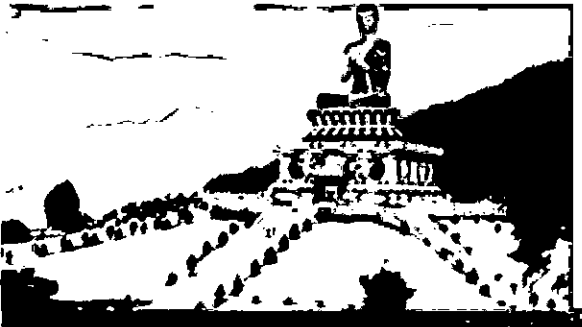
BOARD OF OPEN SCHOOLING AND SKILL EDUCATION
Near Indira Bypass, NH-10, Gangtok, East Sikkim- 737102
Telephone : 03592-295335, 94066 46682 Email : bosse.org.in









The Pathways To Higher Studies

Computer Science

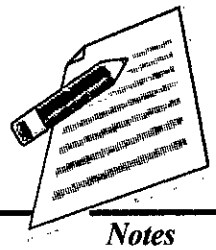
Class-XII

Developed & Published by:
Board of Open Schooling and Skill Education

Copyright:
Board of Open Schooling and Skill Education

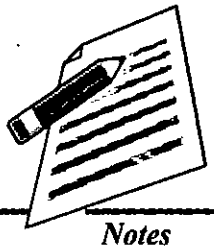
Note:
While writing and printing the book every attention has been given to make it free from all sorts of errors. However, no such assurance can be given that no errors creep. In case of any error and consequently any loss should not be a mater of liability for the publisher, the author or any concern person.

COMPUTER SCIENCE



Syllabus

Module No.	Chapter No.	Chapter Name	Duration (Hrs.)	Marks
1	1	Anatomy of a Digital Computer	54	15
	2	Data Processing Concept		
	3	Computer Software		
	4	Operating System		
	5	Data Communication and Networking		
	6	Fundamentals of Internet and Java Programming		
2	7	Introduction to C++	186	85
	8	General Concept of OOP		
	9	Control Statements		
	10	Functions		
	11	Array		
	12	Structure, Typedef& enumerated data type		
	13	Classes & objects with constructors / destructors		
	14	Inheritance extending classes		
	15	Pointer		
	16	Files		
		Total	**240**	**100**



1 ANATOMY OF DIGITAL COMPUTER

- Understand the concept of computer.
- Discuss the characteristics of computer.
- Describe the components of computer.
- Discuss the scope of computer.

Objective of the chapter:

The basic objective of this chapter is to through some light on the initial concepts of computer so that the fundamentals of computer can be learned.

Introduction

Digital Computer, any of a class of devices capable of solving problems by processing information in discrete form. It operates on data, including magnitudes, letters, and symbols, that are expressed in binary form—i.e., using only the two digits 0 and 1. By counting, comparing, and manipulating these digits or their combinations according to a set of instructions held in its memory, a digital computer can perform such tasks as to control industrial processes and regulate the operations of machines.

We should keep in mind that a computer is a programmable machine. The two main characteristics of a computer are:

-> It responds to a specific set of instructions in a well-defined manner.

-> It can execute a pre-recorded list of instructions (a program).

Modern computers are electronic and digital. The actual machinery – wires, transistors and circuits is called hardware. The instructions and data are called software. All general purpose computers require the following hardware components:

1. Central Processing Unit (CPU) – The 'brain' of the computer, the component that actually executes instructions.
2. Memory – It enables a computer to store, at least temporarily, data and programs.
3. Input device – Usually a keyboard or mouse is used to read data and programs into the computer.
4. Output device – A display screen, printer, etc. that lets you see what the computer has accomplished.
5. Mass storage device – It allows a computer to permanently store large amounts of data. Common mass storage devices include disk drive and tape drive.

In addition to these components, many others make it possible for the basic components of a computer to work together efficiently.

Functions and Components of a Computer

To function properly, the computer needs both hardware and software. Hardware consists of the mechanical and electronic devices, which we can see and touch. The software consists of programs, the operating system and the data that reside in the memory and storage devices. A computer does mainly the following four functions:

-> Receive input – Accept data/information from outside through various input devices like the keyboard, mouse, scanner, etc.

-> Process information – Perform arithmetic or logical operations on data/ information.

-> Produce output—Communicate information to the outside world through output devices like monitor, printer, etc.

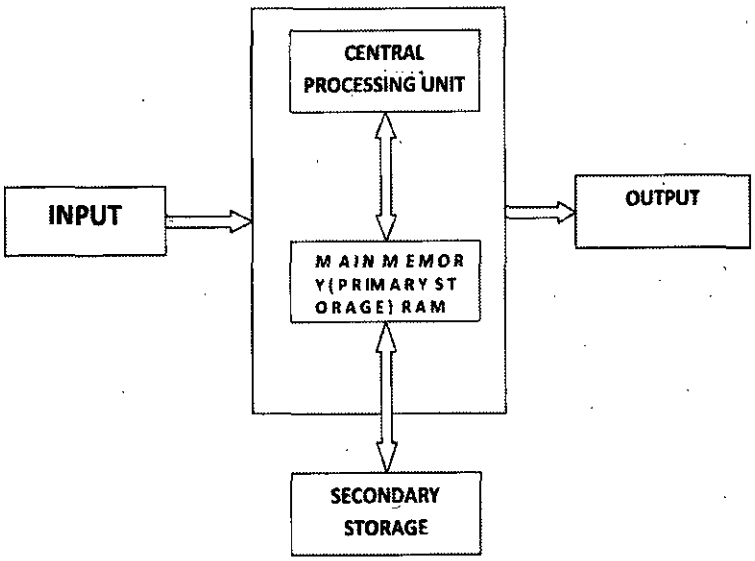
-> Store information—Store the information in storage devices like hard disk, floppy disks, CD, etc.

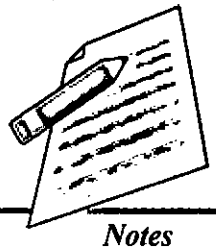
These four basic functions are responsible for everything that computers do. The hardware components of the computer specialize in any one of these functions. Computer hardware falls into two categories: processing hardware and the peripheral devices. The Processing hardware consists of the Central Processing Unit (CPU), and as its name implies, is where the data processing is done. Peripheral devices allow people to interact with the CPU. Together, they make it possible to use the computer for a variety of tasks.

BASIC COMPUTER ORGANIZATION:

A standard fully featured desktop configuration has basically four types of featured devices

1. Input Devices
2. Output Devices
3. Memory
4. Storage Devices





Notes

CPU OPERATION

The fundamental operation of most CPUs

To execute a sequence of stored instructions called a program.

1. The program is represented by a series of numbers that are kept in some kind of computer memory.
2. There are four steps that nearly all CPUs use in their operation: fetch, decode, execute, and write back.
3. Fetch: Retrieving an instruction from program memory.

The location in program memory is determined by a program counter (PC)

After an instruction is fetched, the PC is incremented by the length of the instruction word in terms of memory units.

Decode:

1. The instruction is broken up into parts that have significance to other portions of the CPU.
2. The way in which the numerical instruction value is interpreted is defined by the CPU's instruction set architecture (ISA).
3. Opcode indicates which operation to perform.
4. The remaining parts of the number usually provide information required for that instruction, such as operands for an addition operation.
5. Such operands may be given as a constant value or as a place to locate a value: a register or a memory address, as determined by some addressing mode.

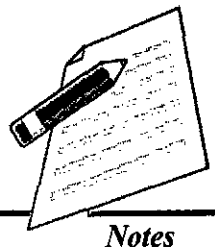
Execute:

1. During this step, various portions of the CPU are connected so they can perform the desired operation.
2. If, for instance, an addition operation was requested, an arithmetic logic unit (ALU) will be connected to a set of inputs and a set of outputs.
3. The inputs provide the numbers to be added, and the outputs will contain the final sum.
4. If the addition operation produces a result too large for the CPU to handle, an arithmetic overflow flag in a flags register may also be set.

Write back:

1. Simply "writes back" the results of the execute step to some form of memory.
2. Very often the results are written to some internal CPU register for quick access by subsequent instructions.
3. In other cases results may be written to slower, but cheaper and larger, main memory.

Some types of instructions manipulate the program counter rather than directly produce result data.



INPUT DEVICES

Anything that feeds the data into the computer. This data can be in alpha-numeric form which needs to be keyed-in or in its very basic natural form i.e. hear, smell, touch, see; taste & the sixth sense…feel?

Typical input devices are:

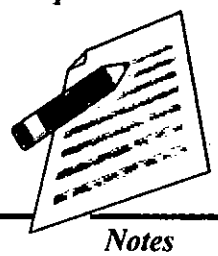
1. Keyboard
2. Mouse
3. Joystick
4. Digitizing Tablet
5. Touch Sensitive Screen
6. Light Pen
7. Space Mouse
8. Digital Stills Camera
9. Magnetic Ink Character Recognition (MICR)
10. Optical Mark Reader (OMR)
11. Image Scanner
12. Bar Codes
13. Magnetic Reader
14. Smart Cards
15. Voice Data Entry
16. Sound Capture
17. Video Capture

The **Keyboard** is the standard data input and operator control device for a computer. It consists of the standard QWERTY layout with a numeric keypad and additional function keys for control purposes.

The **Mouse** is a popular input device. You move it across the desk and its movement is shown on the screen by a marker known as a ‹cursor›. You will need to click the buttons at the top of the mouse to select an option.

Track ball looks like a mouse, as the roller is on the top with selection buttons on the side. It is also a pointing device used to move the cursor and works like a mouse. For moving the cursor in a particular direction, the user spins the ball in that direction. It is sometimes considered better than a mouse, because it requires little arm movement and less desktop space. It is generally used with Portable computers.

Magnetic Ink Character Recognition (MICR) is used to recognize the magnetically charged characters, mainly found on bank cheques. The magnetically charged characters are written by special ink called magnetic ink. MICR device reads the patterns of these characters and compares them with special patterns stored in memory. Using MICR device, a large volume of cheques can be processed in a day. MICR is widely used by the banking industry for the processing of cheques.



