Explain looping constructs
- Describe the different types of loops
- Explain nested loops

6.1 Introduction

Sometimes, a program in the computer needs to perform some tasks repetitively. For this, there is a special type of construct to perform these tasks. It is called the iterative or looping construct.

6.2 Loops

A computer program is a set of statements, which are normally executed sequentially. Often, it is necessary to repeat certain set of steps a specific number of times or till some specified condition is met. For example, a programmer has to write a program that would display a name 10 times. One of the ways to do this would be by writing the pseudocode as shown in example 1.

Example 1:

```
BEGIN
      DISPLAY "Mike"
      DISPLAY "Mike"
      DISPLAY "Mike"
      DISPLAY "Mike"
      DISPLAY "Mike"
      DISPLAY "Mike"
      DISPLAY "Mike"
      DISPLAY "Mike"
      DISPLAY "Mike"
      DISPLAY "Mike"
END
```

This code is still easier to write, but there can be scenarios where one has to display the name 1000 times. A programmer cannot keep on typing **DISPLAY** statement 1000 times. In this kind of scenarios the iteration construct is used. Example 2 shows the pseudocode to add an iteration construct to the DISPLAY statement.

Example 2:

```
Do loop 1000 times
     DISPLAY "Mike"
End loop
```

The block of statements that appear between the Do loop and End loop statements get executed 1000 times. These statements along with the Do loop and End loop statements are called a looping or iterative construct. Loops enable programmers to develop concise programs which otherwise would require thousands of statements to be executed.

6.3 Types of Loops

The Do loop...End loop was used in the earlier example to show a general form of a loop statement. However, there are several specific types of loops that are supported by most of the programming languages. Some of these loops are as follows:

- The WHILE Loop
- The DO...WHILE Loop
- The REPEAT...UNTIL Loop
- The FOR Loop

6.3.1 The WHILE Loop

The **WHILE** loop repeats a statement or a set of statements until a certain specified condition is met or is true. The general form of the **WHILE** loop is as follows:

```
WHILE condition
DO
     statement set
END DO
```

The condition specifies either a variable or a specific condition. The statements in the statement set will be executed as long as the test expression evaluates to True. If the expression evaluates False, the loop exits.

The modified pseudocode for example 2 using the **WHILE** loop is shown in example 3.

Example 3:

```
BEGIN
cnt=0
WHILE (cnt<1000)
DO
     DISPLAY "Mike"
     cnt=cnt+1
END DO
END
```

The program initializes a variable called 'cnt' to zero, which acts as a counter variable. This variable is used to keep a count or track of number of times the loop has been executed. Additionally, the variable is compared with the condition specified at the beginning of the loop.

First, the expression written next to the keyword **WHILE** is evaluated. If the result evaluates to True, then the statements within the **DO** and **END DO** keywords are executed. As the value of the cnt variable is 0, the condition is True and the body of the loop is executed. The message is displayed and the value of the variable is increased by 1.

Since, it is an iterative statement, the control returns again to the **WHILE** statement. The expression is evaluated once again, and if it returns True, the body of the loop is executed one more time. This process continues until the value of the variable **cnt** is 1000. At this stage, the test expression, 1000<1000 will evaluate to False and the loop will exit. Then, the program execution will continue with the statement immediately following the **WHILE** loop.

The flowchart for the pseudocode is shown in figure 6.1.

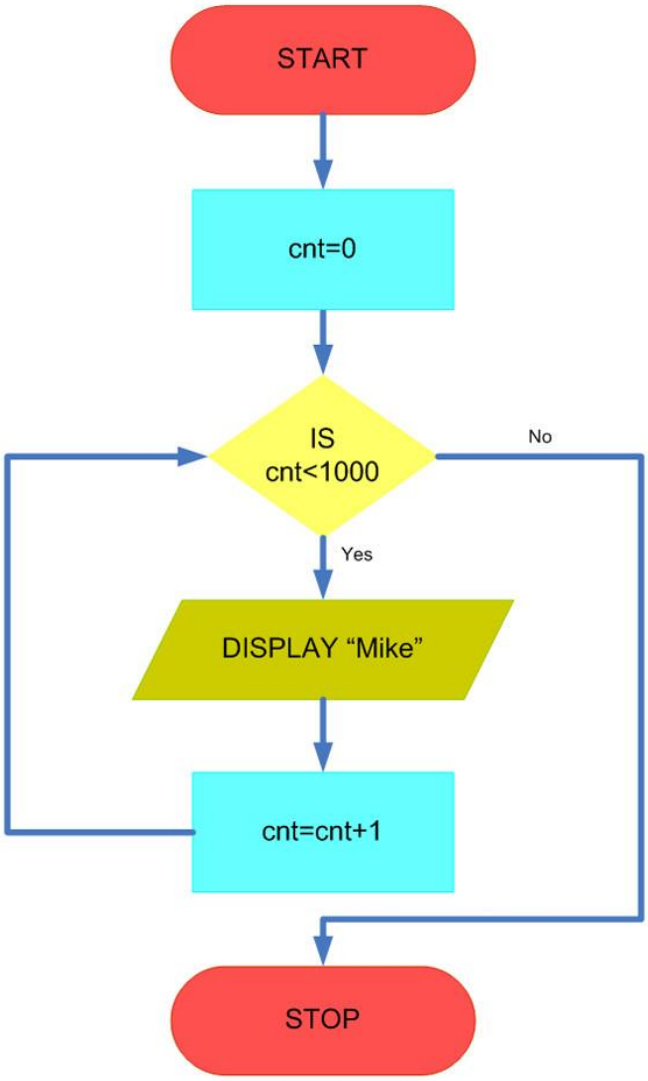


Figure 6.1: Flowchart for Example 3

There is no special symbol for depicting loops in a flowchart. The branching symbol is used to check the condition and manage the flow of the program using flow-lines.

In C, the syntax for the **WHILE** loop is as follows:

```
while(condition)
{
     statement set
}
```

In C, example 3 would be written as shown in example 4.

Example 4:

```
#include<stdio.h>
voidmain()
{
int cnt;
cnt=0;
while(cnt<1000)
     {
     printf("Mike");
     cnt++;
     }
}
```

6.3.2 The DO…WHILE Loop

The **DO**…**WHILE** loop is similar to the **WHILE** loop. However, the **WHILE** loop tests the condition before the body of the loop is executed. Therefore, the body of the loop may not be executed at all if the condition is not satisfied in the first attempt. However, on some occasions it might be necessary to execute the body of the loop before the test is performed. Such situations can be handled with the help of the **DO**…**WHILE** loop. The general form of the **DO**…**WHILE** loop is as follows:

```
DO
Statements
WHILE condition
```

At the start of the **DO** statement, the program executes the body of the loop first. At the end of the loop, the test condition in the **WHILE** statement is evaluated. If the condition is True, the program executes the body of the loop again. This process continues till the condition becomes True. When the condition becomes False, the loop is terminated, and the control goes to the statement that appears immediately after the **WHILE** statement.

As the condition was evaluated at the bottom of the loop, the **DO...WHILE** loop is always executed at least once.

Consider an example where the loop displays a message and accepts a number from the user. The loop continues doing this until the user enters the number 5. Then, it exits the program.

Example 5:

```
BEGIN
DO
      DISPLAY "Enter a number [enter 5 to exit];"
      INPUT num
WHILE (num <> 5)
END
```

The flowchart for example 5 is shown in figure 6.2.

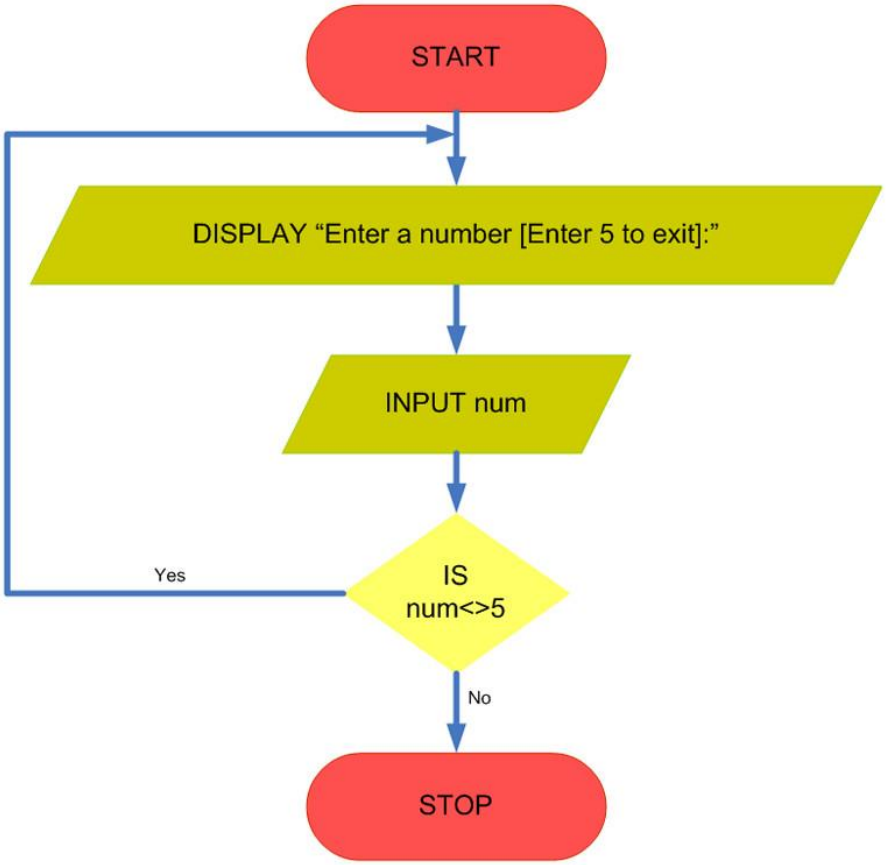


Figure 6.2: Flowchart for Example 5

The C syntax for the **DO...WHILE** loop is as follows:

```
do
{
     statements
}
while (condition)
```

In C, example 3 would be written as shown in example 6.

Example 6:

```
#include<stdio.h>
voidmain()
{
intcnt;
cnt=0;
do
{
     printf("Mike");
     cnt++;
}while(cnt<1000)
}
```

6.3.3 The REPEAT...UNTIL Loop

The **REPEAT...UNTIL** loop is similar to the **DO...WHILE** loop as the statements in the loop body get executed before the condition is evaluated. However, one important difference is that the **DO...WHILE** loop executes while the test condition evaluates to True, that is the **DO...WHILE** loop exits when the condition returns False. In the **REPEAT...UNTIL** loop, the loop executes until the condition evaluates to True. This means that the **REPEAT...UNTIL** loop executes as long as the condition is False.

The general format for the **REPEAT...UNTIL** construct is as follows:

```
REPEAT
     statement set
UNTIL condition
```

The statement set block is executed first, and only then the condition is checked. If it evaluates to False, then the statement set block is executed again. This process repeats till the condition returns True. After the condition returns True, the loop is exited, and execution continues on the line after the **UNTIL** statement.

Consider the scenario for example 5 is modified to use the **REPEAT...UNTIL** loop. This is shown in example 7.

Example7:

```
BEGIN
REPEAT
      DISPLAY "Enter a number [enter 5 to exit];"
      INPUT num
UNTIL (num=5)
END
```

In the earlier example 5, the **DO...WHILE** was used with the condition (num<>5), but in example 6, the condition is changed to (num=5). This is because the **REPEAT...UNTIL** loop exits when the condition evaluates to True, while the **DO WHILE** loop exits when the condition evaluates to False.

The flowchart for example 6 is shown in figure 6.3.

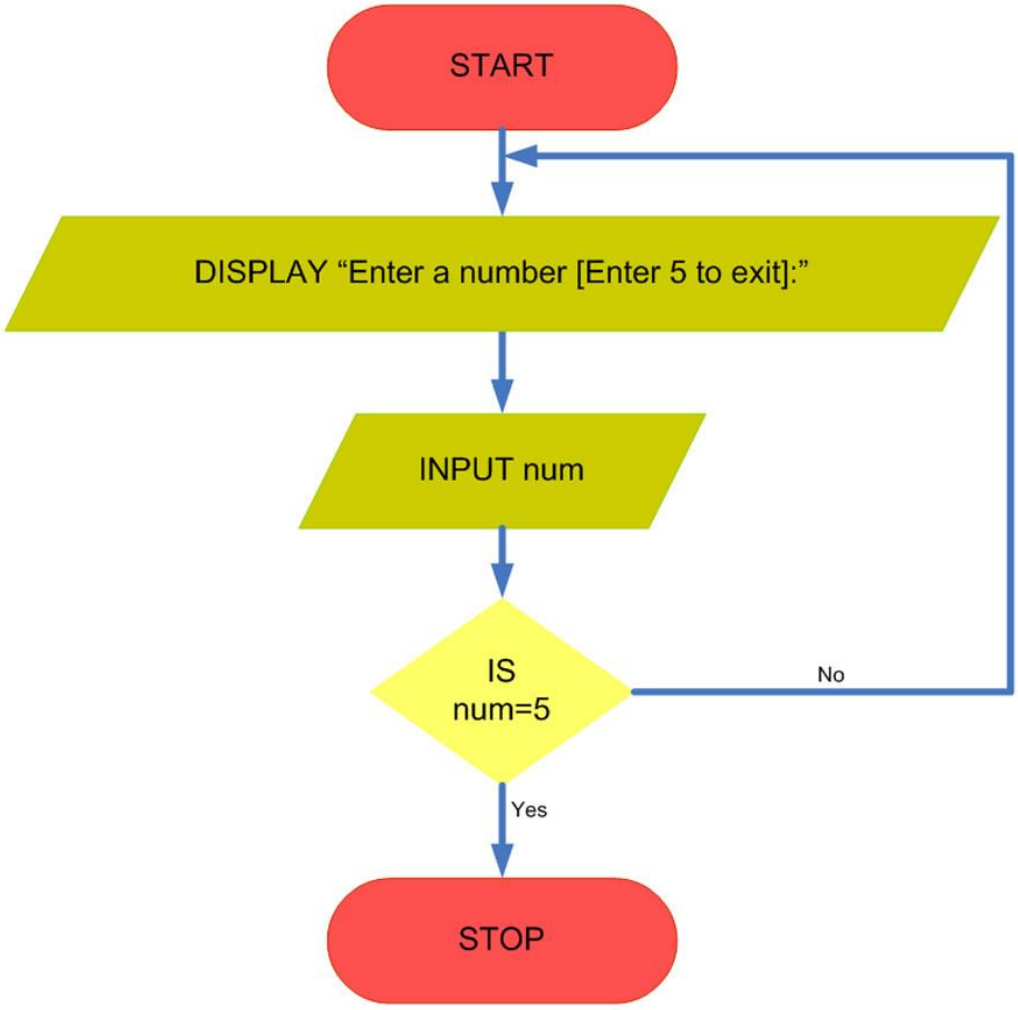


Figure 6.3: Flowchart for Example 6

Note - The C programming language does not support the **`REPEAT...UNTIL`** construct. However, this loop is present in other programming languages.

6.3.4 The FOR Loop

The **FOR** loop provides a more concise loop control structure. The general form for a **FOR** loop is as follows:

```
FOR counterVariable IN RANGE startValue to endValue [STEP value]
DO
     statement set
END DO
```

The **FOR** loop structure is different from the other loop structures. Counter variables are a must in **FOR** loop. The startValue and endValue denote the initial and final values of the counter. In other words, they denote its range. The STEP clause is optional. If specified, the value will denote the amount by which the counter is incremented or decremented with each pass of the loop. If the STEP clause is not specified then, by default, the counter is incremented by 1.

When the program is executed and the control reaches the **FOR** statement, the value of the counter variable is set to the value specified by startValue. It checks to see if the counter variable value is less than the endValue. If it is not, then it exits the loop, else it executes the statement set. When the program encounters the **END DO** statement, it goes back to the **FOR** statement and increments or decrements the counter variable by the value specified by the STEP clause. It then checks if the counter value is still less than or equal to the endValue. If it is, then it goes through the loop once again, else it exits.

The **FOR** loop is used in scenarios where the number of iterations the loop has to go through is already known.

Consider the earlier example to display a name 1000 times in example 8.

Example 8:

```
BEGIN
FOR cnt IN RANGE 1 to 1000
DO
     DISPLAY "Mike"
END DO
END
```

In the example, the STEP part is not specified. So, the counter variable (cnt) value would be incremented by 1 every time. The loop would display the name Mike a thousand times.

Example 9 shows the use of the STEP clause.

Example 9:

```
BEGIN
FOR num IN RANGE 1 to 100 STEP 2
DO
      DISPLAY num
END DO
END
```

The example will increment the value by two every time a loop is run. Hence, the first time 1 will be displayed. The next value that will be displayed will be 3 followed by 5, 7, 9, and so on.

The C syntax for the **FOR** loop along with its execution path is shown in figure 6.4.

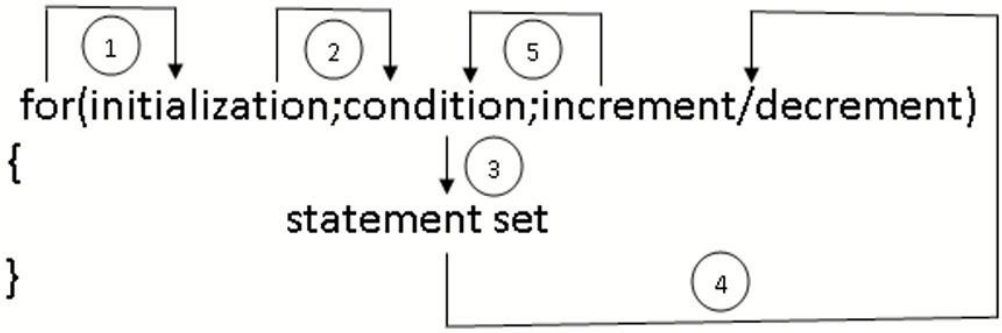


Figure 6.4: FOR Loop with Execution Path

When a program encounters the **FOR** statement, it goes to the initialization section. Here, the program initializes the counter variable. This part is optional and can be left blank if the counter variable is already initialized. However, the semicolon present after this section has to be compulsorily included.