Notes

The joystick is a rotary lever. Similar to an aircraft's control stick, it enables you to move within the screen's environment, and is widely used in the computer games industry.

A **Digitising Tablet** is a pointing device that facilitates the accurate input of drawings and designs. A drawing can be placed directly on the tablet, and the user traces outlines or inputs coordinate positions with a hand-held stylus.

A **Touch Sensitive Screen** is a pointing device that enables the user to interact with the computer by touching the screen. There are three types of Touch Screens: pressure-sensitive, capacitive surface and light beam.

A **Light Pen** is a pointing device shaped like a pen and is connected to a VDU. The tip of the light pen contains a light-sensitive element which, when placed against the screen, detects the light from the screen enabling the computer to identify the location of the pen on the screen. Light pens have the advantage of ‹drawing› directly onto the screen, but this can become uncomfortable, and they are not as accurate as digitising tablets.

The **Space mouse** is different from a normal mouse as it has an X axis, a Y axis and a Z axis. It can be used for developing and moving around 3-D environments.

Digital Stills Cameras capture an image which is stored in memory within the camera. When the memory is full it can be erased and further images captured. The digital images can then be downloaded from the camera to a computer where they can be displayed, manipulated or printed.

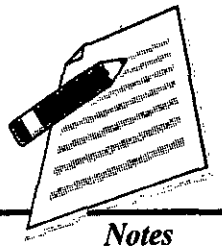
The Optical Mark Reader (OMR) can read information in the form of numbers or letters and put it into the computer. The marks have to be precisely located as in multiple choice test papers.

Scanners allow information such as a photo or text to be input into a computer. Scanners are usually either A4 size (flatbed), or hand-held to scan a much smaller area. If text is to be scanned, you would use an Optical Character Recognition (OCR) program to recognise the printed text and then convert it to a digital text file that can be accessed using a computer.

A **Bar Code** is a pattern printed in lines of differing thickness. The system gives fast and error-free entry of information into the computer. You might have seen bar codes on goods in supermarkets, in libraries and on magazines. Bar codes provide a quick method of recording the sale of items.

Card Reader: This input device reads a magnetic strip on a card. Handy for security reasons, it provides quick identification of the card's owner. This method is used to run bank cash points or to provide quick identification of people entering buildings.

Smart Card: This input device stores data in a microprocessor embedded in the card. This allows information, which can be updated, to be stored on the card. This method is used in store cards which accumulate points for the purchaser, and to store phone numbers for cellular phones.



OUTPUT DEVICES:

Output devices display information in a way that you can you can understand. The most common output device is a monitor. It looks a lot a like a TV and houses the computer screen. The monitor allows you to 'see' what you and the computer are doing together.

Brief of Output Device

Output devices are pieces of equipment that are used to get information or any other response out from computer. These devices display information that has been held or generated within a computer. Output devices display information in a way that you can understand. The most common output device is a monitor.

Types of Output Device

Printing: Plotter, Printer

Sound : Speakers

Visual : Monitor

A **Printer** is another common part of a computer system. It takes what you see on the computer screen and prints it on paper. There are two types of printers; Impact Printers and Non-Impact Printers.

Speakers are output devices that allow you to hear sound from your computer. Computer speakers are just like stereo speakers. There are usually two of them and they come in various sizes.

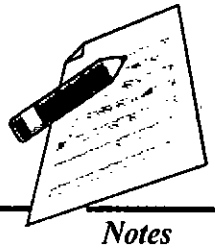
MEMORY OR PRIMARY STORAGE:

Purpose of Storage

The fundamental components of a general-purpose computer are arithmetic and logic unit, control circuitry, storage space, and input/output devices. If storage was removed, the device we had would be a simple calculator instead of a computer. The ability to store instructions that form a computer program, and the information that the instructions manipulate is what makes stored program architecture computers versatile.

Primary Storage

Primary storage is directly connected to the central processing unit of the computer. It must be present for the CPU to function correctly, just as in a biological analogy the lungs must be present (for oxygen storage) for the heart to function (to pump and oxygenate the blood). As shown in the diagram, primary storage typically consists of three kinds of storage:



Notes

Processors Register

It is the internal to the central processing unit. Registers contain information that the arithmetic and logic unit needs to carry out the current instruction. They are technically the fastest of all forms of computer storage.

Main memory

It contains the programs that are currently being run and the data the programs are operating on. The arithmetic and logic unit can very quickly transfer information between a processor register and locations in main storage, also known as a "memory addresses". In modern computers, electronic solid-state random access memory is used for main storage, and is directly connected to the CPU via a "memory bus" and a "data bus".

Cache memory

It is a special type of internal memory used by many central processing units to increase their performance or "throughput". Some of the information in the main memory is duplicated in the cache memory, which is slightly slower but of much greater capacity than the processor registers, and faster but much smaller than main memory.

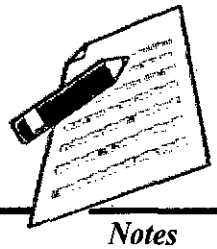
Memory

Memory is often used as a shorter synonym for Random Access Memory (RAM). This kind of memory is located on one or more microchips that are physically close to the microprocessor in your computer. Most desktop and notebook computers sold today include at least 512 megabytes of RAM (which is really the minimum to be able to install an operating system). They are upgradeable, so you can add more when your computer runs really slowly.

The more RAM you have, the less frequently the computer has to access instructions and data from the more slowly accessed hard disk form of storage. Memory should be distinguished from storage, or the physical medium that holds the much larger amounts of data that won't fit into RAM and may not be immediately needed there.

Storage devices include hard disks, floppy disks, CDROMs, and tape backup systems. The terms auxiliary storage, auxiliary memory, and secondary memory have also been used for this kind of data repository.

RAM is temporary memory and is erased when you turn off your computer, so remember to save your work to a permanent form of storage space like those mentioned above before exiting programs or turning off your computer.



TYPES OF RAM:

There are two types of RAM used in PCs - Dynamic and Static RAM.

Dynamic RAM (DRAM): The information stored in Dynamic RAM has to be refreshed after every few milliseconds otherwise it will get erased. DRAM has higher storage capacity and is cheaper than Static RAM.

Static RAM (SRAM): The information stored in Static RAM need not be refreshed, but it remains stable as long as power supply is provided. SRAM is costlier but has higher speed than DRAM.

Additional kinds of integrated and quickly accessible memory are Read Only Memory (ROM), Programmable ROM (PROM), and Erasable Programmable ROM (EPROM). These are used to keep special programs and data, such as the BIOS, that need to be in your computer all the time. ROM is "built-in" computer memory containing data that normally can only be read, not written to (hence the name read only).

ROM contains the programming that allows your computer to be "booted up" or regenerated each time you turn it on. Unlike a computer's random access memory (RAM), the data in ROM is not lost when the computer power is turned off. The ROM is sustained by a small long life battery in your computer called the CMOS battery. If you ever do the hardware setup procedure with your computer, you effectively will be writing to ROM. It is non-volatile, but not suited to storage of large quantities of data because it is expensive to produce. Typically, ROM must also be completely erased before it can be rewritten,

PROM (Programmable Read Only Memory)

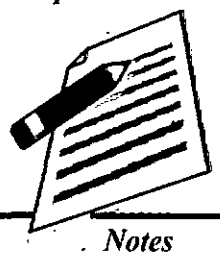
A variation of the ROM chip is programmable read only memory. PROM can be programmed to record information using a facility known as PROM-programmer. However once the chip has been programmed the recorded information cannot be changed, i.e. the PROM becomes a ROM and the information can only be read.

EPROM (Erasable Programmable Read Only Memory)

As the name suggests the Erasable Programmable Read Only Memory, information can be erased and the chip programmed a new to record different information using a special PROM-Programmer. When EPROM is in use information can only be read and the information remains on the chip until it is erased.

STORAGE DEVICES

The purpose of storage in a computer is to hold data or information and get that data to the CPU as quickly as possible when it is needed. Computers use disks for storage: hard disks that are located inside the computer and floppy or compact disks that are used externally.



Notes

- Computers Method of storing data & information for long term basis i.e. even after PC is switched off.
- It is non - volatile
- Can be easily removed and moved & attached to some other device
- Memory capacity can be extended to a greater extent
- Cheaper than primary memory

Storage Involves Two Processes

a) Writing data b) Reading data

Floppy Disks

The floppy disk drive (FDD) was invented at IBM by Alan Shugart in 1967. The first floppy drives used an 8-inch disk (later called a "diskette" as it got smaller), which evolved into the 5.25-inch disk that was used on the first IBM Personal Computer in August 1981. The 5.25-inch disk held 360 kilobytes compared to the 1.44 megabyte capacity of today's 3.5-inch diskette.

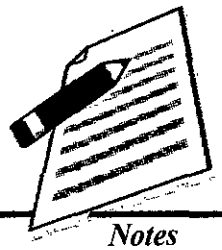
The 5.25-inch disks were dubbed "floppy" because the diskette packaging was a very flexible plastic envelope, unlike the rigid case used to hold today's 3.5-inch diskettes.

By the mid-1980s, the improved designs of the read/write heads, along with improvements in the magnetic recording media, led to the less-flexible, 3.5-inch, 1.44-megabyte (MB) capacity FDD in use today. For a few years, computers had both FDD sizes (3.5-inch and 5.25-inch). But by the mid-1990s, the 5.25-inch version had fallen out of popularity, partly because the diskette's recording surface could easily become contaminated by fingerprints through the open access area.

When you look at a floppy disk, you'll see a plastic case that measures 3 1/2 by 5 inches. Inside that case is a very thin piece of plastic that is coated with microscopic iron particles. This disk is much like the tape inside a video or audio cassette. Basically, a floppy disk drive reads and writes data to a small, circular piece of metal-coated plastic similar to audio cassette tape.

At one end of it is a small metal cover with a rectangular hole in it. That cover can be moved aside to show the flexible disk inside. But never touch the inner disk - you could damage the data that is stored on it. On one side of the floppy disk is a place for a label. On the other side is a silver circle with two holes in it. When the disk is inserted into the disk drive, the drive hooks into those holes to spin the circle. This causes the disk inside to spin at about 300 rpm! At the same time, the silver metal cover on the end is pushed aside so that the head in the disk drive can read and write to the disk.

Floppy disks are the smallest type of storage, holding only 1.44MB.



3.5-inch Diskettes (Floppy Disks) features:

- Spin rate: app. 300 revolutions per minute (rpm)
- High density (HD) disks more common today than older double density (DD) disks
- Storage Capacity of HD disks is 1.44 MB

Floppy Disk Drive Terminology

Floppy disk - Also called diskette. The common size is 3.5 inches.

Floppy disk drive - The electromechanical device that reads and writes floppy disks.

Track - Concentric ring of data on a side of a disk.

Sector - A subset of a track, similar to wedge or a slice of pie.

It consists of a read/write head and a motor rotating the disk at a high speed of about 300 rotations per minute. It can be fitted inside the cabinet of the computer and from outside, the slit where the disk is to be inserted, is visible. When the disk drive is closed after inserting the floppy inside, the monitor catches the disk through the Central of Disk hub, and then it starts rotating.

There are two read/write heads depending upon the floppy being one sided or two sided. The head consists of a read/write coil wound on a ring of magnetic material. During write operation, when the current passes in one direction, through the coil, the disk surface touching the head is magnetized in one direction. For reading the data, the procedure is reverse. I.e. the magnetized spots on the disk touching the read/write head induce the electronic pulses, which are sent to CPU.

The major parts of a FDD include:

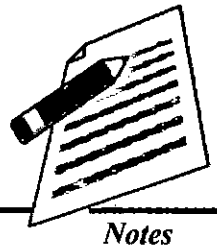
Read/Write Heads: Located on both sides of a diskette, they move together on the same assembly. The heads are not directly opposite each other in an effort to prevent interaction between write operations on each of the two media surfaces. The same head is used for reading and writing, while a second, wider head is used for erasing a track just prior to it being written. This allows the data to be written on a wider "clean slate," without interfering with the analog data on an adjacent track.

Drive Motor: A very small spindle motor engages the metal hub at the center of the diskette, spinning it at either 300 or 360 rotations per minute (RPM).

Stepper Motor: This motor makes a precise number of stepped revolutions to move the read/write head assembly to the proper track position. The read/write head assembly is fastened to the stepper motor shaft.

Mechanical Frame: A system of levers that opens the little protective window on the diskette to allow the read/write heads to touch the dual-sided diskette media. An external button allows the diskette to be ejected, at which point the spring-loaded protective window on the diskette closes.

Circuit Board: Contains all of the electronics to handle the data read from or written to the diskette. It also controls the stepper-motor control circuits used to move the read/write heads to each track, as well as the movement of the read/write heads toward the diskette surface.



Electronic optics check for the presence of an opening in the lower corner of a 3.5-inch diskette (or a notch in the side of a 5.25-inch diskette) to see if the user wants to prevent data from being written on it.

Hard Disks

Your computer uses two types of memory: primary memory which is stored on chips located on the motherboard, and secondary memory that is stored in the hard drive. Primary memory holds all of the essential memory that tells your computer how to be a computer. Secondary memory holds the information that you store in the computer.

Inside the hard disk drive case you will find circular disks that are made from polished steel. On the disks, there are many tracks or cylinders. Within the hard drive, an electronic reading/writing device called the head passes back and forth over the cylinders, reading information from the disk or writing information to it. Hard drives spin at 3600 or more rpm (Revolutions Per Minute) - that means that in one minute, the hard drive spins around over 7200 times!

Optical Storage

- Compact Disk Read-Only Memory (CD-ROM)
- CD-Recordable (CD-R)/CD-Rewritable (CD-RW)
- Digital Video Disk Read-Only Memory (DVD-ROM)
- DVD Recordable (DVD-R/DVD Rewritable (DVD-RW)
- Photo CD

Optical Storage Devices Data is stored on a reflective surface so it can be read by a beam of laser light. Two Kinds of Optical Storage Devices

- CD-ROM (compact disk read-only memory)
- DVD-ROM (digital video disk read-only memory)

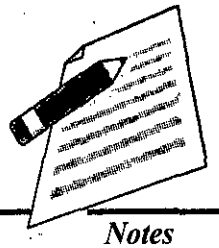
Compact Disks

Instead of electromagnetism, CDs use pits (microscopic indentations) and lands (flat surfaces) to store information much the same way floppies and hard disks use magnetic and non-magnetic storage. Inside the CD-ROM is a laser that reflects light off of the surface of the disk to an electric eye. The pattern of reflected light (pit) and no reflected light (land) creates a code that represents data.

CDs usually store about 650MB. This is quite a bit more than the 1.44MB that a floppy disk stores. A DVD or Digital Video Disk holds even more information than a CD, because the DVD can store information on two levels, in smaller pits or sometimes on both sides.

Recordable Optical Technologies

- CD-Recordable (CD-R)
- CD-Rewritable (CD-RW)
- Photo CD



- DVD-Recordable (DVD-R)
- DVD-RAM

CD ROM - Compact Disc Read Only Memory.

Unlike magnetic storage device which store data on multiple concentric tracks, all CD formats store data on one physical track, which spirals continuously from the center to the outer edge of the recording area. Data resides on the thin aluminium substrate immediately beneath the label. The data on the CD is recorded as a series of microscopic pits and lands physically embossed on an aluminium substrate. Optical drives use a low power laser to read data from those discs without physical contact between the head and the disc which contributes to the high reliability and permanence of storage device.

To write the data on a CD a higher power laser are used to record the data on a CD. It creates the pits and land on aluminium substrate. The data is stored permanently on the disc. These types of discs are called as WORM (Write Once Read Many). Data written to CD cannot subsequently be deleted or overwritten which can be classified as advantage or disadvantage depending upon the requirement of the user. However if the CD is partially filled then the more data can be added to it later on till it is full. CDs are usually cheap and cost effective in terms of storage capacity and transferring the data.

The CD's were further developed where the data could be deleted and re written. These types of

CDs are called as CD Rewritable. These types of discs can be used by deleting the data and making the space for new data. These CD's can be written and rewritten at least 1000 times.

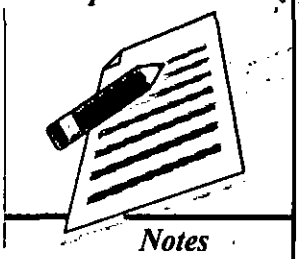
CD ROM Drive

CD ROM drives are so well standardized and have become so ubiquitous that many treat them as commodity items. Although CD ROM drives differ in reliability, which standards they support and numerous other respects, there are two important performance measures.

- Data transfer rate
- Average access

Data transfer rate: Data transfer rate means how fast the drive delivers sequential data to the interface. This rate is determined by drive rotation speed, and is rated by a number followed by _X'. All the other things equal, a 32X drive delivers data twice the speed of a 16X drive. Fast data transfer rate is most important when the drive is used to transfer the large file or many sequential smaller files. For example: Gaming video.

CD ROM drive transfers the data at some integer multiple of this basic 150 KB/s 1X rate. Rather than designating drives by actual KB/s output drive manufacturers use a multiple of the standard 1X rate. For example: a 12X drive transfer data at (12*150KB/s) 1800 KB/s and so on.



Notes

The data on a CD is saved on tracks, which spirals from the center of the CD to outer edge. The portions of the tracks towards center are shorter than those towards the edge. Moving the data under the head at a constant rate requires spinning the disc faster as the head moves from the center where there is less data per revolution to the edge where there is more data. Hence the rotation rate of the disc changes as it progresses from inner to outer portions of the disc.

CD Writers

CD recordable and CD rewritable drives are collectively called as CD writers or CD burners. They are essentially CD ROM drives with one difference. They have a more powerful laser that, in addition to reading discs, can record data to special CD media.

Pen Drives / Flash Drives

- Pen Drives / Flash Drives are flash memory storage devices.
- They are faster, portable and have a capability of storing large data.
- It consists of a small printed circuit board with a LED encased in a robust plastic
- The male type connector is used to connect to the host PC
- They are also used as MP3 players

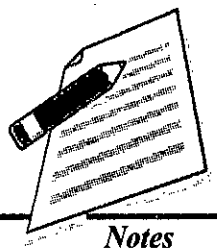
SUMMARY

Computer is an electronic machine that takes an input, processes it to produce the desired output. Hardware consists of the mechanical and electronic devices, which we can see and touch. The software consists of programs, the operating system and the data that reside in the memory and storage devices. Computer has four main components namely input devices, output devices, memory, CPU. An input device is used to get data or instructions from the user. Central Processing Unit is the brain of computer. Output devices receive information from the CPU and present it to the user in the desired form. There are two kinds of computer memory: primary and secondary. Direct access memory is the type of accessing mode in which the value to be stored in a particular memory location is obtained directly. Sequential access memory is the type of memory in which the stored data is read in sequence.

EXERGISE

MCQ

1. A computer is a/ an _____ device.
   a. Electronic
   b. Mechanical
   c. Automatic
   d. Software



2. What is the full form of RAM?
   a. Reliability, Availability and Maintainability
   b. Rarely Adequate Memory
   c. Raised Angle Marker
   d. Random Access Memory

3. CPU stands for _____.
   a. Control Processing Unit
   b. Central Processing Unit
   c. Call Processing Unit
   d. Cost Processing Unit

4. Where is data stored in a computer?
   a. Monitor
   b. Keyboard
   c. CPU
   d. Mouse

5. What is that input device used to type text and numbers on a document in the computer system?
   a. Mouse
   b. Monitor
   c. Motherboard
   d. Keyboard

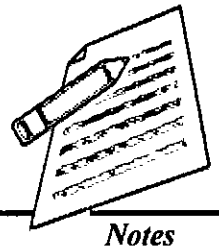
6. Name the computer part that helps a user to hear information from the system.
   a. Keyboard
   b. Speaker
   c. Mouse
   d. Monitor

7. RAM is a _____ memory.
   a. Volatile
   b. Non-volatile
   c. Short
   d. Long

8. What does ROM stand for?
   a. Rough-Order-of-Magnitude
   b. Read-Only Memory
   c. Read Only Media
   d. Read Only Member

Answers:

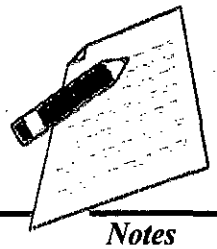
1. a) Electronic
2. d) Random Access Memory
3. b) Central Processing Unit
4. c) CPU
5. d) Keyboard
6. b) Speaker
7. a) Volatile
8. b) Read-Only Memory



Notes

Review Questions

1. What are the major components of a computer?
2. What is CPU and how does it work? Explain briefly.
3. What is a plotter and how does it work?
4. Describe the various types of printers briefly.
5. Differentiate between the following: (a) RAM and ROM (b) Volatile and non-volatile memory. (c) Sequential access and Random access
6. Explain different types of ports.