Next, the flow passes on the condition part. This usually consists of a relational expression. This part determines when the loop will exit. If the expression evaluates to True, then the body of the program is executed. Otherwise, the loop exits, and the program execution continues with the statement following the loop.

If the expression in the condition part evaluates to True, then the statement set inside the body of the loop is executed. After the last statement in the loop has been executed, control is transferred back to the **FOR** statement. Next, the control variable is incremented (or decremented if specified) by the value specified in the increment part. The value of the counter variable changes and the condition is checked again. If it evaluates to True, then the program executes the statement set again. This process continues till the value of the counter variable fails to satisfy the test condition.

The **FOR** loop construct for displaying the name Mike a 1000 times in C language is shown in example 10.

Example 10:

```
…
for(cnt=0;cnt<1000;cnt++)
{
     printf("Mike");
}
…
```

6.4 Nested Loops

A nested loop means a loop within another loop. Almost all programming languages use nested loops. The C language allows upto 15 loops to be nested within one another. This number varies for different programming languages.

An example of a nested **WHILE** loop is shown in example 11 where the program accepts five numbers from the user and displays them twice on the screen.

Example 11:

```
BEGIN
cnt = 1
cnt2 = 1
WHILE cnt < 6
INPUT num
     cnt2 = 1
     WHILE cnt2 < 3
     DO
          PRINT num
     cnt2 = cnt2 + 1
     END DO
cnt = cnt + 1
END DO
END
```

The code contains two **WHILE** loops, one nested within another. The first **WHILE** loop specifies the loop to be executed five times. It accepts a number from the user. The second **WHILE** loop displays this number twice before the next number is accepted.

The **FOR** loop helps to manage counter variables in a better way as compared to the **WHILE** loop. The initialization and incrementing takes place in the same loop. The programmer needs to specify the range in the syntax. Hence, a nested **FOR** loop is easier to read and maintain. Also, complex nested loop constructs are easier to create using the **FOR** loop than using any other loop type.

The following example modifies example 12 and uses the **FOR** loop.

Example 12:

```
BEGIN
FOR cnt IN RANGE 1 TO 5
DO
INPUT num
      FOR cnt2 IN RANGE 1 TO 2
      DO
           DISPLAY num
      END DO
END DO
END
```

As seen in the code, the **FOR** loop improves the readability and helps in easier maintenance of the code incase edits and changes are required to be done by the programmer.

6.5 Check Your Progress

1. Which of the following is the function of a loop?

(A)	Specify the standards	(C)	Execute branching conditions
(B)	Manipulate functions	(D)	Perform repetitive tasks

2. The _____ loop repeats a statement or a set of statements while a certain specified condition is True.

(A)	`WHILE`	(C)	`REPEAT...UNTIL`
(B)	`IF...ELSE`	(D)	`DO....NEXT`

3. Which of the following is the general format of the REPEAT...UNTIL loop?

(A)	`REPEAT`
	`UNTIL condition`
	`statement set`
		(C)	`statement set`
			`REPEAT`
			`UNTIL condition`

(B)	`REPEAT`
	`statement set`
	`UNTIL condition`
		(D)	`DO`
			`WHILE`
			`REPEAT`
			`statement set`
			`UNTIL condition`

4. Which of the following is performed in FOR loop after initialization in C language?

(A)	For	(C)	Statement set
(B)	Condition	(D)	Increment

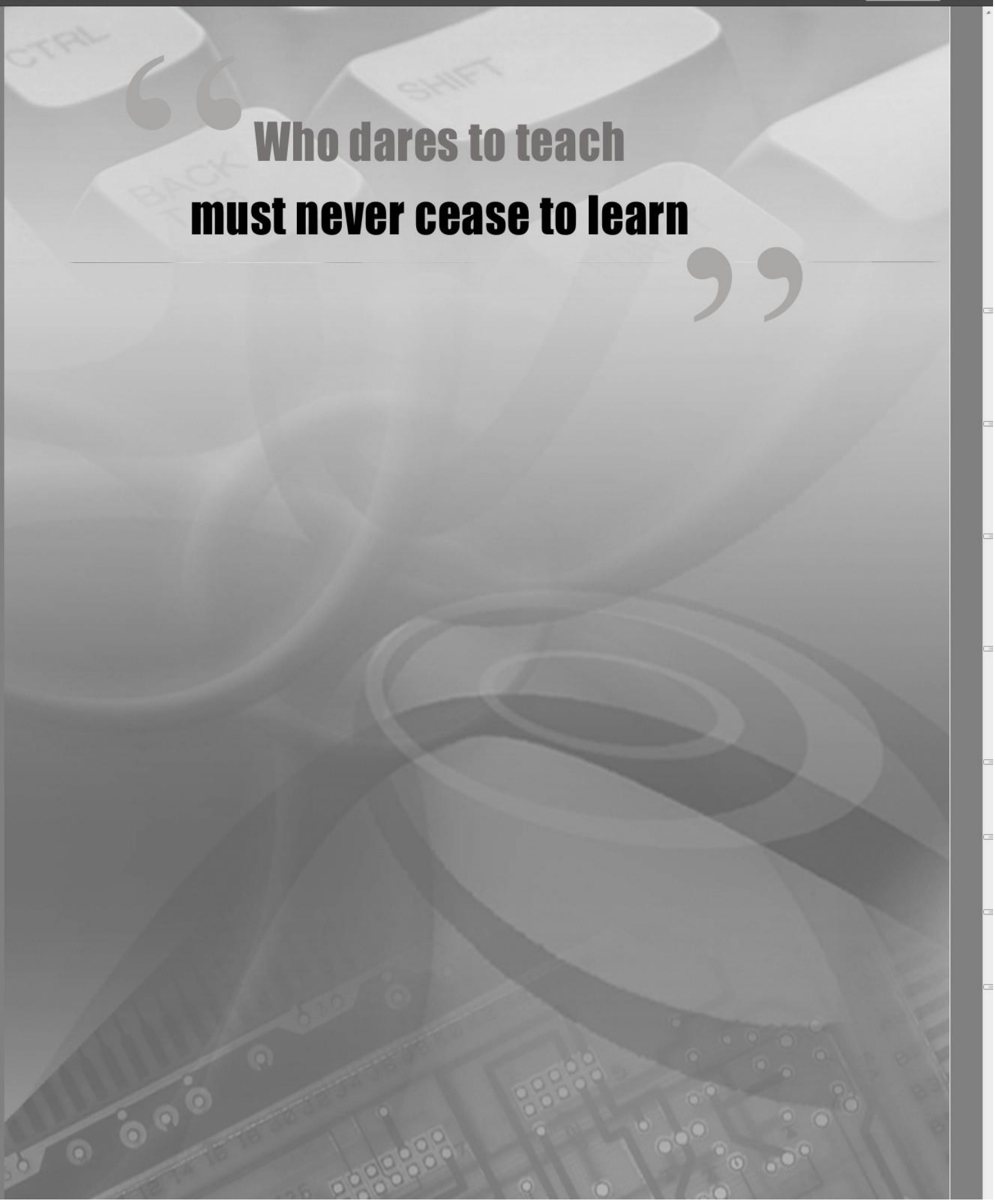
5. The C language allows upto _____ loops to be nested within one another.

(A)	10	(C)	20
(B)	15	(D)	25

6.5.1 Answers

1.	D
2.	A
3.	B
4.	B
5.	B

- The iterative or looping construct is used to repeat certain steps a specific number of times or till some specified condition is met.
- The different types of loops are the **WHILE** loop, the **DO...WHILE** loop, the **REPEAT...UNTIL** loop, and the **FOR** loop.
- The **WHILE** loop repeats a statement or a set of statements while a certain specified condition is True.
- The **DO...WHILE** loop executes the body of the loop before the condition test is performed.
- The **REPEAT...UNTIL** loop executes as long as the condition is False.
- The FOR loop provides a more concise loop control structure, which includes counter variables, range, and step value.
- A nested loop means a loop within another loop.


Who dares to teach
must never cease to learn

Session - 7

Structured Programming

Welcome to the Session, **Structured Programming**.

This session explains structured programming and its benefits. It will also explains top-down structured technique and elements of structured programming.

In this Session, you will learn to:

- Define structured programming
- Describe the benefits of structured programs
- Identify the modeling tool used for structured programming
- Explain the top-down structured technique used in program designing
- Explain the elements of structured programs

7.1 Introduction

Structured programming, also called as modular programming, is a programming approach that applies a logical structure on a program to make it efficient and easier to understand and modify. In the structured programming approach, tasks that are repeatedly performed are defined as functions and written in separate modules or sub-modules. The outcome of this is that code can be loaded into memory more proficiently and that modules can be reused in other programs. After a module has been tested individually, it is then integrated with other modules into the overall program structure.

Structured programming regularly uses a top-down design model, in which developers plan the whole program structure, and then it is divided into subsections.

7.1.1 Benefits of Structured Programming

The advantages of structured programming are described as follows:

- **Code reuse** - Modules (small self-contained blocks) can be used multiple times. This reduces complexity, saves time, and increases reliability. For example, in a program that makes lots of mathematical calculations, a module can be written to calculate a sum of n numbers and the same module can be reused each time with different sets of numbers.

- **Modularity** - This equips programmers to confront problems logically. By splitting a large problem solution into smaller portions, error detection becomes faster and easier.

- **Better flow** - Structured programming promotes better flow as each operation or task is properly segmented or separated. In addition, splitting large complex operations into smaller modules guarantees that the flow of control is understandable.

- **Increased productivity** - Structured programming increases productivity as time taken for error detection and error handling is reduced. Modularity also allows multiple programmers to work on project simultaneously as each can work on a separate module or sub program.

- **Easier debugging and maintenance** - It is also easier to update or fix issues in (debug) the program by replacing individual modules rather than modifying or fixing larger amounts of code.

Consider an example to understand structured programming approach for problem solving. A developer, Peter, has been asked to write a program to instruct a computer to calculate and display the average of two numbers.

To arrive at a solution, the following steps need to be performed. These steps are first broken down in top-down fashion.

Top level:

Display the average of two numbers given as input. Then, the input will be broken down into detailed steps in the next level.

Next level:

1. Accept two numbers as input through the keyboard
2. Calculate average of these two numbers
3. Display the average on the screen

Consider another example where Peter is planning to create a program to manage a list of student names and addresses. The program is divided into four main tasks as follows:

1. Enter new names and addresses.
2. Modify existing entries.
3. Sort entries by last name.
4. Print mailing labels.

Each of these tasks can be assigned to a function. These tasks can still be divided into smaller tasks called subtasks. For example, the first task 'Enter new names and addresses' can be divided into the following subtasks:

1. Read the existing address list from disk.
2. Prompt the user for one or more new entries.
3. Add the new data to the list.
4. Save the updated list to disk.

The 'Modify existing entries' task can be divided into the following subtasks:

1. Read the existing address list from disk.
2. Modify one or more entries.
3. Save the updated list to disk.

In these two sets, there are two common subtasks, reading from disk and saving to disk. A single function can be assigned for the 'Enter new names and addresses', 'Modify existing entries' subtasks. There will be no redundancy or recurrence and the functions created and tested can be reused later by programs requiring the same tasks.

Structured programming method results in a hierarchical or layered program structure, as shown in figure 7.1.

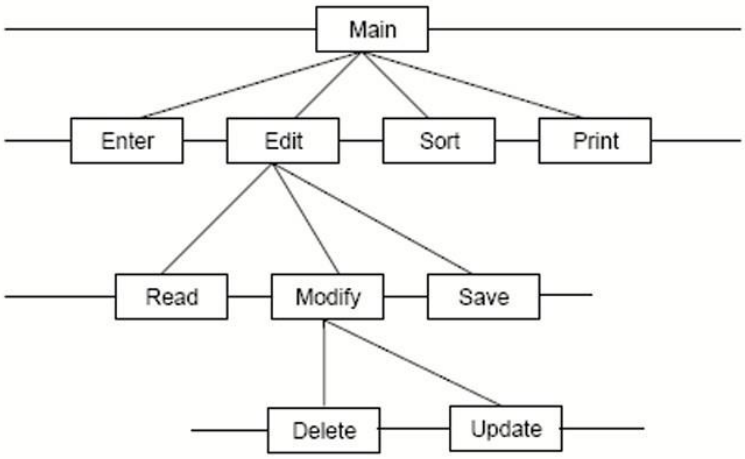


Figure 7.1: Structured Programming Method

7.2 Unified Modeling Language (UML)

Though flowcharts are used to represent structured program logic in a graphical manner, for complex systems, a more formal modeling tool is required.

UML is a popular, standardized modeling tool used in structured programming and object-oriented programming systems. UML is used to specify, visualize, modify, construct, and document the requirements and specifications of a software system under development. In simple terms, UML is used as a tool to 'model' the system.

UML helps designers and developers to read and circulate system structures and design plans and collaborate on application design. UML should be used at the beginning of the software development process when requirements are being collected, reviewed, and evaluated. Traditionally, this used to be done as an informal activity involving discussions within the teams. However, as formal documentation or visualization existed, it would lead to a lot of confusion and uncertainty. UML provides the formalization and visualization, which make the requirements clear and concise.

Figure 7.2 shows basic UML legends that are used in different kinds of UML diagrams.

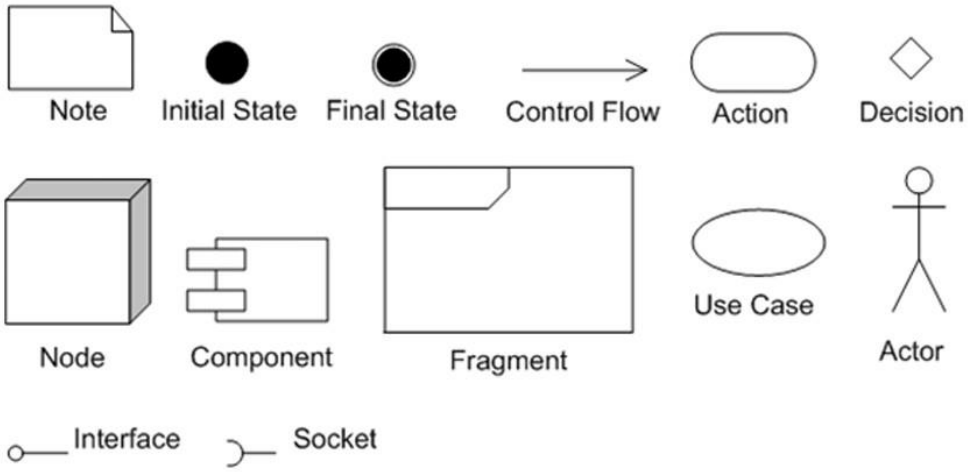


Figure 7.2: UML Legends

UML diagrams are broadly divided into two categories namely, structural and behavioral. Diagrams belonging to each category are as follows:

➔ **Structural Diagrams**:

- Class diagram
- Component diagram
- Composite structure diagram
- Deployment diagram
- Object diagram
- Package diagram

➔ **Behavioral Diagrams**:

- Use case diagram
- Activity diagram
- Communication diagram
- Interaction overview diagram
- Sequence diagram
- State diagram
- Timing diagram

Some of the commonly used UML diagrams are listed as follows:

- **Use Case Diagrams**: A use case diagram is the simplest diagram used to represent interaction of users with the system. A use case diagram portrays the different types of users of a system and the different ways in which they interact with the system. The user, also called as an actor, can be a human or an external system. The actor can be primary or secondary. A primary actor is one that directly interacts with the system. Thus, a use case diagram describes the functionality provided by a system, the goals of actors represented as use cases as well as dependencies among those use cases.

- **Class Diagrams** - A class diagram in UML is a type of static structure diagram that describes the structure of a system by showing the system's classes, their attributes, operations (or) methods, and the relationships between the classes.

- **Activity Diagrams** - An activity diagram illustrates the dynamic nature of a system by modeling the flow of control from activity to activity. An activity represents an operation on some class in the system that results in a change in the state of the system. Typically, activity diagrams are used to model workflow or business processes and internal operation. If any activity diagram is a special kind of state chart diagram, it uses some of the same modeling conventions.

- **Sequence Diagrams** - UML sequence diagrams are used to represent or model the flow of messages, events, and actions between the objects or components of a system. Time is represented in the vertical direction showing the sequence of interactions of the header elements, which are displayed horizontally at the top of the diagram. Sequence diagrams are used primarily to design, document and validate the architecture, interfaces, and logic of the system by describing the sequence of actions that need to be performed to complete a task or scenario.

  UML sequence diagrams are useful design tools. This is because they provide a dynamic view of the system behavior, which can be difficult to extract from static diagrams or specifications. Although, UML sequence diagrams are typically used to describe object-oriented software systems, they are also extremely useful as system engineering tools. They can help to design system architectures, can be used in business process engineering as process flow diagrams, and as message sequence charts for telecom/wireless system design.

  A simple example of a sequence diagram can depict sequence of steps in an online book-shopping site. These steps can include a customer searching a book catalogue, viewing description of a selected book, adding book to shopping cart, and performing checkout.

- **Collaboration Diagrams** - UML collaboration diagrams (interaction diagrams) illustrate the relationship and interaction between software objects. They require use cases, system operation contracts, and domain model that already exist. The collaboration diagram illustrates messages being sent between classes and objects (instances). A diagram is created for each system operation that relates to the current development cycle (iteration).

When creating collaboration diagrams, patterns are used to justify relationships. Patterns are best principles for assigning responsibilities to objects and are described further in the section on patterns. There are two main types of patterns used for assigning responsibilities, which are evaluative patterns and driving patterns.

Each system operation initiates a collaboration diagram. Therefore, there is a collaboration diagram for every system operation.

An example of an interaction diagram can depict the interaction and activities in a deposit or withdrawal transaction in an online banking system.

➔ **State Chart Diagrams** – A state chart diagram defines different states of an object during its lifetime. These states are changed by events. Therefore, state chart diagrams are useful to model reactive systems. Reactive systems can be defined as a system that responds to external or internal events. State chart diagram describes the flow of control from one state to another state. States are defined as a condition in which an object exists and it changes when some event is triggered. Hence, the most important purpose of a state chart diagram is to model the lifetime of an object from creation to termination.