Notes

2 DATA PROCESSING CONCEPT

- Understand the concept of data.
- Discuss the concept of data processing.
- Describe the need of data processing.
- Discuss the methods of data processing.

Objective of the chapter:

The basic objective of this chapter is to through some light on the initial concepts of data processing so that the methods of data processing can be learned.

Introduction

Data processing means taking raw data (facts and figures) and processing them manually or with the help of machines to produce, organized and useful information. It is restructuring or recording of data to increase their usefulness and value for some particular purpose. Data processing can be performed

- Manually with the aid of simple tools as paper, pencil and filing cabinets
- Electro-mechanically with the aid of unit record machines
- Electronically with the aid of a computer.

Data processing is a series of operations that use information to produce a result. Common data processing operations include validation, sorting, classification, calculation, interpretation, organization and transformation of data.

Electronic Data Processing:

Electronic data processing (EDP) refers to the use of automated methods to process commercial data. Typically, this uses relatively simple, repetitive activities to process large volumes of similar information. For example: stock updates applied to an inventory, banking transactions applied to account and customer master files, booking and ticketing transactions to an airline's reservation system, billing for utility services.

The term electronic data processing dates back to the 1960s when automation began to replace manual data processing tasks. In modern times, the term tends to be associated with large scale automation of administrative tasks.

Data processing is manipulation of data by a computer. It includes the conversion of raw data to machine-readable form, flow of data through the Central Processing

Unit (CPU) and memory to output devices, and formatting or transformation of output. Any use of computers to perform defined operations on data can be included under data processing.

Data Processing in a Computer:

Computer data is information processed or stored by a computer. This information may be in the form of text documents, images, audio clips, software programs, or other types of data. Computer data may be processed by the computer's CPU and is stored in files and folders on the computer's hard disk.

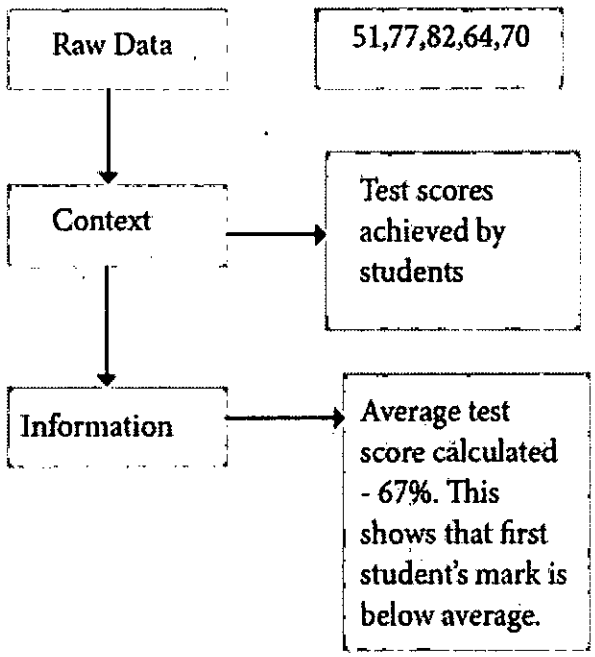


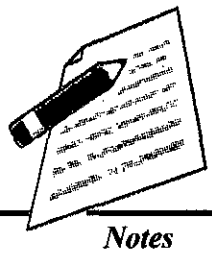
Fig.6.2. Data processing in Computer

Hence, data processing is defined as series of actions or operations that converts data into useful information. The data processing system is used to include the resource such as people procedures, and devices that are used to accomplish the processing of data for producing desirable output. Data processing is restructuring or recording of data to increase their usefulness and value for some particular purpose.

Objectives of Data Processing:

The following are the objectives of data processing:

- To provide mass storage for relevant data.
- To make easy access to the data for the user.
- To provide prompt response to user requests for data.
- To eliminate redundant data.
- To allow multiple users to be active at one time.
- To allow for growth in the data base system.
- To protect the data from harms like physical and unauthorized access.



Types of Data Processing Systems:

Types of data processing can be understood on basis of methods and technology adopted. Generally mechanical and electronic data processing is used and at times manual data processing is used. According to their working, Data processing systems can be of different types:

a. **Manual:** In manual data processing system, whole processing is done manually without use of machine or electronic device i.e. the clerical staff that performs data processing in an organized way with the goal of producing meaningful information. This method is slow and less reliable, chances of error are high and this method is very old when technical innovations were few and rare. This also makes processing expensive and requires large manpower depending on the data required to be processed. Data acquisition, filing, storage, processing, calculation, output production, all these tasks are done manually.

b. **Mechanical Data Processing:** Data processing is done by use of mechanical device or very simple electronic devices like calculator and typewriters. The advantage of this method is more reliability and saving of time as compared to manual data processing but still the output is limited. Any device which facilitates data processing can be considered under this category.

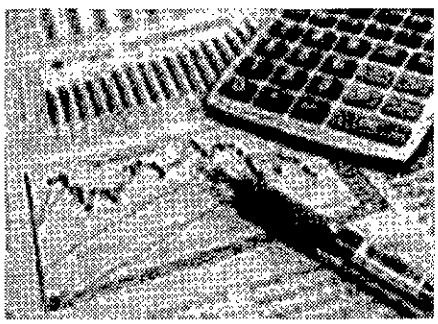


Fig.6.3. Mechanical Data Processing

c. **Electronic Data Processing (EDP):** This is the fastest and best available method with highest reliability and accuracy. With the growth of the organization, it becomes inefficient to process large amount of data with high accuracy through manual or mechanical method. EDP offers better method of data processing at a low cost as it relies on the computer and principles of electronics for processing data.

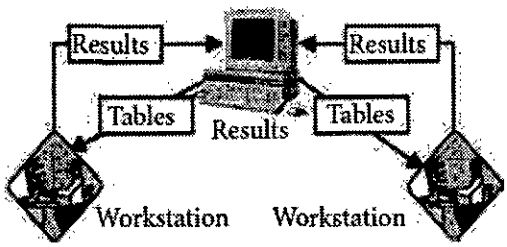
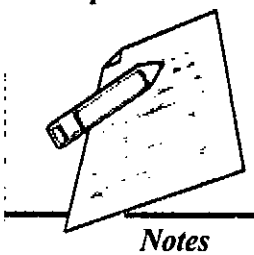


Fig.6.4. Electronic Data Processing



Notes

Modes of Data Processing:

Modes of processing data involve the following:

Interactive Computing or Interactive Processing: refers to software which accepts input from humans — for example, data or commands. Interactive software includes most popular programs, such as word processors or spread sheet applications. By comparison, non- interactive programs operate without human contact; examples of these include compilers and batch processing applications.

Transaction Processing: is information processing that is divided into individual, indivisible operations, called transactions. Each transaction must succeed or fail as a complete unit; it cannot remain in an intermediate state.

Batch Processing: is execution of a series of programs ("jobs") on a computer without human interaction. This is one of the widely used types of data processing which is also known as serial/sequential, tacked/queued of offline processing. The fundamental of this type of processing is that different jobs of different users are processed in the order received. Once the stacking of jobs is complete they are provided/sent for processing while maintaining the same order. This processing of a large volume of data helps in reducing the processing cost thus making it data processing economical.

Examples Include: Examination, payroll and billing system.

Components of EDP:

The electronic data processing cycle consists of four stages, or components.

1. Input: Input refers to all the activities associated with recording data and making it available for processing.
2. Processing: After data is recorded and converted into an appropriate form, it must be processed.
3. Output
4. Storage

Each of this stage performs specific function which is enumerated below:

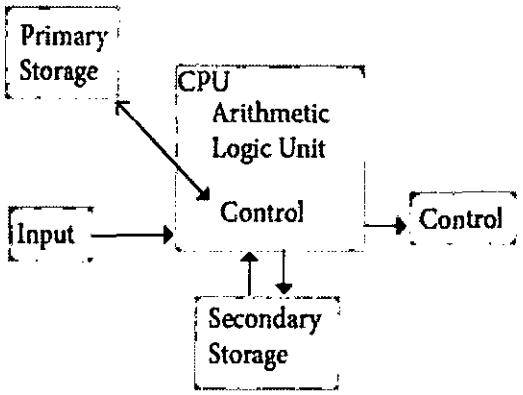
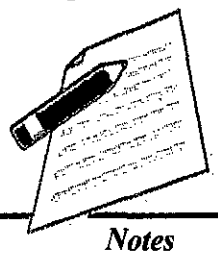


Fig.6.5. Components of EDP

Input:



A computer must receive both program statements and data to solve problems. The entry of program statements and data into a computer occurs by means of an input device such as keyboard, mouse and joystick. Regardless of the type of device used they are all instruments of interpretation and communication between people and the computer.

Central Processing Unit (CPU):

CPU is the heart of the computer which makes comparisons, performs calculations, reads, interprets and controls the execution of the instructions. It consists of three separate sub-units.

(1) **The Control Unit:** Control unit supervises the operations of the entire computer. The control unit instructs the input device when to start and stop transferring data to storage unit and it instructs the storage unit when to start and stop transferring data to output devices.

Thus the control unit does not perform the actual processing operations of the data. Rather, its function is to maintain order and direct the flow of sequence of operations and data within the computer.

(2) **The Arithmetic / Logic Unit:** Arithmetic and logic unit performs mathematical calculations, compares numeric and non-numeric values and makes decisions. The data flows between this unit and the storage unit during processing.

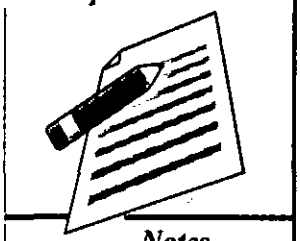
Storage:

Storage consists of primary and secondary storage. Primary storage of the computer consists of the devices used to store the information which will be used during the computations. The storage section of the computer is also used to hold both intermediate and final results as the computer proceeds through the program. Common storage devices are RAM. Since the primary storage capacity of computers is limited it is not always possible to hold a large volume of data and instructions in the primary storage. Hence it becomes necessary to have secondary or auxiliary storage for holding data and programs not currently in use. The various secondary storage devices are CD, DVD, USB flash drive (Pendrive), Hard disk etc.

Output Devices:

Output devices are used to record the results obtained by the computer and present them to the outside world. They take information in machine coded form from storage unit and convert them typically into a form that can be used i.e. printed forms. The most commonly used output devices are printers, visual display unit, monitor etc.

Hardware: Hardware is the physical aspect of computers. Computer hardware is the collection of physical parts of a computer system. This includes the computer



case, monitor, keyboard, and mouse. It also includes all the parts inside the computer case, such as the hard disk drive, motherboard, video card, and many others. These are called hardware since these components can be seen and touched by the user.

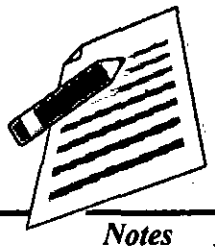
Software: The sets of computer program instructions that direct the operation of the hardware are called software. A complete set of instructions to execute a related set of tasks is a program. These are called software because the programmes cannot be seen or touched. Software instructions are termed as code. Software can be divided into two major categories:

i. **System Software:** System software means the operating system. It is the collection of programs that directs a computer to perform functions associated with controlling and directing computer hardware and also determines how application software is run.

ii. **Application Software:** Application software refers to the computer programs written for an individual application such as payroll processing or personnel skill analysis. They generally require system software in their execution. For example, the application program may specify reading data from a record stored on a disk; the operating system provides the instructions to manage the physical reading of the record from disk storage.

SUMMARY

Data processing means taking raw data (facts and figures) and processing them manually or with the help of machines to produce, organized and useful information. It is restructuring or recording of data to increase their usefulness and value for some particular purpose. Data processing is a series of operations that use information to produce a result. Common data processing operations include validation, sorting, classification, calculation, interpretation, organization and transformation of data. Electronic data processing (EDP) refers to the use of automated methods to process commercial data. Typically, this uses relatively simple, repetitive activities to process large volumes of similar information. For example: stock updates applied to an inventory, banking transactions applied to account and customer master files, booking and ticketing transactions to an airline's reservation system, billing for utility services. The term electronic data processing dates back to the 1960s when automation began to replace manual data processing tasks. In modern times, the term tends to be associated with large scale automation of administrative tasks. Data processing is manipulation of data by a computer. It includes the conversion of raw data to machine-readable form, flow of data through the Central Processing Unit (CPU) and memory to output devices, and formatting or transformation of output. Any use of computers to perform defined operations on data can be included under data processing.



EXERGISE

MCQ

1. Qualitative data analysis is still a relatively new and rapidly developing branch of research methodology.
   a. True b. False
   Answer: a
2. The process of marking segments of data with symbols, descriptive words, or category names is known as _____.
   a. Concurring b. Coding
   c. Colouring d. Segmenting
   Answer: b
3. What is the cyclical process of collecting and analysing data during a single research study called?
   a. Interim analysis b. Inter analysis
   c. Inter-item analysis d. Constant analysis
   Answer: a
4. The process of quantifying data is referred to as _____.
   a. Typology b. Diagramming
   c. Enumeration d. Coding
   Answer: c
5. An advantage of using computer programs for qualitative data is that they _____.
   a. Can reduce time required to analyse data (i.e., after the data are transcribed)
   b. Help in storing and organising data
   c. Make many procedures available that are rarely done by hand due to time constraints
   d. All of the above
   Answer: d

Review Questions

1. Define data processing?
2. Discuss the components of data processing?
3. Describe the methods of data processing?
4. Explain the need of data processing?

Notes

3 COMPUTER SOFTWARE

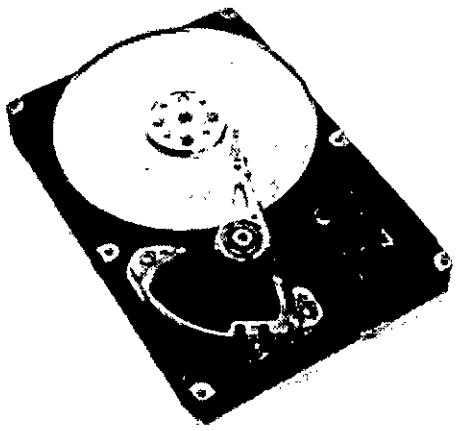
- Understand the concept of software.
- Discuss the types of software.
- Describe the concept of application software.
- Discuss the concept of system software.

Objective of the chapter:

The basic objective of this chapter is to through some light on the initial concepts of software so that the types and applications of computer software can be learned.

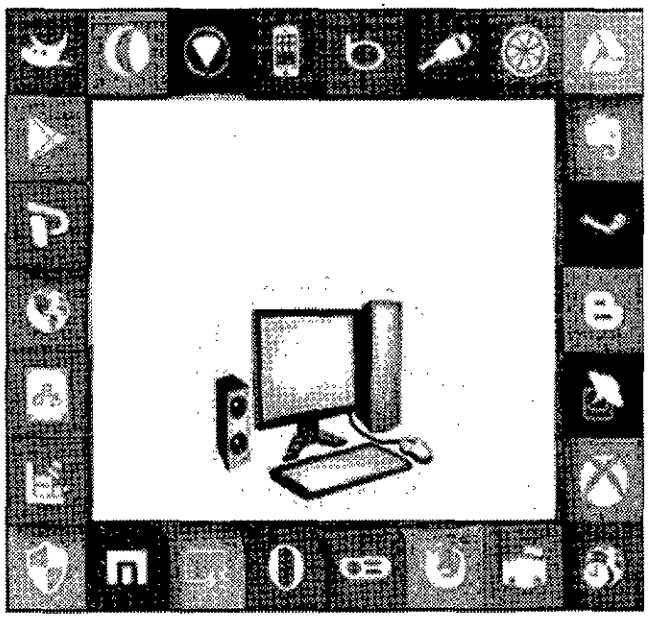
Introduction

Computer is a device comprising both hardware and software. The functions of hardware and software combines together to make the Computer functional. A hardware device helps to enter input information. The software processes the input data and gives the output in the monitor, a hardware device. Thus a computer is like a human body, where human body is the hardware and soul is the software.



Software

Hardware is lifeless without software in a computer. Software is programmed and coded applications to process the input information. The software processes the data by converting the input information into coding or programmed language. Touching and feeling the software is not possible but we can see the functions of the software in the form of output.



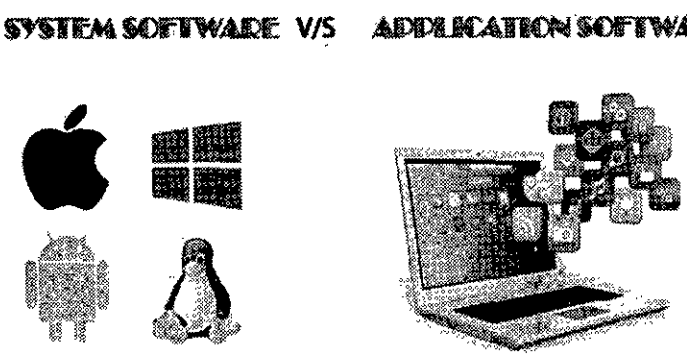


Figure 9.2 System and Application Software

Types of Software

The software is divided into two types based on the process. They are:

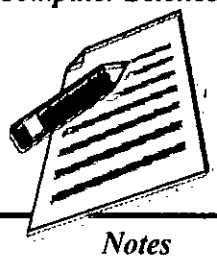
1. System software (Operating System)
2. Application software

1. System software

System software (Operating system) is software that makes the hardware devices process the data fed by the user and to display the result on the output devices like Monitor. Without the operating system, computer cannot function on its own. Some of the popular operating system are Linux, Windows, Mac, Android etc.

2. Application Software

Application software is a program or a group of programs designed for the benefit of end user to work on computer. The application programs can be installed in the hard disk for the usage on a particular computer. This type of application program



completes one or more than one works of the end user. The following are the examples of application program: Video player, Audio player, Word processing software, drawing tools, Editing software, etc.

System and Application Software types

The operating system and application software are available in two forms. They are:

1. Free and Open source
2. Paid and Proprietary Software

1. Free and Open source

Free and open software is available at free of cost and can be shared to many end users. Free software is editable and customizable by the user and this leads to updation or development of new software. Examples of Free and Open source software are: LINUX, Open office, Geogebra etc.

2. Paid and Proprietary Software

There are softwares that need a license to use it. They have to be paid for using either permanently or temporarily. The license of the software would not be provided unless it is purchased. Similarly the end users are legally prohibited to steal the software program or to use the pirated version of the Paid and Proprietary Software. Some of the examples of Paid and Proprietary Software are: Windows, Microsoft office, Adobe Photoshop, etc.

PROGRAMMING LANGUAGE

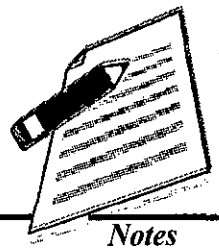
A programming language is a set of symbols and rules for instructing a computer to perform specific tasks. The programmers have to follow all the specified rules before writing program using programming language. The user has to communicate with the computer using language which it can understand.

Types of programming language

1. Machine language
2. Assembly language
3. High level language

Machine language:

The computer can understand only machine language which uses 0's and 1's. In machine language the different instructions are formed by taking different combinations of 0's and 1's.



Advantages:

Translation free:

Machine language is the only language which the computer understands. For executing any program written in any programming language, the conversion to machine language is necessary. The program written in machine language can be executed directly on computer. In this case any conversion process is not required.

High speed

The machine language program is translation free. Since the conversion time is saved, the execution of machine language program is extremely fast.

Disadvantage:

1. It is hard to find errors in a program written in the machine language.
2. Writhing program in machine language is a time consuming process.

Machine dependent: According to architecture used, the computer differs from each other. So machine language differs from computer to computer. So a program developed for a particular type of computer may not run on other type of computer.