State chart diagrams are also used for forward and reverse engineering of a system. However, they are more commonly used to model reactive systems.

The main purposes of state chart diagrams are as follows:

- To model dynamic aspect of a system
- To model the lifetime of a reactive system
- To describe different states of an object during its lifetime
- Define a state machine to model states of an object

➔ **Component Diagrams** - Component diagrams are different from the other diagrams explained so far in terms of nature and behavior. Component diagrams are used to model physical aspects of a system. Physical aspects are the elements like executables, libraries, files, and documents, which reside in a node. Hence, component diagrams are used to visualize the organization and relationships among components in a system. These diagrams are also used to make executable systems. Component diagrams can also be described as a static implementation view of a system. Static implementation represents the organization of the components at a particular moment.

Component diagram is a special kind of diagram in UML. The purpose is also different from all other diagrams explored so far. It does not describe the functionality of the system, but it describes the components used to make those functionalities.

Hence, the purpose of the component diagram can be summarized as follows:

- Visualize the components of a system
- Construct executables by using forward and reverse engineering

- Describe the organization and relationships of the components

➔ **Deployment Diagrams** - Deployment diagrams are used to visualize the topology of the physical components or the hardware components of a system where the software components are deployed. Deployment diagrams consist of nodes and their relationships. Component diagrams and deployment diagrams are closely related. Component diagrams are used to describe the components and deployment diagrams shows how they are deployed in hardware.

UML is mainly designed to focus on software artifacts of a system. However, these two diagrams are special diagrams that are used to focus on software components and hardware components. Therefore, most of the UML diagrams are used to handle logical components but deployment diagrams are made to focus on hardware topology of a system. Deployment diagrams are used by the system engineers.

The purpose of deployment diagrams described are as follows:

- Visualize hardware topology of a system
- Describe the hardware components used to deploy software components
- Describe runtime processing nodes

7.2.1 Structure Charts

A structure chart is a graphic representation of the decomposition of a problem. It is a tool to assist in software designing. It is particularly helpful while solving large problems. A structure chart shows the dividing of a problem into sub-problems and illustrates the hierarchical relationships between the various parts. An organization chart of a company is an example of a structure chart.

Figure 7.3 shows an example of an organization chart.

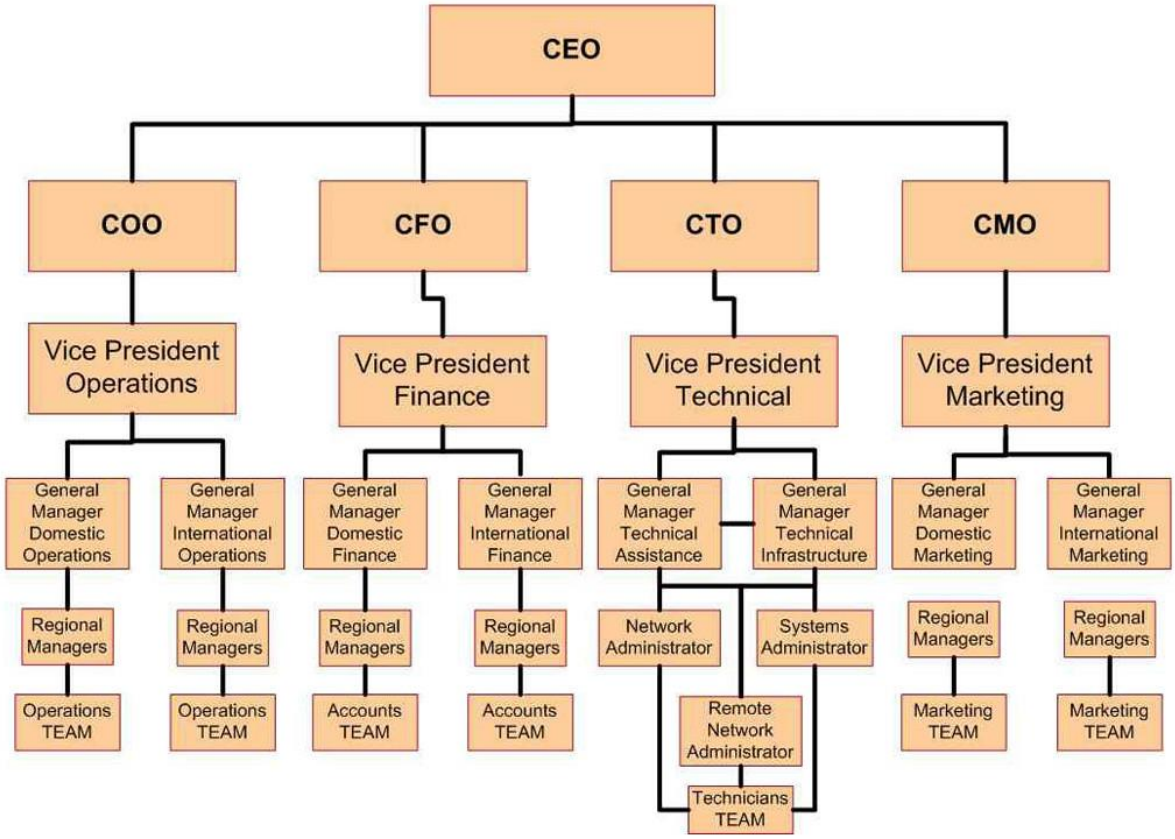


Figure 7.3: Example of Organization Chart

A structure chart is not a flowchart; it has no logical sequence of tasks. It does not depict the order of the tasks performed, yet it illustrates an algorithm.

7.3 Top-Down Structured Technique

Top-down structured technique is the process of dividing the overall task into smaller components. In programming, it signifies breaking a difficult task down and solving pieces independently until every step can easily be implemented.

Figure 7.4 shows a top-down structured technique.

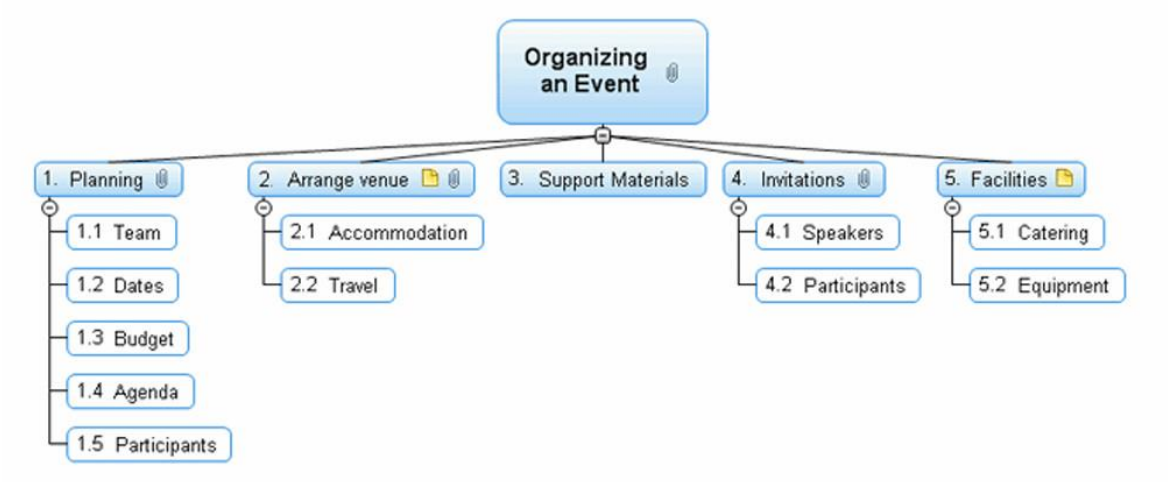


Figure 7.4: Top-Down Structured Technique

Top-down approaches emphasize planning and a complete understanding of the system. Developers implicitly understand that no coding can begin until a sufficient level of detail has been reached in the design of at least some part of the system. Top-down approaches are implemented by attaching the stubs in place of the modules. This, however, delays testing of the ultimate functional units of a system until significant design is complete.

Note - A stub is a piece of dummy code used in a top-down approach.

Thus, the technique for writing a program using top-down methods is to write a main procedure that names all the major functions it will need. Later, the programming team looks at the requirements of each of those functions and the process is repeated. These compartmentalized sub-routines eventually will perform actions so simple that they can be easily and concisely coded. When all the various sub-routines have been coded, the program is ready for testing.

Consider a real-life analogy to understand how top-down structure is viewed. While waiting to land, a plane circles around an airport. Passengers in the plane can see a far-away view and not much detail. They can see green patches and water bodies, but cannot identify the towns or the roads. As the plane descends, passengers can see roads with many vehicles on them. As the plane descends even further, they will be able to add more details such as the human figures, trees, and so on. Finally, as the plane nears landing on the runway, they can see the runway stretch with full clarity. Similar to this, in a top-down design, at the beginning, not much detail is visible but as one goes deeper into the development process, details will begin to take shape.

7.4 Elements of Structured Programs

Five basic elements of programming present in most languages are as follows:

- **Variables** - Variables represent the data. The data can range from something very simple, such as the age of a person, to something very complex. Complex data can include a record of university students, holding their names, addresses, courses taken, and marks obtained. A variable is used to contain the data being used in a program.

- **Loops** - Loops allow carrying out execution of a group of commands a certain number of times.

- **Conditionals** - Conditionals specify execution of a group of statements depending on whether or not some condition is satisfied.

- **Input/output** - This will allow interaction of the program with external entities. This might be as simple as printing something on the screen, or capturing some text the user types on the keyboard, or it can involve reading and/or writing to files.

- **Subroutines and functions** - This will allow putting frequently used snippets of code into one location, which can then be used repeatedly.

7.5 Check Your Progress

1. UML is a popular, standardized modeling tool used in _____ and _____systems.

(A)	Assembly Language Programming	(C)	Structured Programming
(B)	Object-Oriented Programming	(D)	Aspect-Oriented Programming

2. Which of the following is used to model physical aspects of a system?

(A)	Sequence diagram	(C)	Component diagram
(B)	Collaboration diagram	(D)	Class diagram

3. UML collaboration diagrams illustrate the _____ and _____ between software objects.

(A)	support	(C)	efficiency
(B)	relationship	(D)	interaction

4. Which of the following processes divides the overall task into smaller components?

(A)	Top-down structure technique	(C)	Top-up structure technique
(B)	Bottom-up structure technique	(D)	Parallel technique

7.5.1 Answers

1.	B, C
2.	C
3.	B, D
4.	A

- Structured programming also called as modular programming is a programming approach that implements a logical structure on a program to make it competent and easier to comprehend and modify.
- Unified Modeling Language (UML) is a popular modeling language used to specify, visualize, modify, construct, and document the requirements and specification of an object-oriented software system, which is under development.
- A structure chart is a graphical representation of the decomposition of a problem. It is a tool to assist in software designing.
- Top-down structured technique is the process of dividing the overall task into smaller components.

Session - 8

Arrays

Welcome to the Session, **Arrays**.

This session will discuss about arrays and the different types of search and sort techniques that are used for searching an element or ordering the elements in an array.

In this Session, you will learn to:

- Define arrays
- List the different type of arrays
- Describe the different search techniques
- Describe the different sort techniques

8.1 Introduction

In programming language, there are objects which are of same type and size. This organized collection of objects into rows and columns is known as an array. It can be defined as a group of variables of the same data type grouped together under a single name.

8.2 Overview of Array

Arrays are an essential part of programming, as they allow the programmer to store more than one value in a variable, at the same time retaining a single reference. A variable is used to store a piece of information in memory. If several such pieces of information are to be stored, it is hard to identify a variable name for each of these. For example, to store mathematics marks for 100 students in a school, 100 unique names for the variables needs to be identified, or name such as MathsMarks1, MathsMarks2, and so on will be used.

In addition to this, the OS would allocate memory space to these variables at random location. This will not allow the variables to be placed adjacent to each other in memory. Therefore, when performing any operation on these variables, the OS would need to gather information from all these scattered variables. This will be a time consuming process.

A solution to such a problem is arrays. Thus, it can be said that the values in an array are stored in contiguous location in memory and can be accessed by a single name. However, an array consists of several variables, which are grouped together, and accessed using the same name. Each of these variables or objects is of same size and data type. Each of these objects is known as array elements. Therefore, with relation to the example stated, instead of having 100 variables with different names, an array with 100 elements needs to be declared.

One constraint with an array in most languages is that they store only data pertaining to one particular data type. While declaring an array, the data type of the data that is stored needs to be specified and all the elements in it have to be of the same type.

For example, create a program, which accepts five numbers and displays their total. There can be two methods to proceed. In the first method as shown in example 1, declare five variables to store five numbers, and one variable to store the total.

Example 1

```
BEGIN
 DECLARE num1 , num2 , num3 , num4 , num5 and sum as integers
 ACCEPT num1
 ACCEPT num2
 ACCEPT num3
 ACCEPT num4
 ACCEPT num5
 Sum=num1+num2+num3+num4+num5
 Display sum
End
```

In the second method, an array is used as shown in example 2. In this method, use one variable to accept values from the user and another variable, which will accumulate each value input by the user and store a running total.

Example 2

```
BEGIN
Declare num, sum and count as integers
Sum = 0
FOR I IN RANGE 0 TO 5
DO
ACCEPT num
Sum=sum + num
END DO
DISPLAY sum
END
```

In example 1, the programmer declares five integer variable, to accept five numbers. So, if the program is required to be modified to add 50 numbers, then

50 variables need to be declared. In addition to this, the code to accept the 50 numbers would also get repeated, thus resulting in large and inefficient code.

In example 2, though the code to accept the numbers is not large, but the disadvantage is that each value entered by the user is lost as soon as next value is accepted from the user.

The method that can be used for solving the same problem, without losing the input values, would be by using arrays. The pseudocode in example 3, stores three numbers into an array, and displays their total.

Example 3

```
BEGIN
ARRAY arrNums [3] is an integer
arrNums [0] = 2
arrNums [1] = 4
arrNums [2] = 7
total=arrNums [0] +arrNums [1] +arrNums[2]
DISPLAY total
END
```

Some of the advantages of using an array are as follows:

- ➔ Reduction in the number of variable names
- ➔ Selection of the variable based on the value of the variable
- ➔ Storage of the entire data sets for multiple time use in a program
- ➔ Declaration of fixed length data set
- ➔ Access the data in any order or at random

8.2.1 Declaring an Array

In the example, an array is declared with the statement, **`ARRAY arrNums[3]`** is an integer. **`ARRAY`** is the keyword used in an algorithm to declare an array. **`arrNums`** is the name of the array, while [3] indicates the size of the array. In other words, it signifies that the array will have 3 elements, each of which will store an integer value. This is similar to declaring 3 variable of type integer, except that here they can be accessed by the same name **`arrNums`**. Moreover, the elements in the array will be placed adjacent to each other in the memory.

Figure 8.1 shows declaration of variables occupying scattered location in the memory whereas the variables in an array occupy contiguous memory location.

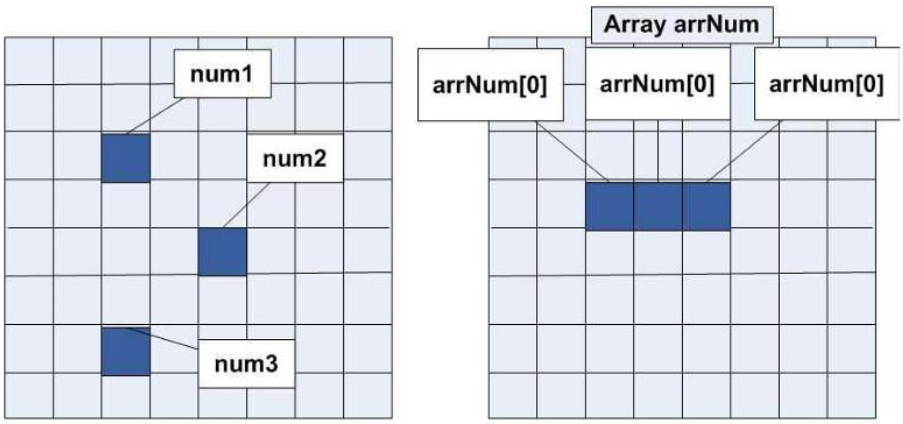


Figure 8.1: Variables Scattered and Variable in an Array

The different elements in the array can be accessed using the array name followed by the element number enclosed within square brackets.

Syntax:

```
arrNum [element number] = value
```

Thus, any value in an array can be accessed using the statement `DISPLAY arrNum[element number]`, which displays the value assigned to the specific element identified by the element number. Thus, it can be said that an array element represents a value in an array. If first element in an array is accessed as arrNum[0], then the number in bracket is called the array index or subscript. Array index or subscript is an integer indicating the position of the element in the array. Array index or subscripts can be expressions or constants. Arrays generally start with index zero, so the first element of an array has the index set to zero. For example, consider the last element of an array as arrNum[2]. In this case, zero is the lower bound of the array, while two is the upper bound as they indicate the lower and the upper boundaries of the array respectively. Any attempt to access any elements outside these bounds would result in an 'Array Out of Bounds' error or runtime error that will abort the program until an error processing code is written that will trap and respond to the error. The number of element in an array determines the array size.

8.3 Different Type of Arrays

As discussed, an array is a universal term for a group of similar quantities. These similar quantities can be marks of 100 students, number of curtains in home, or salaries of 200 employees or ages of 20 students. Thus, an array is a collection of similar elements. These similar elements can be all integers or floats or characters. Typically, the array of characters is called a 'string', where as an array of integers or floats is called an array. All elements of any array must be of the same data type and therefore, there cannot be an array consisting of elements of different data type such as a mix of integers and floats values.

Arrays can be divided into two categories and they are as follows:

- Single dimensional (one-dimensional) arrays
- Multi dimensional arrays

8.3.1 Single Dimensional Arrays

Single dimensional arrays are the simplest form of arrays. It is a type of linear array. All items in a one-dimensional array are stored in a row starting from zero to the size of array. To access an element in a one-dimensional array, a single subscript is used which can either represent a row or column index. In other words, a one-dimensional array consists of a single list accessed by a single index number.

The different components of one-dimensional array are as follows:

- A name
- A data type
- A size

Note - Size determines the number of elements in an array.

Indices or subscripts must be integers within the range. The smallest and the largest indices or subscripts are referred as the lower bound and the upper bound, respectively.

8.3.2 Multi Dimensional Arrays

A single dimensional array stores data as a single set of values in a linear manner. However, sometimes data has to be stored and manipulated in a tabular format in the form of a matrix. For example, consider the data provided in table 8.1, which represents marks scored by 3 students in 2 different subjects. If the result of the highest scored student in all subjects needs to be realized then, one way is by finding out who got highest in physics and chemistry. Next, in the same case check out if the same name is returned in both the cases. This could be carried out using two single dimensional arrays, one for each subject.

	Physics	Chemistry
John	45	60
Mathew	20	67
Ronald	90	35

Table 8.1: An Example of Data in Table

If average marks in each of the subjects and each student's average marks also need to be calculated then, this would require three single dimensional arrays for each student's marks. Similarly, if several operations are required to be carried out on the data, then these data needs to be stored in the memory. One way would be to have different arrays and another way would be to store each values in different variables. In either case, keeping a track of these variables or arrays would be a tedious process.

The solution to this problem is a multi dimensional array, where array would resemble the structure as shown in table 8.1.

Most languages support multi dimensional arrays, where instead of storing the data in a single dimension, it can be stored in more than one dimension. The number of indices required to specify an element is called the dimension of the array. In a two-dimensional array, two indices are used to access an element of an array. A two-dimensional array resembles a matrix, and has rows and columns. In other words, multi dimensional arrays are implemented as arrays of arrays. Table 8.2 shows a two-dimensional array of the example.

	0	1
0	45	60
1	20	67
2	90	35

Table 8.2: Two-dimensional Array

In table 8.2, subjects are listed in columns and individual student's marks are listed in rows. The syntax demonstrates the declaration of a two-dimensional array.

Syntax:

```
ARRAY arrMarks [element number 1] [element number 2]
```

Note - In case of three-dimensional array, the declaration will be as follows:

```
ARRAY arrMarks [3] [2] [4]
```

Note that two pairs of brackets are used in the declaration and the length of the array is specified within the square brackets. Thus, the syntax creates an array named **arrMarks** with 3 rows and 2 columns. Since, arrays are considered to have a base of 0, the arrays index of the first element would be [0][0]. Therefore, to access the marks of any particular student in any particular subject, the row number as well as the column number is specified in addition to the array name. For example, to assign Physics and Chemistry marks to the first student in the array, the following statement is used:

```
ARRAY arrMarks [0] [0] = 45

ARRAY arrMarks [0] [1] = 60
```

Similarly, values can be assigned to all other elements. However, values need to be entered into a two-dimensional array rather than directly hard coded. Example 4 shows the use of FOR loop for this task.

Example 4

```
FOR i IN RANGE 0 TO 1
DO
 FOR j IN RANGE 0 TO 2
 DO
ACCEPT arrMarks [i][j]
 END DO
END DO
```

8.4 Different Search Techniques

Searching refers to the operation of finding the location of a specific item in a group of items. The different search algorithms are as follows:

- Sequential Search
- Binary Search

8.4.1 Sequential Search

The default way to search for a specific item in data is to compare the item with each element of data one by one. Sequential or linear search is the simplest form of search algorithm. In this technique, search starts at the first element and continues until either the item is found or end of the list is reached. For example, in a phonebook to look for the name 'john'. Open to the first page of the phone book and look for the first name and check if it is 'john'. If not then, check the next name whether it is 'john'. If not then continue checking the next names until the name 'john' is found. The algorithm for a sequential or linear search is as follows:

```
INPUT: Array of Size N. Target Value T
OUTPUT: Position of T in the list-1
BEGIN
Set FOUND = false
Set I := 0
While (I <= N) and (FOUND is false)
 IF List[i] == T THEN
 FOUND = true
ELSE
 I = I+1
 END
```

```
IF FOUND==false THEN
  T is not present in the List
END
```

8.4.2 Binary Search

Binary search is the best search algorithm for a sorted array. A binary search is an algorithm for locating the position of an item in a sorted array. It is a powerful technique for searching an ordered list. The concept is similar to the way people look for an entry in a dictionary or telephone book. Generally, people do not start from starting page and read through all the pages to search all entries. Rather, people turn to a particular page in the book where they expect the item to be located. If lucky they will find the item without delay. If not then, they will repeat the process from that specific part of the book. In other words, the algorithm consists of finding the middle element of the list, comparing it with search value, deciding which half of the list to search, and continuing the steps with that half of the list.

In binary search the data set is split into half and the middle item value is compared with the search value. If the search value is smaller than the middle item then the first half of the data set or list is searched. This continues until the search value is located or the remaining list consists of only one item. This is better than sequential search, where each unsuccessful comparison eliminates just one item. The algorithm for a binary search is as follows:

```
BOTTOM=first element
   TOP = last element
   WHILE ((TOP>=BOTTOM) and (not found)) loop
    MID= (TOP + BOTTOM) /2
    IF (LIST (MID) = item to find) THEN
     FOUND = true
    ELSE IF (item to find > LIST (MID) then
     BOTTOM=MID+1
    ELSE
     TOP = MID - 1
    END IF
   END loop
   IF FOUND = true
    Wanted item is in database
   ELSE
    Wanted item is NOT in database
   END IF
```

The complexity is measured by the number of comparisons to locate the item in the data where data contains n elements. Observe that each comparison reduces the sample size into half.

The binary search algorithm requires two conditions and they are as follows:

1. The list must be sorted.
2. One must have direct access to the middle element in any sublist.

8.5 Different Sort Techniques

The function of sorting or ordering a list of objects according to some linear order is very fundamental. Sorting algorithm arranges the elements of a list in a sorted order. It is present in engineering applications in all disciplines. The two types of sorting methods are as follows:

- ➔ Internal sort
- ➔ External sort

8.6 Internal Sort

Internal sorting takes place in the main memory of the computer. This is possible when the data collection to be sorted is small and can be accommodated in the main memory. The programmer benefits in this type of sorting because of the random access nature of the main memory.

The different types of internal sorts are as follows:

- ➔ Selection sort
- ➔ Quick sort
- ➔ Bubble sort
- ➔ Insertion sort

8.6.1 Selection Sort

Selection sort is a very simple sorting algorithm. It works by following these steps:

1. Find the minimum value in the list by iterating over the whole list
2. Swap this value with the first value in the list
3. Repeat these two steps, but each time decrease the list by starting with the second position

By following these steps, the list is divided into two parts: a sorted part, the part to the left, and an unsorted part, the part to the right. With each iteration, the sorted part grows. In other words, the programmer aims to find the smallest element in the unsorted part of the array and swap the value with the first element present in the unsorted part. Figure 8.2 shows an example of selection sort.

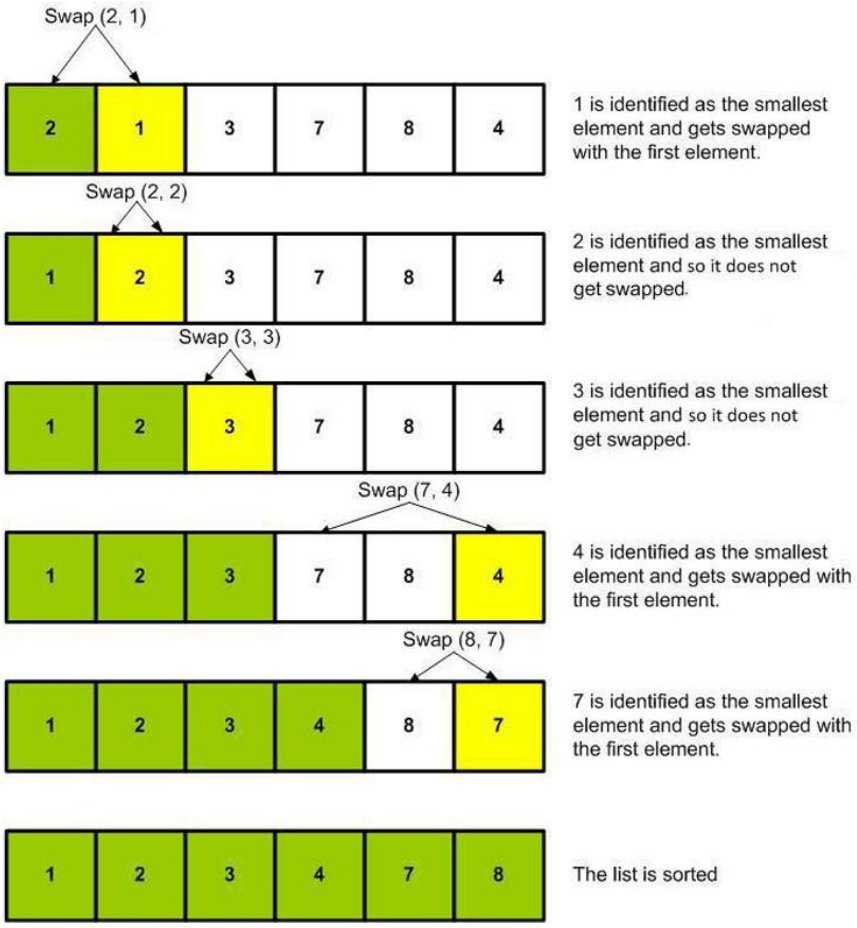


Figure 8.2: An Example of Selection Sort

8.6.2 Quick Sort

Quick sort divides the lists into two sub lists, and then sorts the sub lists. The algorithm for quick sort are as follows:

1. Pick an element from the list. This element is called the pivot value.
2. Reorder the list so that all elements smaller than the pivot value are positioned before the pivot value, and all the larger elements after the pivot value. This operation is called the partitioning.

3. Consider the two newly created partitions, place the inner lists before and after the pivot value, as separate lists. Recursively sort these with quick sort, which means beginning at step one again.

When selecting the pivot value in step one, it is important to remember that the optimal performance is achieved when selecting a value as close to the mean as possible. It is of course not possible to search after the perfect pivot value since this would severely damage the algorithms speed. A simple solution is to select the pivot value at random. Another solution is to choose a mean out of three elements in the list or even perhaps randomly mix the entire list before sorting it, making it possible to always select the first value as the pivot value.

8.6.3 Bubble Sort

Bubble sort is another simple algorithm. It works by repeatedly iterating through a list, comparing two adjacent elements at a time and swapping them wherever necessary. If an iteration through the list makes no further swaps, then the sorting is considered to be complete.

Figure 8.3 shows an example of bubble sort.

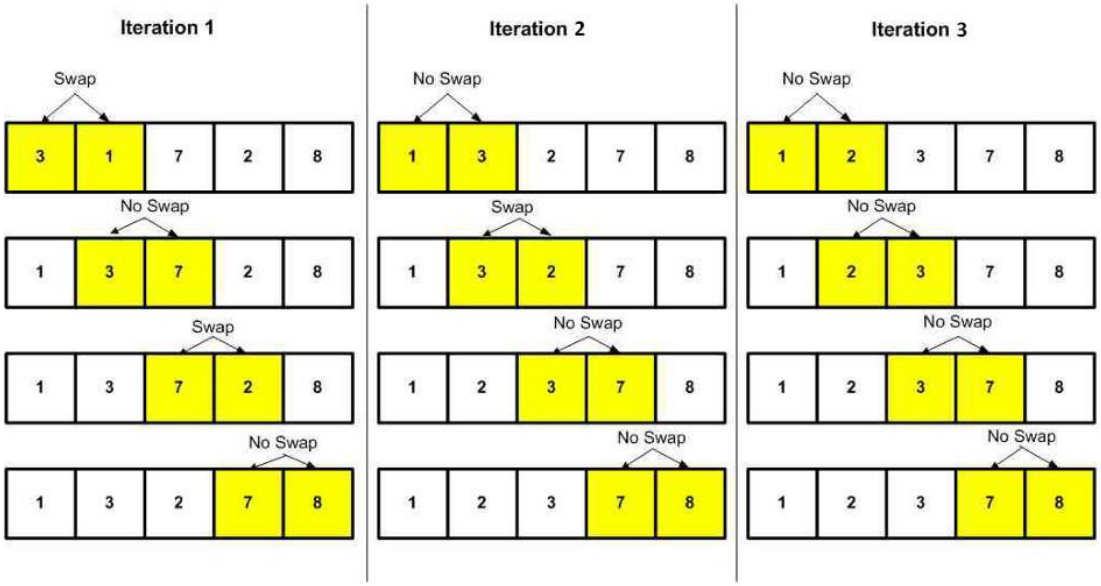


Figure 8.3: An Example of Bubble Sort

The algorithm for a bubble sort is as follows:

```
Procedure bubbleSort( A : list of sortable items )
 REPEAT
  SWAPPED = false
  FOR i = 1 to length(B) - 1 inclusive do:
   IF B[i-1] > B[i] THEN
```

```
    SWAP(B[i-1], A[i])
    SWAPPED = true
   END IF
  END FOR
 UNTIL not SWAPPED
END Procedure
```

8.6.4 Insertion Sort

Insertion sort is a simple sorting algorithm that builds the final sorted array (or list) one item at a time. It is less efficient when the list is large. Characteristics of insertion sort are as follows:

- ➔ Simple to implement
- ➔ Efficient for small data sets
- ➔ Efficient for data sets that are already considerably sorted
- ➔ More efficient in practice than most other algorithms such as selection sort or bubble sort
- ➔ It is stable and does not change the relative order of elements with equal keys
- ➔ It can sort a list as it receives the list

When humans manually sort something (for example, a deck of playing cards), most of them use a method that is similar to insertion sort.

Figure 8.4 shows the operation of insertion sort on an array named A and initialized with the values 5, 2, 4, 1, and 3. Each part shows what happens for a particular iteration.

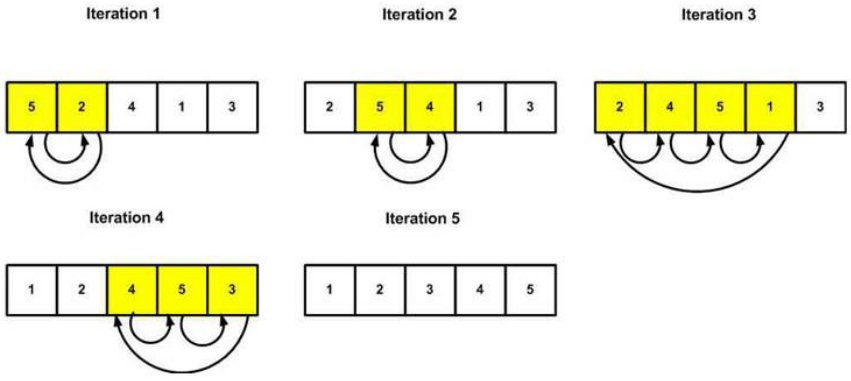


Figure 8.4: An Example of Insertion Sort

8.7 External Sort

External sorting is necessary when the number and size of objects are large and cannot be accommodated in the main memory. This technique is used when the data to be sorted is large and needs to be stored in an external storage such as hard disks. These methods involve as much external processing as processing in the CPU. To study the external sorting, study the various external devices used for storage in addition to sorting algorithms. This sorting requires auxiliary storage.

The examples of external sorting are as follows:

- Sorting with Disk
- Sorting with Tapes

8.8 Check Your Progress

1. _____ is a word used in an algorithm to declare an array.

(A)	ArrNums	(C)	arrNum
(B)	ARRAY	(D)	array

2. Which is the default way to search for a specific item in data?

(A)	To arrange the data according to the item	(C)	To compare the item with each element of data one by one
(B)	To add the item with each element	(D)	To arrange each element of data one by one

3. Which of the following options arranges the following steps for selection sort in correct sequence?

(A)	Find the minimum value in the list by iterating over the whole list
(B)	Repeat these two steps, but each time decrease the list by starting with the second position
(C)	Swap this value with the first value in the list

(A)	1, 2, 3	(C)	1, 3, 2
(B)	3, 2, 1	(D)	3, 1, 2

4. External sorting is necessary when the _____ and _____ are large to be accommodated in the main memory.

(A)	strength	(C)	force
(B)	number	(D)	size of objects

5. Which of the following sorting algorithm repeatedly iterates through a list, compares the two adjacent elements at a time and swap them wherever necessary?

(A)	Selection sort	(C)	Bubble sort
(B)	Quick sort	(D)	Insertion sort

8.8.1 Answers

1.	B
2.	C
3.	C
4.	B, D
5.	C

- Arrays are an essential part of programming, as they allow the programmer to store more than one value in a variable, at the same time retaining a single reference.
- Array can be defined as a collection of elements of same type that are referenced by a common name.
- All items in a one-dimensional array are stored either in a row or column and indexing starts from zero and ends with the size of the array minus one.
- Most languages support multi dimensional arrays, where instead of storing the data in a single dimension, it can be stored in more than one dimension.
- Searching refers to the operation of finding the location of a specific item in a group of items. The different search algorithms are as follows:
  - Sequential Search
  - Binary Search
  - Binary Tree Search
- Internal sorting takes place in the main memory when the data to be sorted is small. The different types of internal sorts are as follows:
  - Selection sort
  - Quick sort
  - Bubble sort
  - Insertion sort
- External sorting is necessary when the number and size of objects are large and cannot be accommodated in the main memory.

Session - 9

File Handling

Welcome to the Session, **File Handling**.

The session will explain file organization and different types of files. It also explains the different file operations and control breaks used in programming language.

In this Session, you will learn to:

- Explain file organization
- Describe different types of files
- Explain the different file operations

9.1 Introduction

Files are used to store information permanently and retrieve it when it needs to be processed by specific programs. A file is a collection of correlated data or collection of bytes stored on a secondary storage device, such as hard disk drive, a pen drive, and so on. The collection of bytes can be interpreted as words, characters, lines, paragraphs and pages from a textual document, fields and records belonging to a database, or pixels from a graphical image. The significance of a particular file is determined by the data structures and operations used by a program to process that file. It is feasible that a graphics file will be read and displayed by a program designed to process textual data. There will be no meaningful output. A file is simply a machine readable storage media where programs and data are stored for machine usage.