Assembly language:

To overcome the issues in programming language and make the programming process easier, an assembly language is developed which is logically equivalent to machine language but it is easier for people to read, write and understand.

Assembly language is symbolic representation of machine language. Assembly languages are symbolic programming language that uses symbolic notation to represent machine language instructions. They are called low level language because they are so closely related to the machines.

Assembler

Assembler is the program which translates assembly language instruction in to a machine language.

1. Easy to understand and use.
2. It is easy to locate and correct errors.

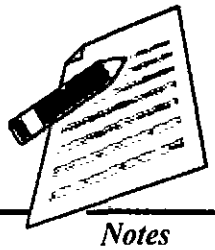
Disadvantage

Machine dependent

The assembly language program which can be executed on the machine depends on the architecture of that computer.

Hard to learn

It is machine dependent, so the programmer should have the hardware knowledge to create applications using assembly language.



Less efficient

1. Execution time of assembly language program is more than machine language program.
2. Because assembler is needed to convert from assembly language to machine language.

High level language

High level language contains English words and symbols. The specified rules are to be followed while writing program in high level language. The interpreter or compilers are used for converting these programs in to machine readable form.

Translating high level language to machine language

The programs that translate high level language in to machine language are called interpreter or compiler.

Compiler:

A compiler is a program which translates the source code written in a high level language in to object code which is in machine language program. Compiler reads the whole program written in high level language and translates it to machine language. If any error is found it display error message on the screen.

Interpreter

Interpreter translates the high level language program in line by line manner. The interpreter translates a high level language statement in a source program to a machine code and executes it immediately before translating the next statement. When an error is found the execution of the program is halted and error message is displayed on the screen.

Advantages

Readability

High level language is closer to natural language so they are easier to learn and understand

Machine independent

High level language program have the advantage of being portable between machines.

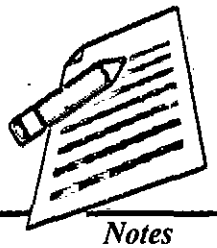
Easy debugging

Easy to find and correct error in high level language

Disadvantages

Less efficient

The translation process increases the execution time of the program. Programs in high level language require more memory and take more execution time to execute.



SUMMARY

A complete set of instructions written to solve a problem on a computer is called software. i.e., software refers to the set of computer programs that cause the hardware (computer system) to function in the desired manner. Computer software is normally classified into two broad categories (i) system software and (ii) application software. System software includes general programs written for a computer. Application software is written to perform a specific task or process. Compiler reads the entire program for compilation. Interpreter reads single statement at a time for interpretation. Programming languages can be classified as machine language, assembly language and high-level language. Machine language and assembly language programs are difficult to write and read. High level languages are easy to learn and write. Compiler and interpreter convert the program written in high level language into machine language code. A translation program is run to convert the high-level language program, which is called the source code, into a machine language code. This translation process is called compilation.

EXERCISE

MCQ

1. The physical devices of a computer:
   a) Software
   b) Package
   c) Hardware
   d) System Software

   Answer: c

2. Software Package is a group of programs that solve multiple problems.
   a) True
   b) False

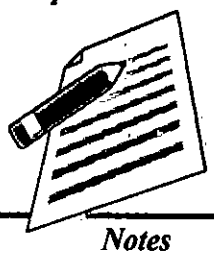
   Answer: b

3. _____ refer to renewing or changing components like increasing the main memory, or hard disk capacities, or adding speakers, or modems, etc.
   a) Grades
   b) Prosody
   c) Synthesis
   d) Upgrades

   Answer: d

4. Which of the following is designed to control the operations of a computer?
   a) Application Software
   b) System Software
   c) Utility Software
   d) User

   Answer: b



Notes

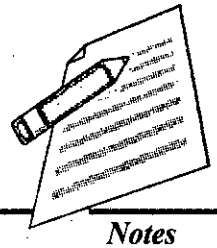
5. Which of the following is not an example of system software?
   a) Language Translator
   b) Utility Software
   c) Communication Software
   d) Word Processors

   Answer: d

Review Quotations

1. Write any one difference between hardware and software.
2. What are the advantages and disadvantages of machine language?
3. What is the difference between source program and object program?
4. Explain assembly language. What are its advantages over machine language?
5. Define operating system. Write down its various functions.
6. Describe briefly about application software.

Describe by



Notes

4 OPERATING SYSTEM

- Understand the concept of operating system.
- Discuss the types of operating system.
- Describe the uses of operating system.
- Discuss the applications of operating system.

Objective of the chapter:

The basic objective of this chapter is to through some light on the initial concepts of operating system so that the types and applications of operating system can be learned.

Introduction

An Operating System (OS) is a system software which serves as an interface between a user and a computer. This controls input, output and other peripheral devices such as disk drives, printers and electronic gadgets. The functions of an Operating System include file management, memory management, process management and device management and many more.

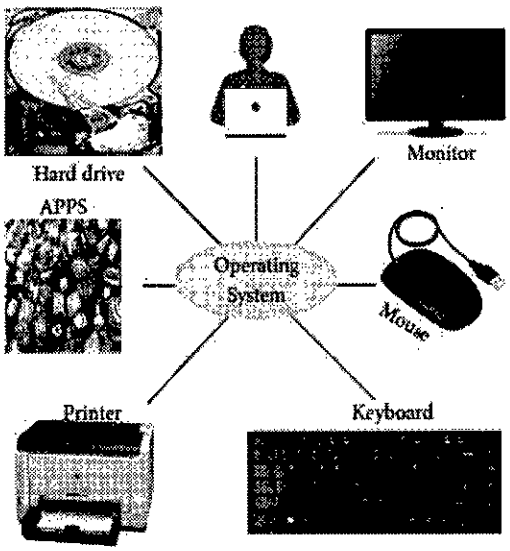
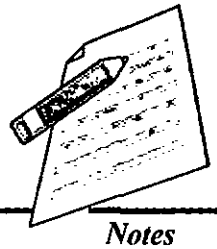


Figure: 4.1 Operating System

Without an Operating System, a computer cannot effectively manage all the resources. When a computer is switched on, the operating system is loaded in to the memory automatically. A user cannot communicate directly with the computer hardware, unless an operating system is loaded.



Notes

Some of the popular Operating Systems used in personal computers and laptops are **Windows, UNIX** and **Linux.** The mobile devices mostly use Android and ioS as mobile **OS.**

Need for Operating System

Operating System has become essential to enable the users to design applications without the knowledge of the computer's internal structure of hardware. Operating System manages all the Software and Hardware. Most of the time there are many different computer programmes running at the same time, they all need to access the Computers, CPU, Memory and Storage. The need of Operating System is basically - an interface between the user and hardware.

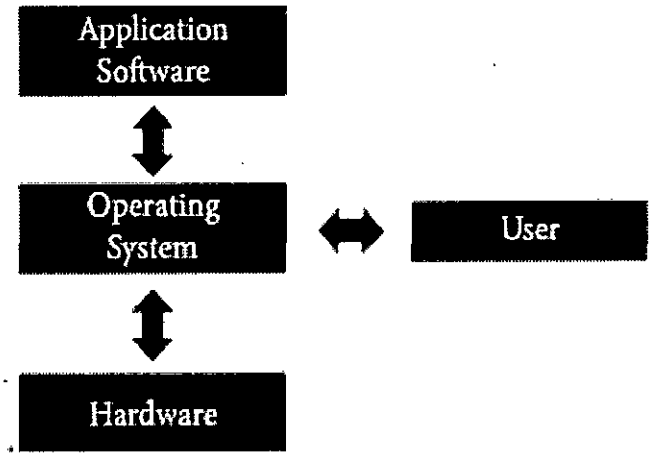


Figure: 4.2 Interaction of Operating system and user

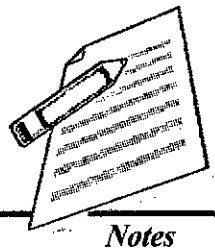
Operating System works as translator, while it translates the user request into machine language (Binary language), processes it and then sends it back to Operating System. Operating System converts processed information into user readable form

Uses of Operating Systems

The following are few uses of Operating System

The main use of Operating System is

- To ensure that a computer can be used to extract what the user wants it do.
- Easy interaction between the users and computers.
- Starting computer operation automatically when power is turned on (Booting).
- Controlling Input and Output Devices
- Manage the utilization of main memory.
- Providing security to user programs.



Types of Operating System

Operating System is classified into the following types depending on their processing capabilities.

1. Single User Operating Systems

An operating system allows only a single user to perform a task at a time. It is called as a Single user and single Task operating system. For a user, a task is a function such as printing a document, writing a file to disk, editing a file or downloading a file etc. MS-DOS is an example for a single user and single task Operating System.

2. Multi-user Operating Systems

It is used in computers and laptops that allow same data and applications to be accessed by multiple users at the same time. The users can also communicate with each other. Windows, Linux and UNIX are examples for multi-user Operating System.

Key features of the Operating System

The various key features are given below

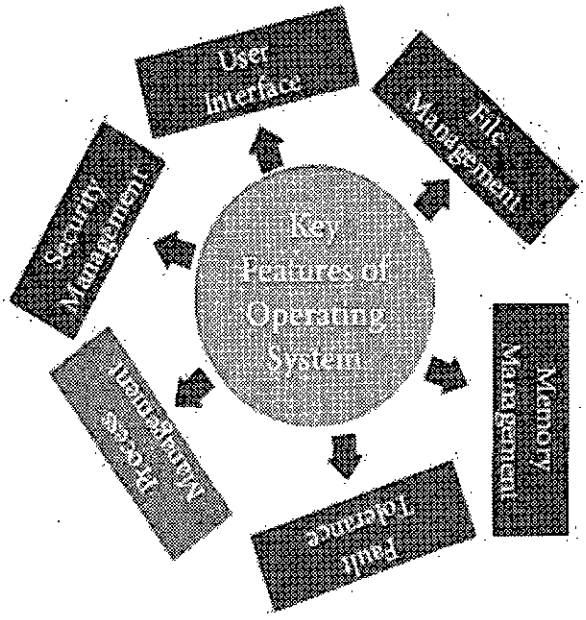
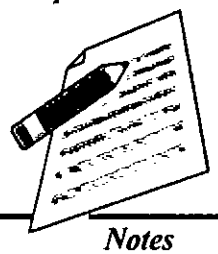


Figure: 4.3 Key Features of the Operating System

1. User Interface (UI)

User interface is one of the significant features in Operating System. The only way that user can make interaction with a computer. If the computer interface is not user-friendly, the user slowly reduces the computer usage from their normal life. This is the main reason for the key success of GUI (Graphical User Interface) based Operating



System. The GUI is a window based system with a pointing device to direct I/O, choose from menus; make selections and a keyboard to enter text. Its vibrant colours attract the user very easily. Beginners are impressed by the help and pop up window message boxes. Icons are playing vital role of the particular application.