There are two kinds of files and they are as follows:

- Text files
- Binary files

9.1.1 Text Files

A text file is a stream of characters that a computer can process sequentially. It processes sequentially in forward direction. So at specific time, a text file is typically opened only for a single operation such as reading, writing, or appending. Figure 9.1 shows a text file.

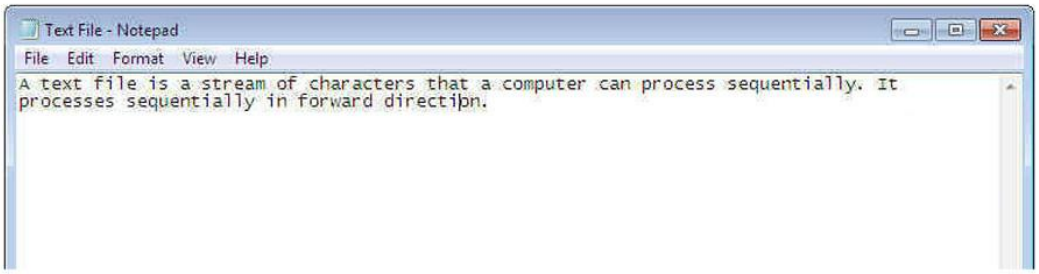


Figure 9.1: Text File

Similarly, since text files can only process characters, so they only read or write data one character at a time. In C Programming, functions are provided that deal with lines of text, but still these process data one character at a time. Depending on whether data of a file is being written or read, the newline character can be converted to linefeed combinations and vice versa. Other character conversions can also occur as per the storage requirements of the operating system. These translations occur transparently. They occur because the programmer has stated the objective to process a text file.

Concepts

9.1.2 Binary Files

A binary file is a collection of bytes. No special processing of the data occurs and each byte of data is transferred to or from the disk unprocessed. A programming language does not place constructs on the file, and it is read, or written, in a method selected by the programmer. Figure 9.2 shows a binary file.

0000	FF	D8	FF	E1	1D	FE	45	78	69	66	00	00	49	49	2A	00
0010	08	00	00	00	09	00	0F	01	02	00	06	00	00	00	7A	00
0020	00	00	10	01	02	00	14	00	00	00	80	00	00	00	12	01
0030	03	00	01	00	00	00	01	00	00	00	1A	01	05	00	01	00
0040	00	00	A0	00	00	00	1B	01	05	00	01	00	00	00	A8	00
0050	00	00	28	01	03	00	01	00	00	00	02	00	00	00	32	01
0060	02	00	14	00	00	00	B0	00	00	00	13	02	03	00	01	00
0070	00	00	01	00	00	00	69	87	04	00	01	00	00	00	C4	00
0080	00	00	3A	06	00	00	43	61	6E	6F	6E	00	43	61	6E	6F
0090	6E	20	50	6F	77	65	72	53	68	6F	74	20	41	36	30	00
00A0	00	00	00	00	00	00	00	00	00	00	00	00	B4	00	00	00
00B0	01	00	00	00	B4	00	00	00	01	00	00	00	32	30	30	34
00C0	3A	30	36	3A	32	35	20	31	32	3A	33	30	3A	32	35	00
00D0	1F	00	9A	82	05	00	01	00	00	00	86	03	00	00	9D	82
00E0	05	00	01	00	00	00	8E	03	00	00	00	90	07	00	04	00

Figure 9.2: Binary File

Binary files are either processed sequentially or depending on the needs of the application. They can be processed using random access techniques. Binary files are generally processed using read and write operations simultaneously. For this process, the files are moved from their current position to an appropriate place in the file. This process is carried out before reading or writing data on the file.

For example, a database file will be created and processed as a binary file. A record update operation will involve locating the appropriate record, reading the record into memory, modifying it in some way, and finally writing the record back to disk at its appropriate location in the file.

9.2 File Organization

File organization refers to the relationship of the key of the record to the physical location of that record in the computer file. File organization may either be a physical file or logical file. A physical file is a physical unit, such as magnetic tape or a disk. A logical file on the other hand is a complete set of records for a specific application or purpose. A logical file may occupy a part of physical file or may extend over more than one physical file.

The objectives of computer based file organization are as follows:

- Ease of file creation and maintenance
- Efficient means of storing and retrieving information

Concepts

9.3 Types of File Organization Methods

Single file organization is a way of organizing the data or records in a file. It does not refer to how files are organized in folders, but how the contents of a file are added and accessed. There are three types of file organization methods, they differ in how effortlessly records can be accessed and the complexity in which records can be organized.

The various file organization methods are as follows:

- ➔ Sequential access
- ➔ Direct access
- ➔ Indexed sequential access

The selection of a particular method depends on:

1. Type of application
2. Method of processing
3. Size of the file
4. File inquiry capabilities
5. File volatility
6. The response time

9.3.1 Sequential Access

In sequential access searching the records are arranged in the ascending, descending, or chronological order of a key field which can be numeric or both. There is no storage location identified, since the records are ordered by a key field. Sequential access is used in applications like payroll management where each file in the record is processed. Here, to have an access to a particular record, each record must be examined until the desired record is attained.

Sequential files are normally created and stored on magnetic tape using batch processing method. Figure 9.3 shows an example of sequential access.

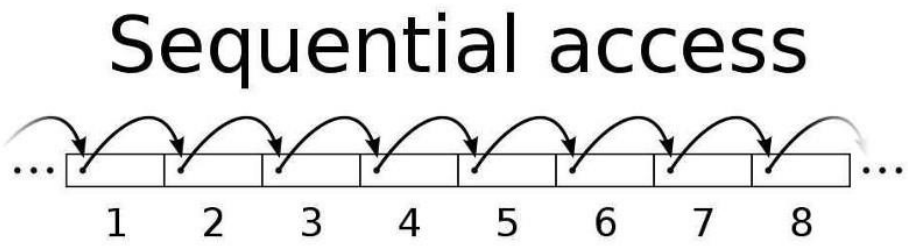


Figure 9.3: Example of Sequential Access

Concepts

Advantages of sequential access are as follows:

- ➔ It is simple to understand.
- ➔ It is easier to maintain and organize.
- ➔ In sequential access loading a record requires only the record key.
- ➔ A relatively inexpensive I/O media and devices can be used.
- ➔ It is easy to reconstruct the files.
- ➔ The proportion of file records to be processed is high.

Disadvantages of sequential access are as follows:

- ➔ To get specific information, the entire file must be processed.
- ➔ It stores very low activity rate.
- ➔ Transactions are required to be placed and stored in a sequence prior to processing.
- ➔ Data redundancy increases, as same data can be stored at different places with different keys.
- ➔ Random enquiries are impossible to handle.

In a sequential file organization, records are organized in the sequence by which they were added. A new record cannot be inserted between existing records, rather it can only be inserted at the end of the last record. Sequential file organization allows to process batches of records in the file without making any changes in the record. However, to access a specific record, all the other records must be processed. This laborious work is done because it does not generate any random key to identify the location of the record. Searching for a record, especially when there are thousands of entries, can be time consuming. Also, inserting or deleting records would imply rearranging the entire sequence.

9.3.2 Direct Access

The files are stored in direct access storage devices such as magnetic disk, using an identifying key. The identifying key connects to its actual storage position in the file. The computer can directly locate the key to identify the specific record without searching through the whole record. In the online system the response and updation are fast.

Concepts

Figure 9.4 shows an example of direct access.

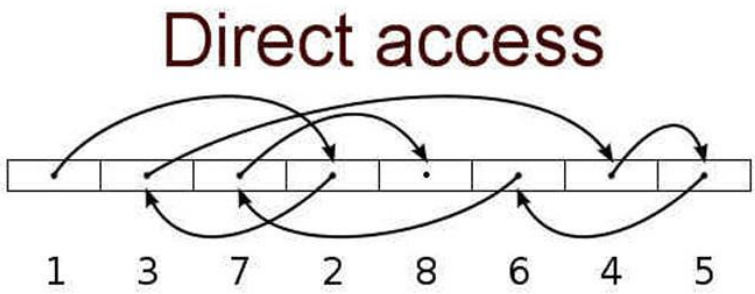


Figure 9.4: Example of Direct Access

Advantages of direct access are as follows:

- Records can be immediately accessed for updation.
- Several files can be simultaneously updated during transaction processing.
- Transaction need not be sorted.
- Existing records can be amended or modified.
- Very easy to handle random enquiries.
- Most suitable for interactive online applications.

Disadvantages of direct access are as follows:

- Data may be accidentally erased or over written unless special precautions are taken.
- Risk of loss of accuracy and breach of security. Special backup and reconstruction procedures must be established.
- Less efficient use of storage space.
- Expensive hardware and software are required.
- High complexity in programming.
- File updation is more difficult when compared to that of sequential method.

Concepts

9.3.3 Indexed Sequential Access

In indexed sequential access, the records are stored sequentially on a direct access device such as magnetic disk and the data is accessible randomly and sequentially. It covers the positive features of both sequential and direct access files. This type of file organization is appropriate for batch processing and online processing. The records in indexed sequential access are organized in sequence for efficient processing of large batch jobs, but an index is also used to speed up access to the records. Indexing permits access to selected records without searching the entire file.

Advantages of indexed sequential access are as follows:

- Permits efficient and economic use of sequential processing technique when the activity rate is high.
- Permits quick access to records, in an efficient way.

Disadvantages of indexed sequential access are as follows:

- Slow retrieval, when compared to other methods.
- Does not use the storage space efficiently.
- Hardware and software used are relatively expensive.

An indexed file organization contains reference numbers, such as employee numbers, that identifies a particular record from the collection of other records. These references are unique for each record and they are called the primary keys. Alternate keys can also be defined to allow alternate methods of accessing the record. For example, instead of accessing an employee's record using employee's number, the user can use an alternate key that refers the employee by departments. This allows greater flexibility for users to randomly search through thousands of records in a file. However, this requires complex programming for implementation.

9.4 Different File Operations

A file is an abstract data type. To define a file, consider the operations that can be performed on files.

Six basic file operations that an OS can provide are as follows:

- **Creating a file** - Two steps are necessary to create a file and they are as follows:
  - Space in the file system must be found for the file.
  - An entry for the new file must be made in the directory.

Concepts

➔ **Writing a file** - To write a file, a system call is made specifying both the name and the information to be written to the file. The system must keep a write pointer to the location in the file where the next write is to take place. The write pointer must be updated whenever a write occurs. Example 1 pseudocode shows a scenario to create employee record file and to write a record into it.

Example 1

```
START
OPEN EMP-FILE FOR OUTPUT
DISPLAY "Enter Employee code:"
ACCEPT EmpCode
DISPLAY "Enter Employee Name:"
ACCEPT EmpName
DISPLAY "Enter Employee Department:"
ACCEPT Dept
DISPLAY "Enter Employee Salary:"
ACCEPT EmpSalary
WRITE EmpCode, EmpName, Dept, EmpSalary
CLOSE EMP-FILE
STOP
```

Figure 9.5 shows a flowchart for creating and writing a file.

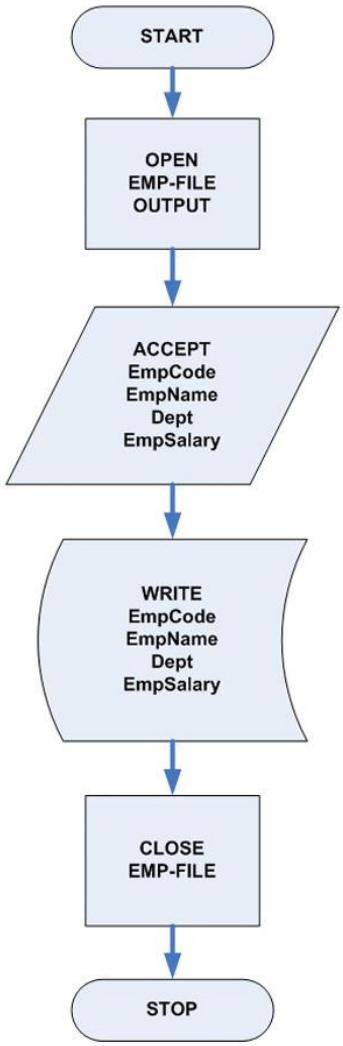


Figure 9.5: Flowchart for Creating and Writing a File

➔ **Reading a file** - To read from a file, a system call is made that specifies the name of the file and where the next block of the file should be placed in memory. The system has to keep a read pointer to the location where the next read is to take place in the file. Example 2 pseudocode shows a scenario to read the employee records from EMP-FILE and Print it.

Concepts

Example 2

```
START
OPEN EMP-FILE FOR INPUT
READ EMP-FILE INTO EmpCode, EmpName, Dept, EmpSalary
WHILE not EOF
DO
READ EMP_FILE INTO EmpCode, EmpName, Dept, EmpSalary
PRINT EmpCode, EmpName, Dept, EmpSalary
END DO
STOP
```

Figure 9.6 shows a flowchart for reading and printing a file.

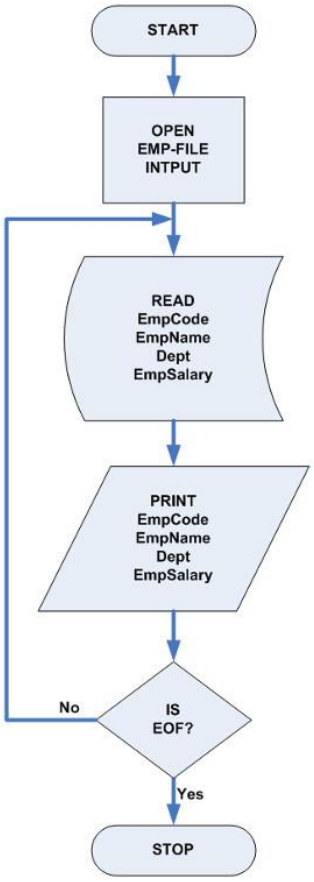


Figure 9.6: Flowchart for Reading and Printing a File

Concepts

➔ **Repositioning within a file** - The directory is searched for the appropriate entry, and the current file position pointer is repositioned to a given value. Repositioning within a file need not involve any actual I/O. This file operation is also known as a file seek.

➔ **Deleting a file** - To delete a file, search the directory for the named file. Having found the associated directory entry, release all file space, so that it can be reused by other files, and erase the directory entry.

➔ **Truncating a file** - The user can erase the contents of a file, but keep its attributes. Rather than forcing the user to delete the file, and then recreate it, this function allows all attributes to remain unchanged. This function reset file to length zero and its file space released.

9.5 Sequential Files and Control Break Logic

Sequential file techniques provide a straightforward way to read and write files. The reason for using sequential files is its easy portability to other programming languages and computers. Therefore, sequential files are used as the common denominator of data processing. They can be read by word-processing programs, adapted to other applications, and transferred over the Internet to other computers.

The disadvantage of sequential files is that it allows to access one line at a time, starting with the first line. Hence, if last line in a sequential file of 23,000 lines is required to be accessed, then preceding 22,999 lines needs to be read. So, sequential files, are specifically suited for applications that perform sequential processing such as, counting words, checking spelling, and printing mailing labels in file order. These type of applications allows to read the file entirely at the start of a program and to rewrite the entire file at the end. In between, the information can be stored in an array which can be accessed randomly.

In a computer program, a control break occurs when there is a change in the value of a single key on which a file is sorted for some added processing. For example, an input file sorted by post code, the number of items identified in each postal district needs to be printed on a report, with a heading displayed for the next district. Usually there is a hierarchy of nested control breaks in a program such as, streets within districts, within areas, the requirement for a grand total at the end. Structured programming techniques are developed to ensure correct processing of control breaks in languages. The fourth generation programming languages such as SQL, processes control breaks automatically.

Control break processing is used when writing report programs manually. Such report designers handle all of this control break process in back-end while the format of the report is visually prepared. Writing report programs manually includes handling all details of the printing and subtotaling logic manually within the program mainframe environments using a programming language.

9.6 Check Your Progress

1. In sequential access searching the records are arranged in the _____, _____, or _____ order of a key field which can be numeric or both.

(A)	ascending	(C)	chronological
(B)	descending	(D)	distributive

2. Which of the following file organization can directly locate the key to identify the specific record without searching through the whole record?

(A)	Direct file organization	(C)	Random file organization
(B)	Sequential file organization	(D)	Indexed sequential file organization

3. Which of the following file organization does not allow to insert file in between the records rather it needs to be added at the end?

(A)	Direct file organization	(C)	Random file organization
(B)	Sequential file organization	(D)	Indexed sequential file organization

4. A _____ is a stream of characters that a computer can process sequentially.

(A)	text file	(C)	data file
(B)	binary file	(D)	sequential file

5. Which of the following are file operations that an OS can provide?

(A)	creating a file	(C)	deleting a file
(B)	reading a file	(D)	attaching a file

9.6.1 Answers

1.	A, B, C
2.	A
3.	B
4.	A
5.	A, B, C

- A file is a collection of correlated data or collection of bytes stored on a secondary storage device, such as hard disk drive, a pen drive, and so on.
- A text file is a stream of characters that a computer can process sequentially.
- A programming language does not place constructs on the file, and it is read, or written, in a method selected by the programmer.
- File organization refers to the relationship of the key of the record to the physical location of that record in the computer file.
- There are three types of file organization methods, they are as follows:
  - Sequential
  - Relative
  - Indexed
- In sequential access searching the records are arranged in the ascending, descending, or chronological order of a key field which can be numeric or both.
- In a computer program, a control break occurs when there is a change in the value of a single key on which a file is sorted for some added processing.

Session - 10

Data Flow Diagrams

Welcome to the Session, **Data Flow Diagrams**.

The session will discuss the different system flowcharts and the use of Data Flow Diagram (DFD). It also explains the usage of decision tables and Hierarchy plus Input-Process-Output (HIPO) chart.

At the end of this session, you will be able to:

- Describe system flowcharts
- Explain DFDs
- Explain decision tables
- Describe HIPO charts

10.1 Introduction

In 1970 DFDs were introduced and popularized for structured analysis and design (Gane and Sarson 1979). DFDs depict the flow of data from external entities into the system. It also shows how the data moves from one process to another.

The DFD is an excellent communication tool for analysts to model processes and functional requirements. It is still considered one of the best modeling techniques for extracting and representing the processing requirements of a system. If utilized effectively, it is an efficient and easy to understand modeling tool.

10.2 System Flowchart

The system flowchart is a way of visually presenting the flow of data through an information processing system, the operations executed within the system, and the order in which they are performed. The flowchart can be similar to a blueprint of an under construction building. Flowcharts are usually drawn in the early stages of developing computer solutions. For example, a building contractor always draws a blueprint before starting construction on a building. Similarly, a programmer chooses to draw a flowchart before writing a computer program. The flowcharts play a vital role in the identification of a problem and are helpful in understanding the logic of difficult and lengthy programs. Hence, a flowchart is helpful for the better documentation of a complex program.

Figure 10.1 shows a sample flowchart diagram.

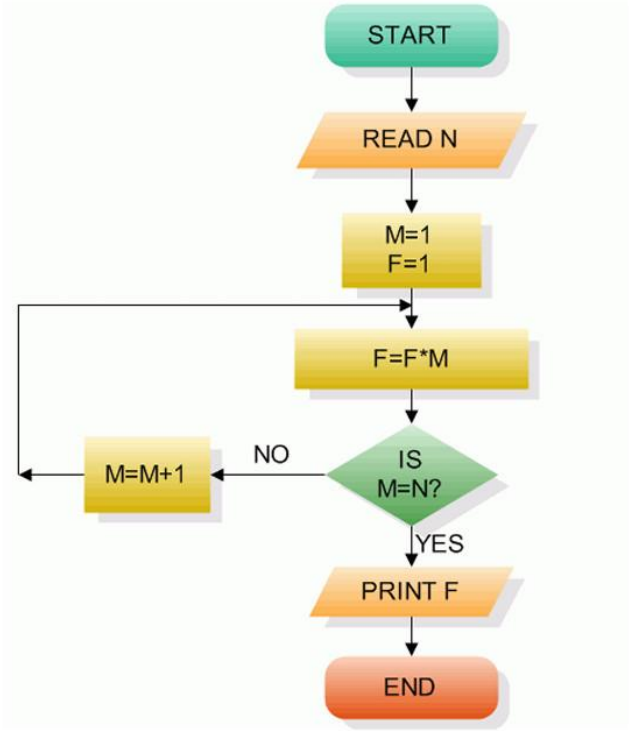


Figure 10.1: Sample Flowchart Diagram

A flowchart clarifies how the program is currently working and how it could be improved. It also helps out in identifying the key elements of a process, while understanding where a process is ending and the next process is starting. Creating a flowchart motivates communication among programmers and establishes a common understanding about the process. Flowcharts also reveals those steps that are redundant or misplaced. Flowcharts are also used to identify appropriate team members, to identify who provides inputs or resources to whom, to create important areas for monitoring or data collection, to identify areas for improvement or increased efficiency, and to generate hypothesis about causes. For example, in a hospital, flowcharts can be used to observe processes for the flow of patients, information, materials, clinical care, or combinations of these processes. It is recommended to create flowcharts through group discussion, as individuals seldom knows the entire process and the discussion contributes to the development.

10.2.1 Types of Flowchart

A flowchart is the graphical representation of how a process works and depicts the sequence of steps. There are three types of flowcharts and they are as follows:

- **High-Level Flowchart** - A high-level also called first-level or top-down flowchart depicts the most important steps in a process. It can also include the intermediate outputs of each steps and sub-steps involved. This type of flowchart offers a basic depiction of the process and identifies the changes taking place within the process. It is considerably useful for distinguishing appropriate team members involved in the process. It is also useful for developing indicators for monitoring the process as it focuses on intermediate outputs.

  Most processes can be effectively portrayed in four or five flowchart symbols that depicts the major steps or activities of the process. It is recommended to use only a few flowchart symbols, as it forces the programmer to consider the most important steps. Other steps are typically sub-steps of the important ones.

Figure 10.2 shows a sample high-level flowchart.

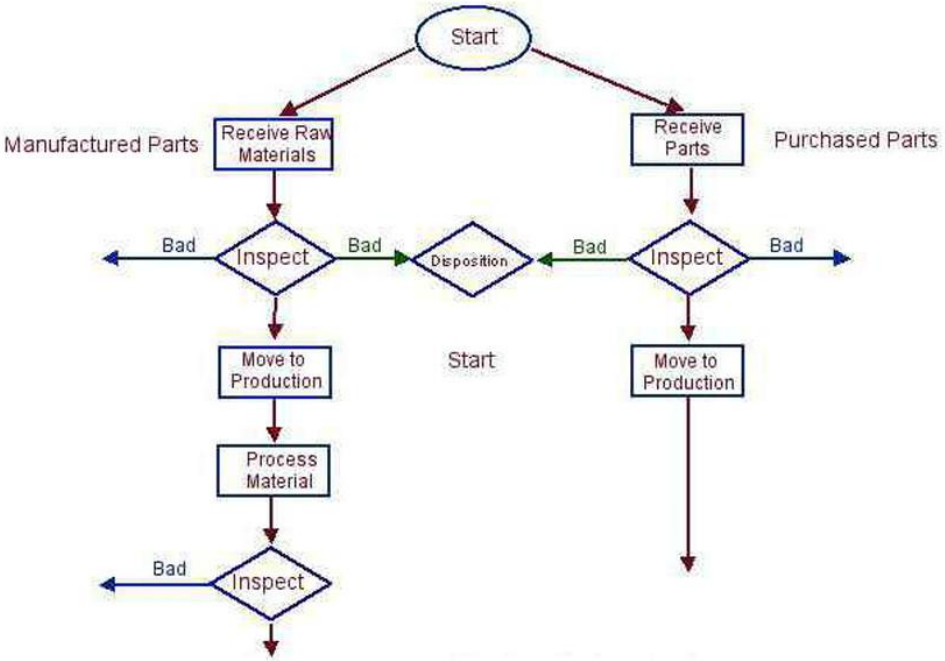


Figure 10.2: High-level Flowchart

➔ **Detailed Flowchart** - The detailed flowchart provides a detailed depiction of a process by mapping all the steps and activities that appear in the process. The detailed flowchart specifies the steps or activities of a process and also comprises decision points, waiting periods, reworks, and feedback loops. This flowchart is useful for examining all areas of the process and also to determine problems or areas of inefficiency. For example, the detailed flowchart of a patient registration in a hospital discloses the delay when the record clerk and clinical officer are not present to assist the patient.

Figure 10.3 shows a sample detailed flowchart.

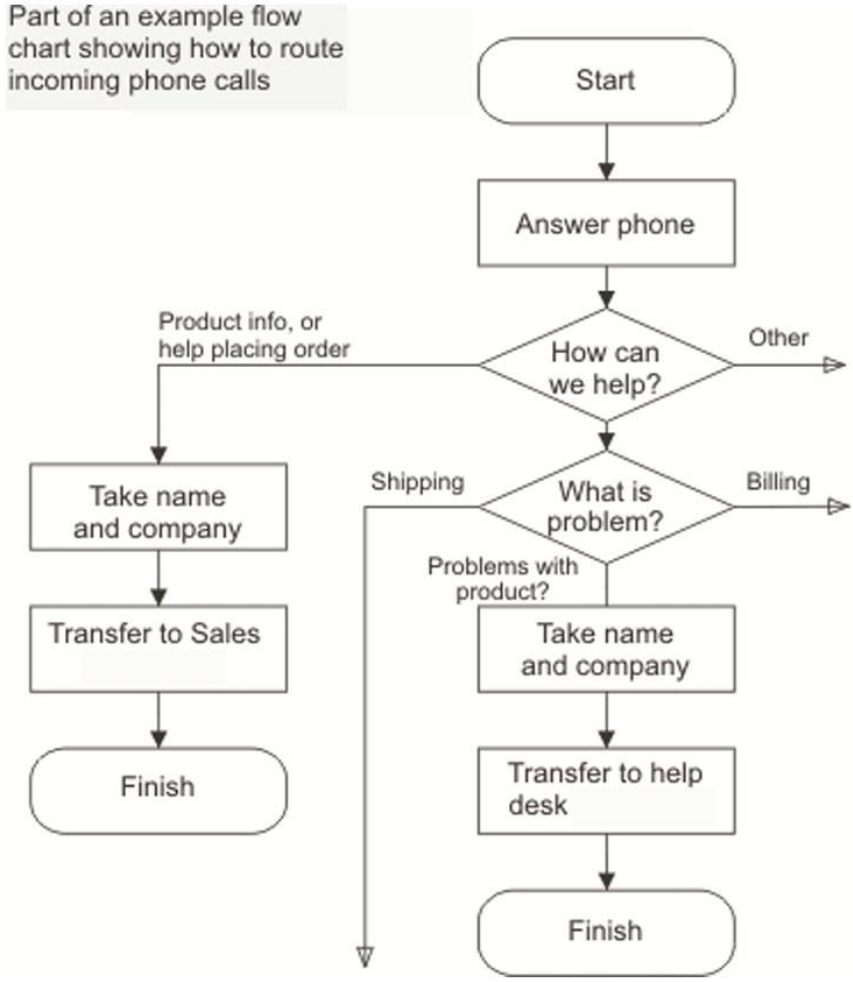


Figure 10.3: Detailed Flowchart

➔ **Deployment or Matrix Flowchart** - A deployment flowchart depicts the process in terms of who is doing the steps. It is in the form of a matrix, illustrating various members in a team and the flow of steps among these members. It is essentially useful in identifying who is providing inputs or services to whom. It is also useful in the disclosure of the areas where programmers are executing the same task.

Figure 10.4 shows a sample deployment or matrix flowchart.

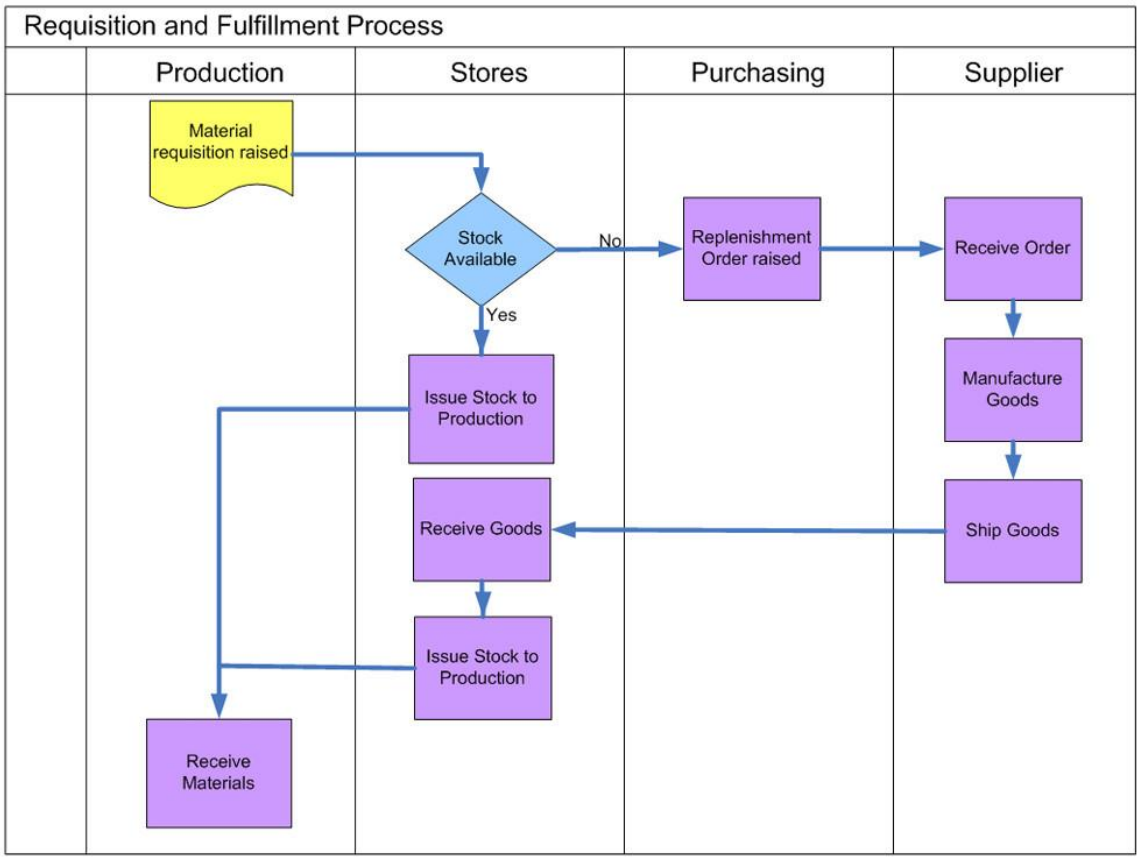


Figure 10.4: Deployment or Matrix Flowchart

Each type of flowchart has its advantages and disadvantages; the high-level flowchart is the easiest to create, but it does not provide sufficient details. The group that decides the flowchart type should be clear on their purpose for flowcharting. If the programmers are uncertain about any flowchart then, start with the high-level and gradually shift to detailed and deployment flowcharts. The detailed and deployment flowcharts are time-consuming.

10.2.2 Flowchart Symbols

It is not strictly mandatory to use boxes, circles, diamonds, or other symbols to construct a flowchart, but these symbols help to describe the process in the chart clearly. Few of the standard symbols which are applicable in most flowcharts are as follows:

➔ **Rounded box** - This symbol is used to represent events that appears automatically. Such an event will activate a subsequent action, for example `receiving telephone calls'.

- **Rectangle or box** - This symbol is used to represent an event that is controlled within the process. Typically, this is a step or action which is taken. In all flowcharts this will be the most frequently used symbol.

- **Diamond** - This symbol is used to represent a decision point in the process. Typically, the statement in the symbol will require a `yes' or `no' response and branch to different parts of the flowchart accordingly.

- **Parallelogram** - This symbol is used to denote input or output of information. For example, the input symbol can be depicted for input of an image in the computer by scanning, the process can be editing it, and the output can print the edited image.

- **Circle** - This symbol is used to represent a point at which the flowchart connects with another process. The name or reference to the other process should appear within the symbol.

Figure 10.5 shows some flowchart symbols.

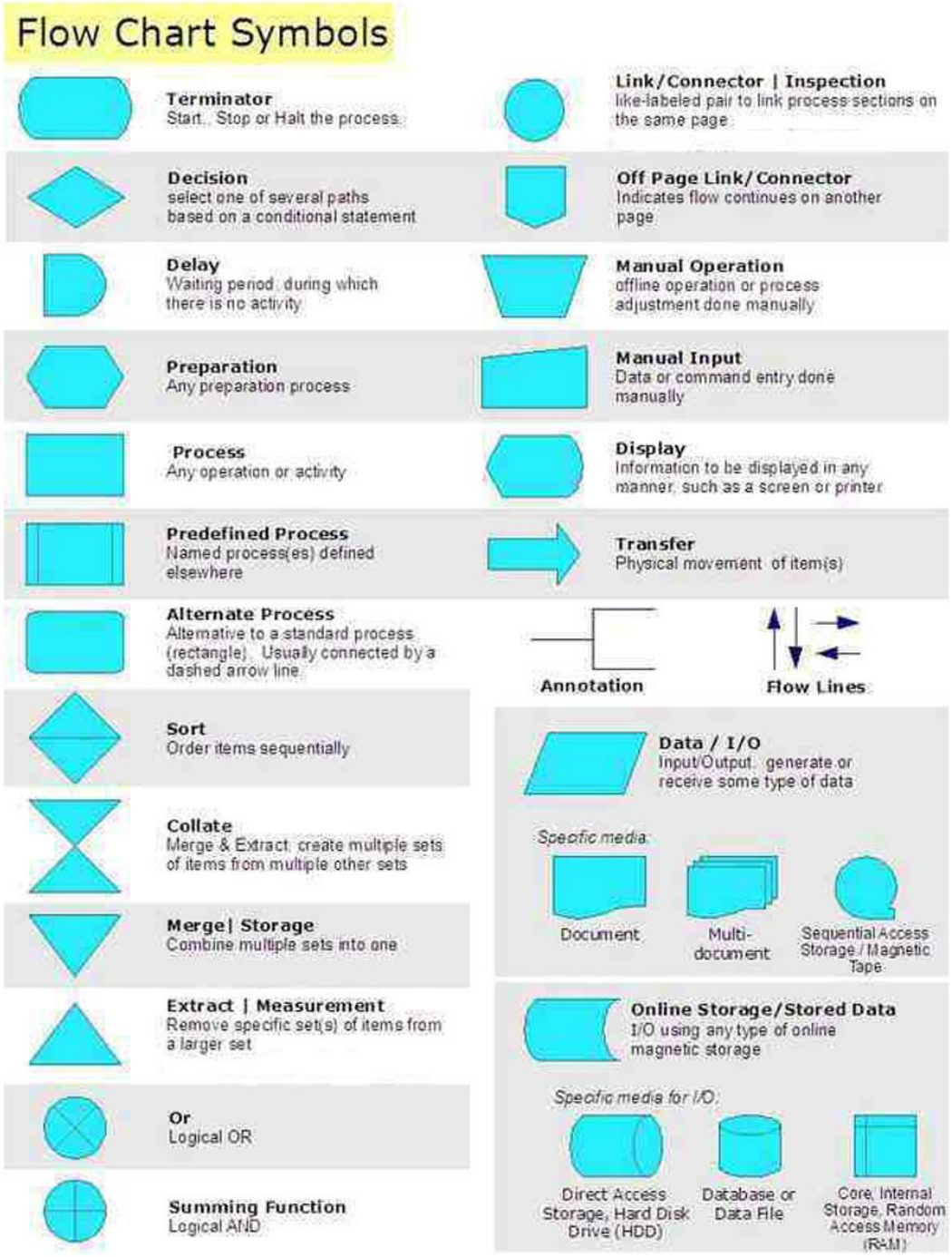


Figure 10.5: Flowchart Symbols

10.3 Data Flow Diagram

A Data Flow Diagram (DFD) is a modeling technique for examining and building information processes. DFD signifies as an illustration that explains the course or flow of information in a process. DFD shows this flow of information in a process based on the inputs and outputs. A DFD can be referred as a Process Model.