Now Linux distribution is also available as GUI based Operating System. The following points are considered when User Interface is designed for an application:

- The user interface should enable the user to retain this expertise for a longer time.
- The user interface should also satisfy the customer based on their needs.
- The user interface should save user's precious time. Create graphical elements like Menus, Window, Tabs, Icons and reduce typing work will be an added advantage of the Operating System.
- The ultimate aim of any product is to satisfy the customer. The User Interface is also to satisfy the customer.
- The user interface should reduce number of errors committed by the user with a little practice the user should be in a position to avoid errors (Error Log File)

2. Memory Management

Memory Management is the process of controlling and coordinating computer's main memory and assigning memory block (space) to various running programs to optimize overall computer performance. The Memory management involves the allocation of specific memory blocks to individual programs based on user demands. At the application level, memory management ensures the availability of adequate memory for each running program at all times.

The objective of Memory Management process is to improve both the utilization of the CPU and the speed of the computer's response to its users via main memory. For these reasons the computers must keep several programs in main memory that associates with many different Memory Management schemes.

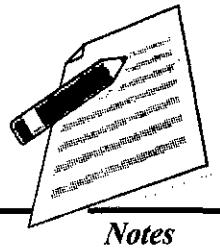
The Operating System is responsible for the following activities in connection with memory management:

- Keeping track of which portion of memory are currently being used and who is using them.
- Determining which processes (or parts of processes) and data to move in and out of memory.
- Allocation and de-allocation of memory blocks as needed by the program in main memory. (Garbage Collection)

3. Process management

Process management is function that includes creating and deleting processes and providing mechanisms for processes to communicate and synchronize with each other.

A process is the unit of work (program) in a computer. A word-processing program being run by an individual user on a computer is a process. A system task, such as sending output to a printer or screen, can also be called as a Process.



A computer consists of a collection of processes; they are classified as two categories:

- Operating System processes which is executed by system code
- User Processes which is execute by user code

All these processes can potentially execute concurrently on a single CPU.

A process needs certain resources including CPU time, memory, files and I/O devices to finish its task.

The Operating System is responsible for the following activities associated with the process management:

- Scheduling processes and threads on the CPUs
- Creating and deleting both user and system processes
- Suspending and resuming processes
- Providing mechanisms for process synchronization
- Providing mechanisms for process communication

The following algorithms are mainly used to allocate the job (process) to the processor.

1. FIFO 2. SJF 3. Round Robin 4. Based on Priority

FIFO (First In First Out) Scheduling:

This algorithm is based on queuing technique. Assume that a student is standing in a queue to get grade sheet from his/her teacher. The other student who stands first in the queue gets his/ her grade sheet first and leaves from the queue. Followed by the next student in the queue gets it collected and so on. This is the basic logic of the FIFO algorithm.

Technically, the process that enters the queue first is executed first by the CPU, followed by the next and so on. The processes are executed in the order of the queue.

SJF (Shortest Job First) Scheduling:

This algorithm works based on the size of the job being executed by the CPU.

Consider two jobs A and B.

1) A = 6 kilo bytes 2) B = 9 kilo bytes

First the job "A" will be assigned and then job "B" gets its turn.

Round Robin Scheduling

The Round Robin (RR) scheduling algorithm is designed especially for time sharing systems. Jobs (processes) are assigned and processor time in a circular method. For example take three jobs A, B, C. First the job A is assigned to CPU then job B and job C and then again A, B and C and so on.

Based On Priority

The given job (process) is assigned based on a Priority. The job which has higher priority is more important than other jobs. Take two jobs A and B. Let the priority of A be 5 and priority B be 7.

Job B is assigned to the processor before job A.

4. Security Management

The major challenge in computer and software industry is to protect user's legitimate data from hackers. The Operating System provides three levels of securities to the user end. They are

- File access level
- System level
- Network level

In order to access the files created by other people, you should have the access permission. Permissions can either be granted by the creator of the file or by the administrator of the system.

System level security is offered by the password in a multi-user environment.

Both windows and Linux offer the password facility.

Network security is an indefinable one. So people from all over the world try to provide such a security.

All the above levels of security features are provided only by the Operating System.

5. Fault Tolerance

The Operating Systems should be robust. When there is a fault, the Operating System should not crash, instead the Operating System have fault tolerance capabilities and retain the existing state of system.

6. File Management

File management is an important function of OS which handles the data storage techniques. The operating System manages the files, folders and directory systems on a computer. Any type of data in a computer is stored in the form of files and directories/ folders through File Allocation Table (FAT). The FAT stores general information about files like filename, type (text or binary), size, starting address and access mode (sequential/indexed / indexed-sequential/ direct/relative). The file manager of the operating system helps to create, edit, copy, allocate memory to the files and also updates the FAT. The OS also takes care of the files that are opened with proper access rights to read or edit them. There are few other file management techniques available like Next Generation File System (NTFS) and ext2(Linux).

7. Multi-Processing

This is a one of the features of Operating System. It has two or more processors for a single running process (job). Processing takes place in parallel is known as parallel processing. Each processor works on different parts of the same task or on two or more different tasks. Since the execution takes place in parallel, this feature is used for high speed execution which increases the power of computing.

8. Time-sharing

This is a one of the features of Operating Systems. It allows execution of multiple tasks or processes concurrently. For each task a fixed time is allocated. This division of time is called Time- sharing. The processor switches rapidly between various processes after a time is elapsed or the process is completed.

For example assume that there are three processes called P1, P2, P3 and time allocated for each process 30, 40, 50 minutes respectively. If the process P1 completes within 20 minutes then processor takes the next process P2 for the execution. If the process P2 could not complete within 40 minutes, then the current process P2 will be paused and switch over to the next process P3.

9. Distributed Operating Systems

This feature takes care of the data and application that are stored and processed on multiple physical locations across the world over the digital network (internet/ intranet). The Distributed Operating System is used to access shared data and files that reside in any machine around the world. The user can handle the data from different locations. The users can access as if it is available on their own computer.

The advantages of distributed Operating System are as follows:

1. A user at one location can make use of all the resources available at another location over the network.
2. Many computer resources can be added easily in the network
3. improves the interaction with the customers and clients.
4. Reduces the load on the host computer.

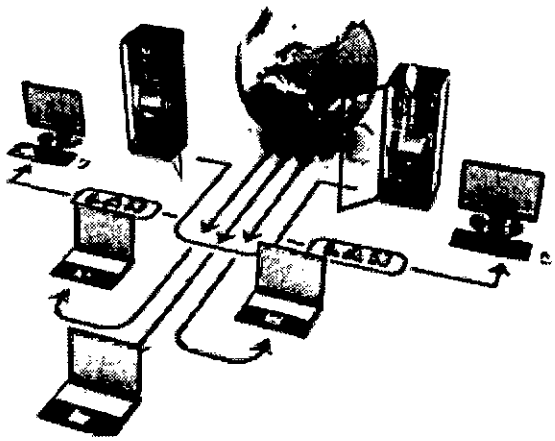
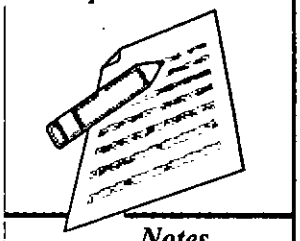


Figure: 4.4 Distributed Operating Systems

SUMMARY

An Operating System (OS) is an interface between users and computer hardware. It provides users an environment in which a user can execute programs conveniently and efficiently. An operating system controls the allocation of resources and services

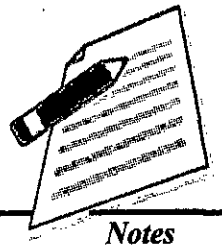


such as memory, processors, devices and information. A file is a collection of related information that is recorded on secondary storage device. Multiprogramming is the ability of an operating system to execute more than one program on a single processor machine. More than one task/program/ job/process can reside into the main memory at one point of time. A multiuser operating system is a computer operating system (OS) that allows multiple users on different computers or terminals to access a single system with one OS on it. Multitasking is the ability of an operating system to execute more than one task simultaneously on a single processor machine. Though we say so but in reality no two tasks on a single processor machine can be executed at the same time. Actually CPU switches from one task to the next task so quickly that appears as if all the tasks are executing at the same time.

EXERCISE

MCQ

1. Operating system is a
   A) Application Software B) Hardware
   C) System Software D) Component
2. Identify the usage of Operating Systems
   A) Easy interaction between the human and computer
   B) Controlling input & output Devices
   C) Managing use of main memory
   D) All the above
3. Which of the following is not a function of an Operating System?
   A) Process Management B) Memory Management
   C) Security management **D) Complier Environment**
4. Which of the following OS is a commercially licensed Operating system?
   A) Windows B) UBUNTU
   C) FEDORA D) REDHAT
5. Which of the following Operating systems support Mobile Devices?
   A) Windows 7 B) Linux
   C) BOSS **D) iOS**
6. File Management manages
   A) Files B) Folders
   C) Directory systems **D) All the Above**
7. Interactive Operating System provides
   A) Graphics User Interface (GUI)
   B) Data Distribution
   C) Security Management
   D) Real Time Processing

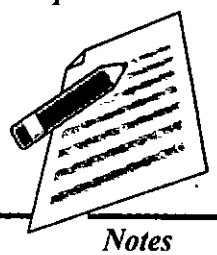


8. Android is a
   - **A) Mobile Operating system**
   - B) Open Source
   - C) Developed by Google
   - D) All the above

9. Which of the following refers to Android operating system's version?
   - **A) JELLY BEAN**
   - B) UBUNTU
   - C) OS/2
   - D) MITTIKA

Review Questions

1. State all the important functions of operating system. Explain with the help of diagrams.
2. Explain all the functions of operating system.
3. What are the advantages and disadvantages of GUI based operating system?
4. Explain different types of smart phone operating systems.
5. What do you understand by open source software? Give some examples.
6. What is the difference between Windows and Linux operating systems?