Additionally, a DFD can be utilized to visualize data processing or a structured design. A DFD represents technical or business processes with the assistance of external data stored, the data flow from one process to another, and the results.

DFDs consists of four major components:

- Entities
- Processes
- Data stores
- Data flows

DFDs are the method of choice over technical descriptions to communicate how information data flows through system and how data is transformed in the process. There are three principle reasons for using DFD and they are as follows:

1. DFDs are easily comprehended by technical and non-technical audiences.
2. DFDs offer a high level system overview, such as entire boundaries and connections to other systems.
3. DFDs provide a detailed illustration of system components.

A DFD illustrates the following:

- The sending and receiving of data in external devices.
- Changes in data specific to a process.
- If data is flowing by itself.
- Data storage locations in the process.

At present, DFDs are the preferred tool for showing program design, thus replacing flowcharts and pseudocode. A DFD illustrates those functions that have to be performed in a program as well as the data that the function requires. DFD depends on four symbols to state program design.

Table 10.1 shows four symbols used to write DFD.

DFD Components	DFD Symbols
External Entities	Rectangular box
Data Flow	Arrow headed lines
Process	Bubble (Circle or round corner square)
Data Store	Narrow opened rectangle

Table 10.1: Four Symbols Used to Write DFD

A designer usually creates a context-level DFD that illustrates the relationship between the entities inside and outside of a system as a single step. This basic DFD can then be divided into a lower level diagram demonstrating smaller steps displaying details of the system. Various levels are required to explain a complicated system.

Earlier, a user using copy machines experienced frequent paper jams and this resulted in wastage of time. Often, this problem could be cleared by simply opening and closing the access panel. So, a flowchart for troubleshooting procedure was created after observing the situation, thus making it easier to troubleshoot.

Figure 10.6 shows a sample DFD.

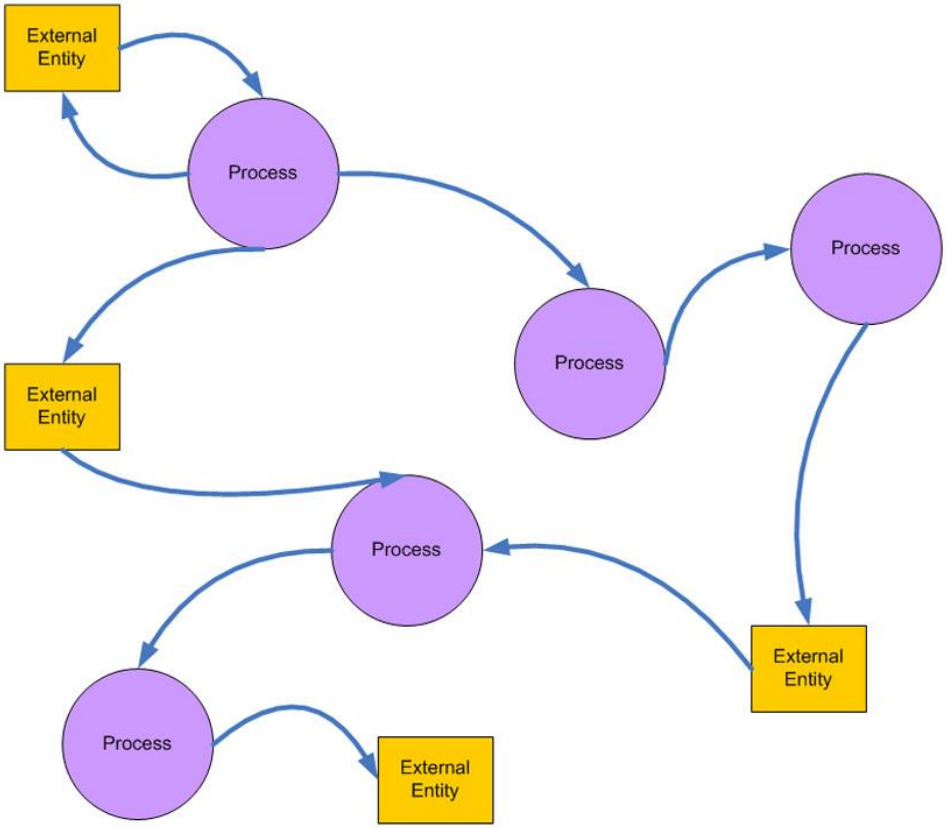


Figure 10.6: Sample DFD

10.4 Decision Tables

A decision table is a table composed of rows and columns, separated into four separate quadrants. Figure 10.7 shows four separate quadrants of decision table.

Conditions	Condition Alternatives
Actions	Action Entries

Figure 10.7: Four Separate Quadrants of Decision Table

The upper left quadrant includes the conditions. The upper right quadrant includes the condition rules of alternatives. The lower left quadrant includes the actions to be taken and the lower right quadrant includes the action rules.

In order to build decision tables, determine the maximum size of the table, eliminate any impractical situations, inconsistencies, redundancies, and simplify the table.

The guidelines for developing decision tables are as follows:

1. Determine the number of conditions that can affect the decision. Combine the rows that overlap, for example, conditions that are mutually exclusive. The number of conditions becomes the number of rows in the top half of the decision table.
2. Determine the number of possible actions that can be taken. This becomes the number of rows in the lower half of the decision table.
3. Determine the number of condition alternatives for each condition. In the simplest form of decision table, there would be two alternatives (Y or N) for each condition. In an extended-entry table, there may be many alternatives for each condition.
4. Calculate the maximum number of columns in the decision table by multiplying the number of alternatives for each condition. If there were four conditions and two alternatives (Y or N) for each of the conditions, there would be sixteen possibilities as follows:

   Condition 1: x 2 alternatives

   Condition 2: x 2 alternatives

   Condition 3: x 2 alternatives

   Condition 4: x 2 alternatives

   16 possibilities
5. Start with the first condition and divide the number of columns by the number of alternatives for that condition. Consider the 16 possibilities in the earlier guideline, which has sixteen columns and two alternatives (Y and N). Hence, sixteen divided by two is eight. Then, select one of the alternatives and write Y in all of the eight columns. Finally, write N in the remaining eight columns as shown in table 10.2.

   Condition 1 YYYYYYYYNNNNNNNN

   Repeat this for each condition using a subset of the table:

Y	Y	Y	Y	Y	Y	Y	Y	N	N	N	N	N	N	N	N
Y	Y	Y	Y	N	N	N	N								
Y	Y	N	N												
Y	N														

Table 10.2: 16 Possibilities in 16 Columns and 2 Alternatives

and continue the pattern for each condition as shown in table 10.3:

Condition 1	Y	Y	Y	Y	Y	Y	Y	Y	N	N	N	N	N	N	N	N
Condition 2	Y	Y	Y	Y	N	N	N	N	Y	Y	Y	Y	N	N	N	N

Condition 3	Y	Y	N	N	Y	Y	N	N	Y	Y	N	N	Y	Y	N	N
Condition 4	Y	N	Y	N	Y	N	Y	N	Y	N	Y	N	Y	N	Y	N

Table 10.3: Repeating the Possibilities on each Conditions

6. Complete the table by inserting an X where rules suggest certain actions.
7. Combine rules where it is apparent that an alternative does not make a difference in the outcome; for example:

   Condition 1 Y Y

   Condition 2 Y N

   Action 1 X X

   can be expressed as:

   Condition 1 Y

   Condition 2 --

   Action 1 X

   The dash (-) signifies that condition 2 can be either Y or N and action will still be taken.
8. Check table 10.3 for any impossible situations, contradictions, and redundancies.
9. Rearrange the conditions and actions (or even rules) to make the decision table more understandable.

10.5 HIPO Chart

The HIPO chart is a tool used to analyze a problem and visualize a solution using the top down design approach. Starting at the global (macro) level, the chart is decomposed repeatedly at ever-greater levels of detail until the logical building blocks (functions) are identified.

Figure 10.8 shows a sample HIPO chart.

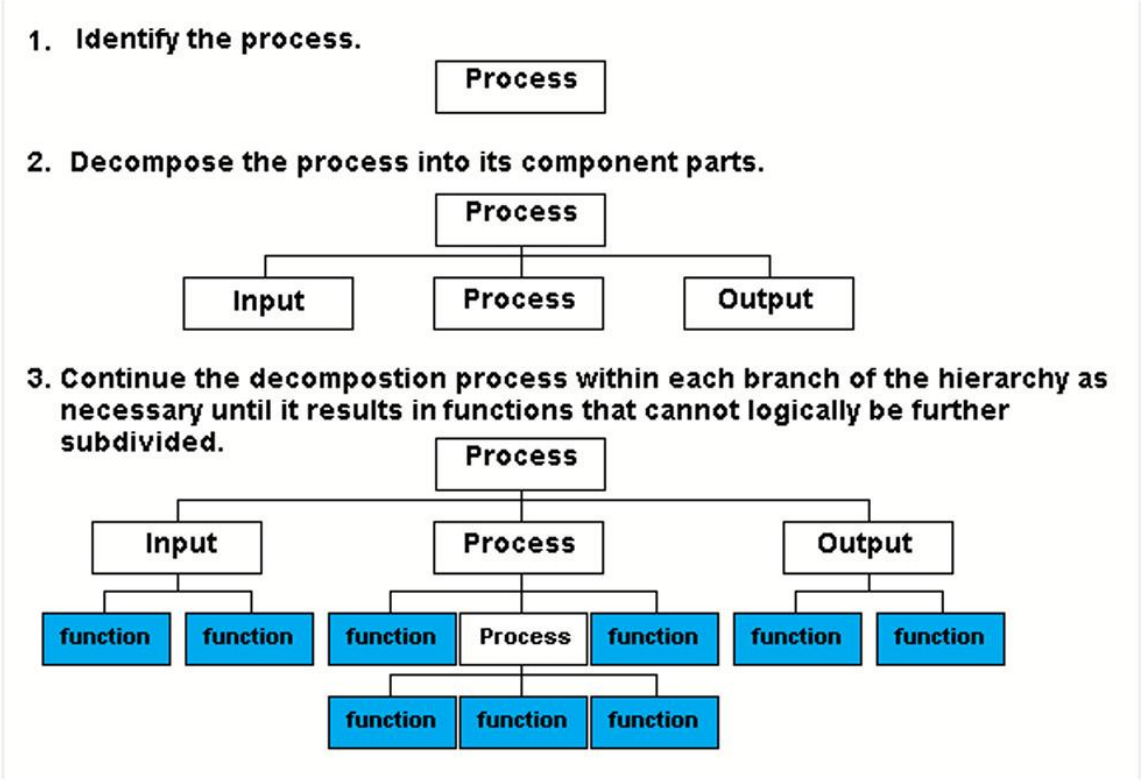


Figure 10.8: HIPO chart

A basic process is broken down and illustrated in figure 10.8. The process can be much more complicated and detailed in reality.

HIPO is a form driven technique that uses standard forms for documenting the information. It consists of a hierarchy chart and an associated set of input, process, and output charts. HIPO follows the top-down decomposition method. It describes the data input and output from processes and defines the data flow composition. It was developed by IBM as a design aid and implementation technique with the following objectives:

1. Provide a structure by which the functions of a system can be realized.
2. State the functions to be performed by the program.
3. Provide a visual description of the input used and the output produced for each level of the diagram.

HIPO uses vector symbols between processes that define data communication and data direction.

The procedure for generating HIPO diagrams are as follows:

1. Start at the highest level of abstraction and define the inputs to the system and the outputs from it

in aggregate terms.

2. Processing steps can be identified by those that convert input to output.
3. Document each element using HIPO diagram notation and the associated tree like structure.
4. Identify sub processes with their inputs and outputs. Continue decomposition until the processes cannot be decomposed any further.

The HIPO package format consists of the following:

1. Visual table of content which depicts the structure of the diagram and the relationships of the functions in a hierarchical manner. It also has legends to explain function of each symbols.
2. Major functions are stated in overview diagrams. Overview diagram refers to the detail diagrams, which is required to expand the functions effectively.

10.6 Check Your Progress

1. Flowcharts are usually drawn in the early stages of developing _____.

(A)	hardware solutions	(C)	computer solutions
(B)	system solutions	(D)	storage solutions

2. Which of the following are the major components of DFD?

(A)	Entities	(C)	Data Flow
(B)	System Flow	(D)	Sequential Flow

3. Which of the following flowchart provides detailed depiction of a process by mapping all the steps and activities that appear in the process?

(A)	High-level flowchart	(C)	Deployment or Matrix flowchart
(B)	Detailed flowchart	(D)	Indexed flowchart

4. HIPO is a _____ driven technique that uses standard forms for documenting the information.

(A)	forms	(C)	algorithm
(B)	table	(D)	code

5. Match the following:

DFD Components		DFD Symbols	
(a)	External Entities	1.	Bubble (Circle or round corner square)
(b)	Data Flow	2.	Narrow opened rectangle
(c)	Process	3.	Rectangular box
(d)	Data Store	4.	Arrow headed lines

(A)	a-4, b-3, c-2, d-1	(C)	a-3, b-4, c-1, d-2
(B)	a-2, b-3, c-4, d-1	(D)	a-2, b-1, c-4, d-3

10.6.1 Answers

1.	C
2.	A, C
3.	B
4.	A
5.	C

- The system flowchart is a way of visually presenting the flow of data through an information processing system, the operations executed within the system and the order in which they are performed.
- A flowchart is the graphical representation of how a process works and depicts the sequence of steps.
- A Data Flow Diagram is a modeling technique for examining and building information processes.
- A decision table is a table composed of rows and columns, separated into four separate quadrants.
- The HIPO chart is a tool used to analyze a problem and visualize a solution using the top down design approach.
- HIPO is a form driven technique that uses standard forms for documenting the information.

Session 10

Data Flow Diagrams

Aptech™ COMPUTER EDUCATION Unleash your potential

Session - 11

Elements of Programming Language

Welcome to the session, **Elements of Programming Language**.

This session explains software program and characteristics of a good software program. It also explains the program development process and programming languages. Finally, this session explains program execution and program testing.

At the end of this session, you will be able to:

- Define software programs
- Describe the program development process
- Explain programming languages
- Explain program execution
- Explain program testing

Session 11

Elements of Programming Language

11.1 Introduction

Programming languages are computer languages used to describe the processes or tasks which a computer has to perform. Programming languages can also be called as tools used to instruct a computer. These languages must be concise, clear, and easy to understand. Programming languages are used to create either system software or application software. System software is used to control the operations of computer hardware. Application software helps the user perform tasks with the help of both system software and computer hardware.

11.2 Software Programs

Software programs are a set of written procedures or instructions in a sequential format that are used to perform specific tasks by a computer. The task is executed when the software program is initiated. Software programs are created for business requirement as well as personal needs.

All software programs must have certain characteristics, regardless of the task they perform. Some of the characteristics that must be present in a software program are as follows:

- **Scalability** – Scalability is the ability of a system to handle increasing workload or number of users. For example, it is the capability of a system to increase the total throughput when the load is increased by addition of more resources.

- **Security** – Software must be secure and must not result in loss of data or information. For this, correct authentication measures must be programmed in a software to prevent data loss due to external threats.

- **Usability** – The software must be easy to learn without putting in much effort. It must not be too complicated for a user.

- **Reusability** – The software code that is created known as the source code must be reusable to create any other new software on similar lines as the existing software.

- **Modularity** – The software must be divided into different modules or units. In case of any change in a module, the whole software must not be affected except for the module in which the change is required. These modules must be integrated later but must be independent of each other for the purpose of editing and testing.

- **Testability** – All software programs that are created must undergo the testing round. There are different rounds of testing that a software program may have to go through.

- **Interoperability** – Software must be able to exchange information with other software programs if required for task completion.

- **Accuracy** – The software that is created must adhere to the software specifications provided and must give accurate results as expected of the software.

- **Efficiency** – Software programs must use the available resources efficiently. Resources can include storage space, CPU speed, and so on.
- **Flexibility** – Any changes to be made to the software program must be easy to make, without much effort required.
- **Portability** – The software should be compatible in different locations which may have different performance environments, platforms, and so on.
- **Reliability** – The software must be built in a sturdy manner, so that it does not crash during execution. It should be seen that the software is free of defects.

11.3 Program Development Process

The program development process includes the general steps that are used for creating a software program, using different methodologies and resources. This process forms the framework for creating any kind of software for business and personal use. Figure 11.1 shows the program development process.

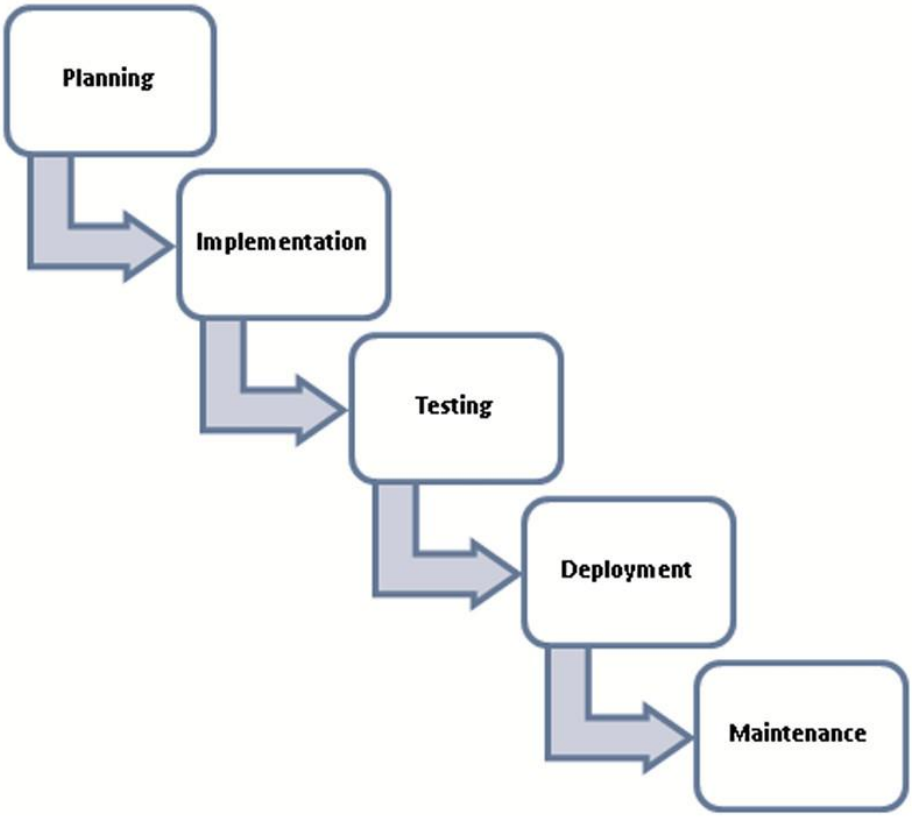


Figure 11.1: Program Development Process

➔ **Planning** – Before starting with the procedure of development of software, the requirements for the software have to be gathered. These requirements are generally given by the clients to the IT software development company. This process of collecting the requirements is called requirement gathering. Then, the requirements are analyzed and any queries and discrepancies are resolved. After the requirements are gathered, the scope document is created. This clearly defines the scope of the project and also the costs are mentioned. This document has to be approved by both the client and the company creating the software.

➔ **Implementing** – During the implementation stage, the software programmer has to actually write the code and create the software based on the requirements that were specified from the client. For this, sometimes a demo version of the software is created, known as Prototype. This prototype software is not fully functional. After the prototype is approved by the client, the full software is created keeping the prototype as the base. Implementation also includes installing the environment that is convenient for the software. It includes creating and connecting to databases if required. It also includes compiling and integrating the different modules of the software together at the end of the development.

➔ **Testing** – The testing process is conducted to address all defects that may be present in the software program. These defects may have been missed out by the programmer during the actual coding, but may prove fatal to the software program. The first objective that testing is to see that the software development meets the conformance as per the requirements that were specified by the client. Testing also reduces errors and provides quality assurance to the software package that is developed.

➔ **Deployment** – After the testing is complete, and the product is fully approved, it is released at the client-end. This process is known as deployment. First, the software is installed at the locations where the client requires the software to be used. So additional programming may be required to provide customization to the client as per the location. Also, sometimes there is another round of testing that is conducted at the client-end. The client is also asked to use the product and any doubts regarding the usage of the software are resolved during the deployment stage. Generally, training about the software also forms a part of the deployment stage.

➔ **Maintenance** – There may be certain routine tasks that may be required to be performed on the software after certain intervals. These tasks form a part of maintenance. For example, reports are required to be generated on a monthly basis using the software. This forms a maintenance task. There may also be additional support that the client may require. This also forms a part of maintenance.

11.4 Programming Languages

A programming language is a language designed using formal codes of reserved words and symbols that express the process to be followed by a computer. It contains the instructions that need to be provided to the computer for performing tasks in a precise manner.

11.4.1 Types of Programming Languages

Programming languages are developed in different ways and are used for different tasks. These tasks can be related to any sectors such as banking, construction, aviation, and so on. For easier understanding, the programming languages are classified in the following types:

- **Structured Programming** – Structured programming was one of the first types of programming languages that were used for development of computer programs. The clarity required for development of a computer program using structures and loops was first seen in structured languages. Structured programming allowed reuse of code and required less memory for execution. Structured programming uses procedures, also known as methods and functions that contain a serious of instructions to be executed. Some examples of structured programming languages are BASIC, C, and so on.

- **Object-oriented Programming** – After structured programming language, Object-oriented Programming (OOP) language was introduced. An object is any person or a thing, living or non-living which has some characteristics or attributes which help to describe it. These objects have their own specific characteristics and features. OOP recognizes the need of building an application that represents the real world. This can be used by using objects. Figure 11.2 shows examples of objects used for a car.

Figure 11.2: Objects for a Car

All these objects come together to create a car. In the same way, OOP uses 'objects' which are data structures consisting of attributes and behavior along with their interaction, for designing

computer programs. OOP is designed to provide flexibility and maintainability to a program. The code written using OOP program is simpler and easy to understand as well as to maintain. Some examples of OOP are C++, Java, and so on.

- **Aspect-oriented Programming** – Aspect-oriented programming (AOP) involves breaking the program logic into distinct parts, also called concerns that are cohesive areas of functionality. For example, at times security-related operations are applied in such a way that they remain scattered across several methods. Now, a change to the operation would require a lot of effort. AOP helps to solve this problem by allowing cross-cutting concerns to be expressed in stand-alone modules, also called aspects. Aspects may contain advice, that is, code connected to specific points in the program and inter-type declarations, that is, structural members added to other classes. All the concerns with regards to a certain module are first resolved. This means only one aspect is focused on. The next aspect is focused on after the concerns regarding the currently developed module are resolved. Some examples of AOP are AspectC++, AspectJ, AspectLua, and so on.

11.4.2 Features of a Programming Language

Though programming languages have evolved over the number of years, the common features of a programming language are still the same. Some of these common features are as follows:

- All languages must be compiled preferably using a compiler. There can be different ways to implement the compiling process but the process must be followed for all the programming languages.
- A program should have a memory handling and data storage facility. Usually arrays are used for data storage and are also known as data structures.
- Control statements must be included in a programming language for logical implementation of the program.
- The programs, functions, and procedures should be written as well as provision should be provided for editing as required.
- The built-in functionalities of a program such as classes and functions must be used to write better programs.
- Libraries must be created in a software program so that these libraries can be used in other software programs also.
- Most programming languages support exception handling. This feature of a programming language must be used to find any errors that may be present in the computer program that is developed.

➔ Operators must be used to execute operations especially those that are mathematical in nature.

➔ Ground rules for writing a program must be specified and followed by all programmers before starting to write a code for a program.

➔ Procedures are used to execute a program and functions must be used to return values after the execution of a program.

11.5 Program Execution

Program execution is a process where the computer performs the instructions that are given to it by a computer program. These instructions initiate a sequence of tasks that are executed by the machine. After the tasks are performed, the user receives certain results as specified in the program. Programs can be executed by a person or may even be automatically executed given certain triggers for execution. There are two methods that are used to execute a program written in a high-level language. These two methods are as follows:

➔ **Interpreter** – An interpreter is used to execute instructions that are written in high-level languages. The interpreter translates the high-level instructions into an intermediate language that the machine can understand. In this way, an interpreter helps to execute a program. Interpreters are used by most high-level languages such a BASIC and LISP.

➔ **Compiler** – A complier is used to translate the source code into the object code. The compiler checks the entire program and check for any compilation issues. All programs must go through a compiler before execution to check the integrity of the code. All high-level programming languages have a compiler already integrated within them. This complier determines if the instructions provided are acceptable.

The process of compiling and interpreting is shown in figure 11.3.

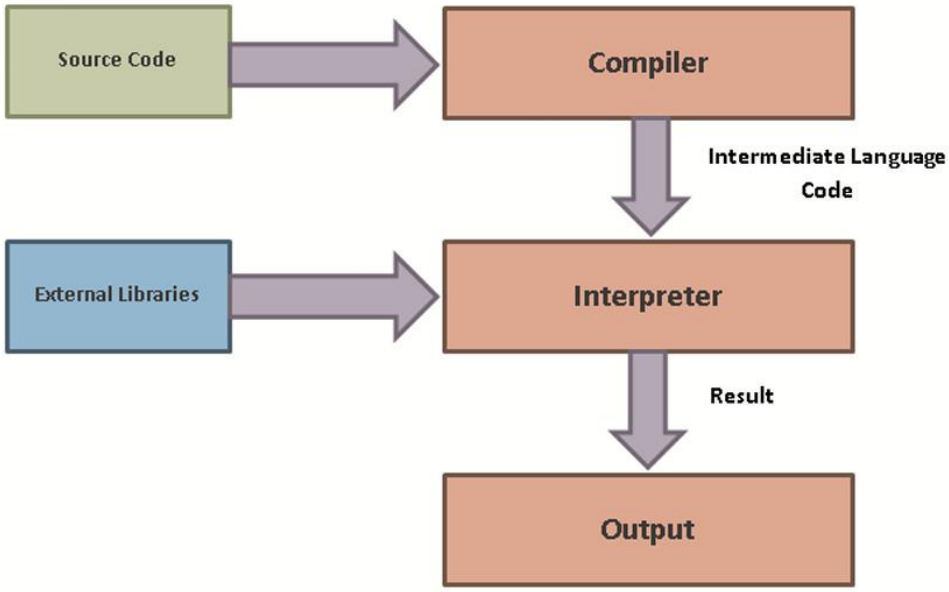


Figure 11.3: Process of Compiling and Interpreting

11.6 Program Testing

Program testing is a process of validating, evaluating, and verifying the capability of a software program and determining if the software program has met the required results. Program testing is an important part of the program development process and must be conducted for all software development programs. Program testing provides quality assurance, thus proving that the software is accurate and reliable to use. Some of the aspects that form a part of testing are as follows:

a. The software must meet the requirements specified

b. All the operations of the software must be in working condition

c. The implementation of the software at the client-end must not create any issues

d. The software must satisfy the needs of the users

The popular software testing tools available are as follows:

➔ HP QuickTest Pro

➔ Rational Robot

➔ eggPlant

- SilkTest
- TestComplete
- TestPartner
- TestManager
- Visual Studio Test Professional

Testing of a program is a detailed process that needs to be followed by a programmer in collaboration with the testing team. The process of testing is shown in figure 11.4.

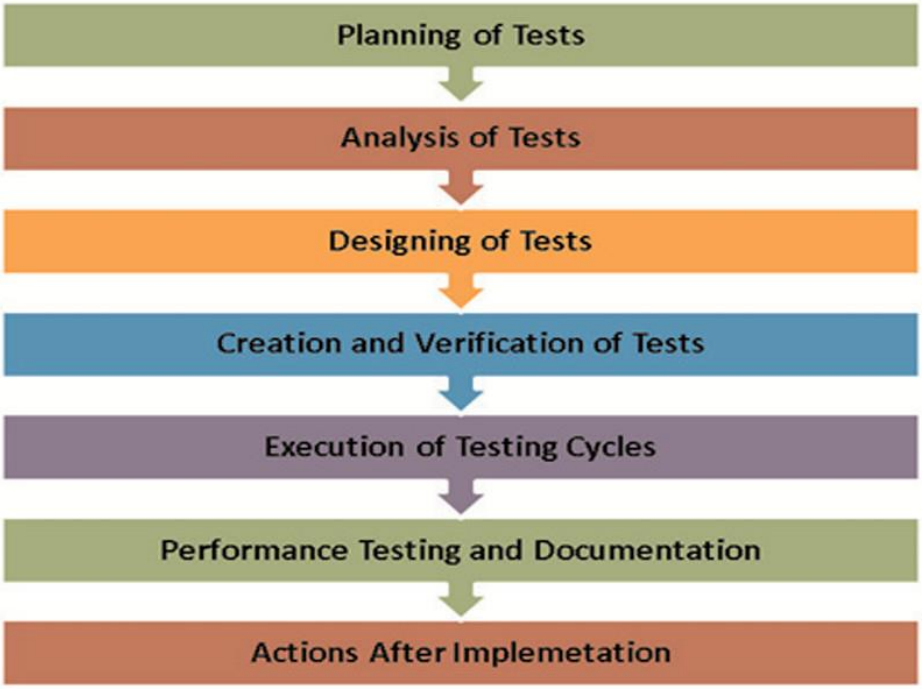


Figure 11.4: Process of Testing

- **Planning of Tests** – In this stage, the software requirements are provided to the testing team. Also, non-functional requirements for the software are also specified. The timelines for the testing process are also decided in this stage.
- **Analysis of Tests** – Based on the functional and non-functional requirements received and considering the timelines, the analysis of the testing process takes place.

The test strategy is prepared taking these into confirmation. Also, the test plan documents are created and signed off by the client.

- **Designing of Tests** – A high-level design document is created in this stage. This document gives an outline of the overall test process flow. This document also helps to trace data related to the testing process easily.

- **Creation and Verification of Tests** – The test cases are created in this stage. Also, the specifications for testing are documented and signed off from the client at this stage.

- **Execution of Testing Cycles** – In this stage, the actual testing process starts and the software goes through different rounds of testing. The defects found in the software are recorded and sent to the programming team for resolution. These defects are stored in a Test Defect Report. After the programmer addressed and resolves the defect, the defect status is marked as closed.

- **Performance Testing and Documentation** – After the rounds of testing are complete, the overall software is tested from performance point of view and defect if any are logged and resolved. After all the defects are closed, the test completion report is created and signed off by the client.

- **Actions After Implementation** – After the testing process is completion is complete, the recommendations for process improvement are suggested in this stage.

11.7 Check Your Progress

1. Which characteristic indicates the ability of a system to handle increasing workload or number of users?

(A)	Reliability	(C)	Portability
(B)	Scalability	(D)	Usability

2. Which characteristic specifies that software must be able to exchange information with other software programs?

(A)	Testability	(C)	Accuracy
(B)	Interoperability	(D)	Efficiency

3. Which process of the program development process is conducted to address all defects that may be present in the software program?

(A)	Implementing	(C)	Deployment
(B)	Testing	(D)	Maintenance

4. The focus of aspect-oriented programming is on _____.

(A)	Structures	(C)	Aspects
(B)	Objects	(D)	Testing

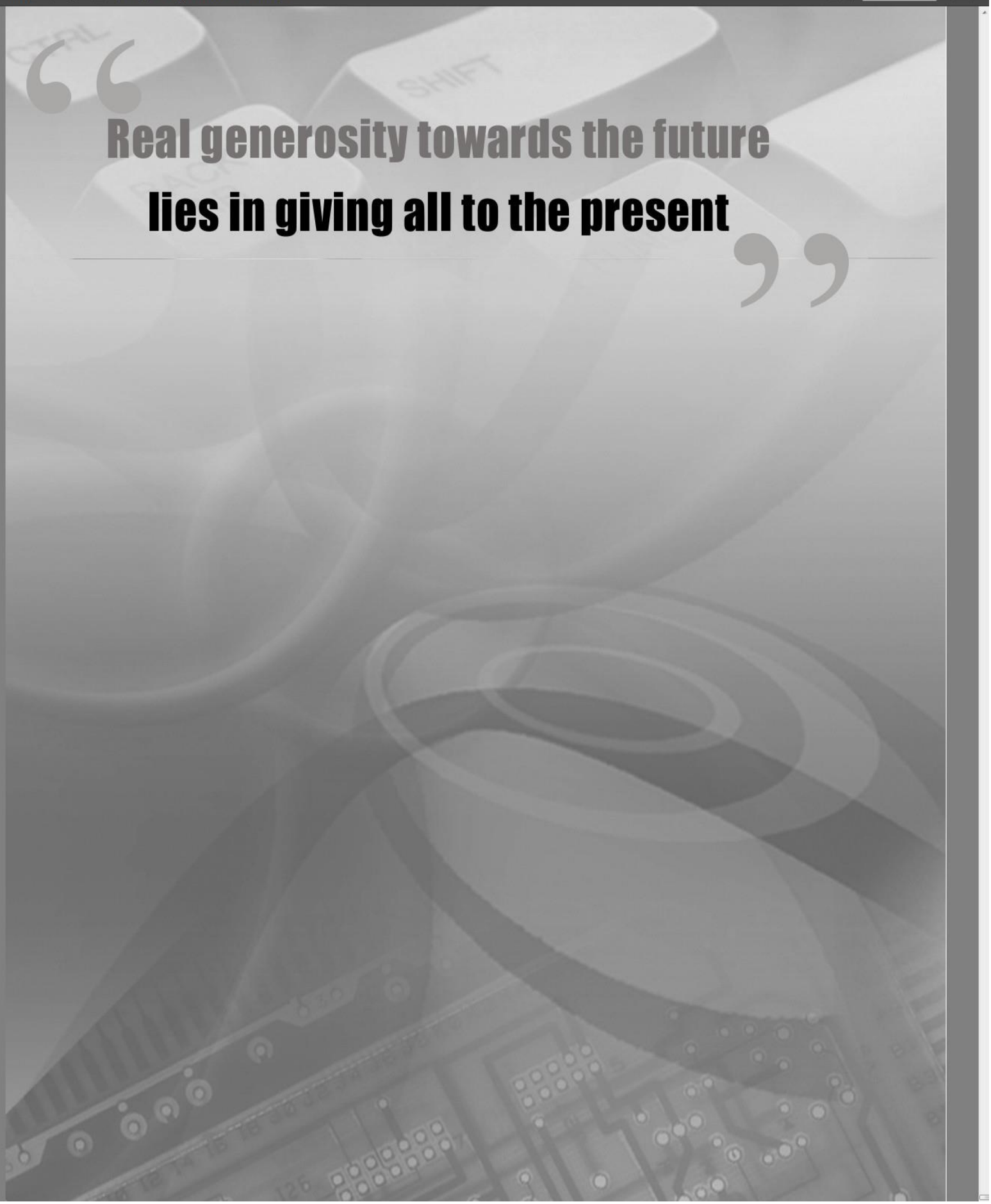
5. _____ are used to execute a program.

(A)	Procedures	(C)	Executors
(B)	Functions	(D)	Requirements

11.7.1 Answers

1.	B
2.	B
3.	B
4.	C
5.	A

- Software programs are a set of written procedures or instructions in a sequential format that are used to perform specific tasks by a computer.
- The program development process includes planning, implementation, testing, deployment, and maintenance.
- A programming language is a language designed using formal codes of reserved words and symbols that express the process to be followed by a computer.
- The programming languages are classified as structured programming, object-oriented programming, and aspect-oriented programming.
- An interpreter is used to execute instructions that are written in high-level languages.
- A complier is used to translate the source code into the object code.
- Program testing is a process of validating, evaluating, and verifying the capability of a software program and determining if the software program has met the required results.


Real generosity towards the future
lies in giving all to the present