Notes

5 DATA COMMUNICATION AND NETWORKING

- Understand the concept of data communication.
- Discuss the types of data communication.
- Describe the concept of network.
- Discuss the types of network.

Objective of the chapter:

The basic objective of this chapter is to through some light on the initial concepts of data communication and network so that the types and applications of networking can be learned.

Introduction

Data communications are the exchange of data between two devices via some form of transmission medium such as a wire cable.

1. Characteristics:

The effectiveness of a data communications system depends on four fundamental characteristics:

a. Delivery:

The system must deliver data to the correct destination. Data must be received by the intended device or user and only by that device or user.

b. Accuracy:

The system must deliver data accurately.

c. Timeliness:

The system must deliver data in a timely manner. In the case of video and audio, timely delivery means delivering the data as they are produced. In the same order, that they are produced, and without significant delay. This kind of delivery is called real-time transmission.

d. Jitter:

Jitter refers to the variation in the packet arrival time. It is the uneven delay in the delivery of audio or video packet.

2. Components:

A data communication system has five components.

a. Message:

The message is the information to be communicated. Popular forms of information include text, numbers, pictures audio and video.

b. Sender:

The sender is the device that sends the data message. It can be a computer, workstation, telephone handset, video camera, and so on.

c. Receiver:

The receiver is the device that receives the message. It can be a computer, workstation, telephone handset, television and so on.

d. Transmission medium:

The transmission medium is the physical path by which a message travels from sender to receiver. Ex. Twisted pair wire, coaxial cable, fibre optic cable and radio waves.

e. Protocol:

A protocol is the set of rules that governs data communications. It represents an agreement between the communicating devices.

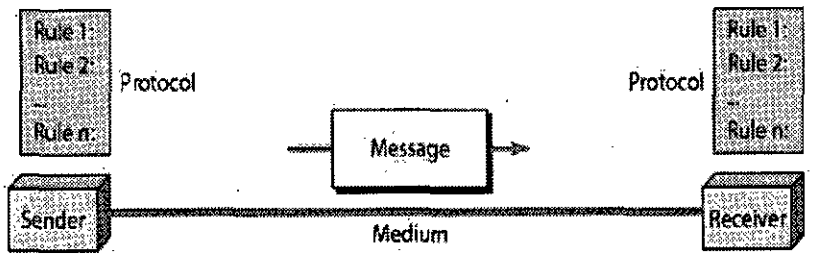


Figure 1.1 Components of a data communication system

3. Data Representation:

Information today comes in different forms such as text, numbers, images, audio, and Video.

a. Text:

Text is represented as a bit pattern, a sequence of bits (Os or 1s). Different sets of bit patterns have been designed to represent text symbols. Each set is called a code, and the process of representing symbols is called coding.

b. Numbers:

Numbers are also represented by bit patterns. The number is directly converted to a binary number to simplify mathematical operations.

c. Images:

Images are also represented by bit patterns. An image is composed of a matrix of pixels (picture elements), where each pixel is a small dot. The size of the pixel depends on the resolution.

d. Audio:

Audio refers to the recording or broadcasting of sound or music. It is continuous, not discrete.

e. Video:

Video refers to the recording or broadcasting of a picture or movie. Video can either be produced as a continuous entity (e.g., by a TV camera), or it can be a combination of images, each a discrete entity, arranged to convey the idea of motion.

4. Data Flow

Communication between two devices can be simplex, half-duplex, or full-duplex.

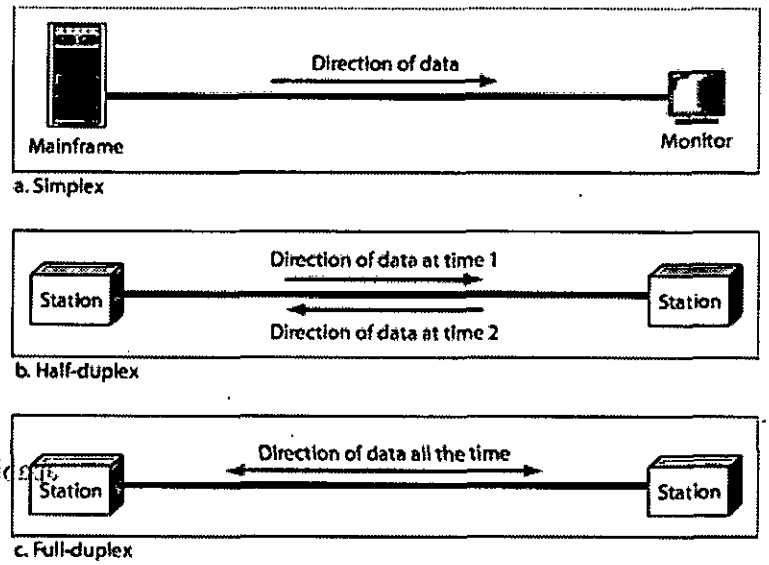
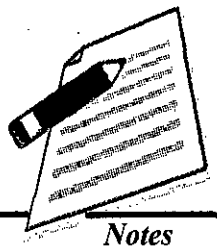


Figure 1.2 Data flow (simplex, half-duplex and full-duplex)

a. Simplex:

In simplex mode, the communication is unidirectional, as on a one-way street. Only one of the two devices on a link can transmit; the other can only receive (see Figure 1.2a). Keyboards and traditional monitors are examples of simplex devices.



b. Half-Duplex:

In half-duplex mode, each station can both transmit and receive, but not at the same time. When one device is sending, the other can only receive, and vice versa (see Figure 1.2b). Walkie-talkies and CB (citizens band) radios are both half-duplex systems.

c. Full-Duplex:

In full-duplex mode (also called duplex), both stations can transmit and receive simultaneously (see Figure 1.2c). One common example of full-duplex communication is the telephone network.

Networks:

A **network** is a set of devices (often referred to as nodes) connected by communication links. A node can be a computer, printer, or any other device capable of sending and/or receiving data generated by other nodes on the network.

1. Distributed Processing:

Most networks use distributed processing, in which a task is divided among multiple computers. Instead of one single large machine being responsible for all aspects of process, separate computers (usually a personal computer or workstation) handle a subset.

2. Network Criteria:

A network must be able to meet a certain number of criteria. The most important of these are performance, reliability, and security.

a. **Performance:** Performance can be measured in many ways, including transit time and response time.

   Transit time is the amount of time required for a message to travel from one device to another. Response time is the elapsed time between an inquiry and a response.

   The performance of a network depends on a number of factors, including the number of users, the type of transmission medium, the capabilities of the connected hardware, and the efficiency of the software.

b. **Reliability:** In addition to accuracy of delivery, network reliability is measured by the frequency of failure, the time it takes a link to recover from a failure, and the network's robustness in a disaster.

c. **Security:** Network security issues include protecting data from unauthorized access, protecting data from damage and development, and implementing policies and procedures for recovery from breaches and data losses.

3. Physical Structures:

Type of Connection: A network is two or more devices connected through links. A link is a communications pathway that transfers data from one device to another.

There are two possible types of connections: point-to-point and multipoint.

a. **Point-to-Point:** A point-to-point connection provides a dedicated link between two devices. The entire capacity of the link is reserved for transmission between those two devices (see Figure 1.3a).

b. **Multipoint:** A multipoint (also called multi drop) connection is one in which more than two specific devices share a single link (see Figure 1.3b). In a multipoint environment, the capacity of the channel is shared, either spatially or temporally. If several devices can use the link simultaneously, it is a spatially shared connection. If users must take turns, it is a timeshared connection.

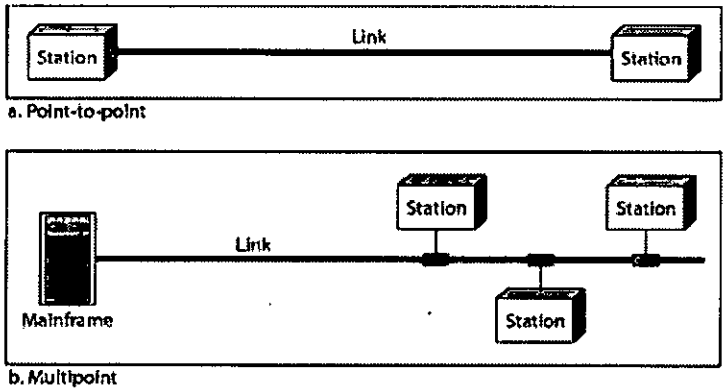


Figure 1.3 Types of connections: point-to-point and multipoint

Physical Topology:

The term physical topology refers to the way in which a network is laid out physically. Two or more devices connect to a link; two or more links form a topology. The topology of a network is the geometric representation of the relationship of all the links and linking devices (usually called nodes) to one another.

There are four basic topologies possible: mesh, star, bus, and ring (see Figure 1.4)

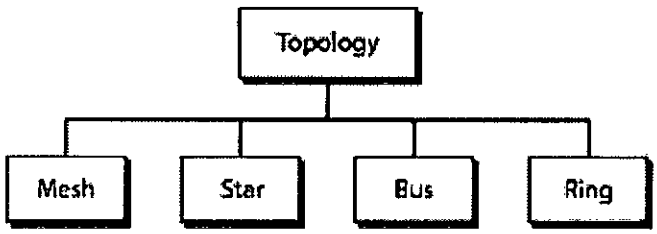
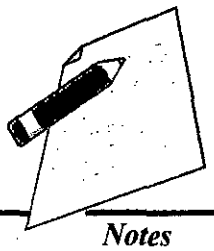


Figure 1.4 Categories of Topology

a. **Mesh:** In a mesh topology, every device has a dedicated point-to-point link to every other device. A fully connected mesh network with n nodes has n (n -1) /2 physical channels.



To accommodate that many links, every device on the network must have n – 1 input/output (I/O) ports (see Figure 1.5) to be connected to the other n – 1 stations.

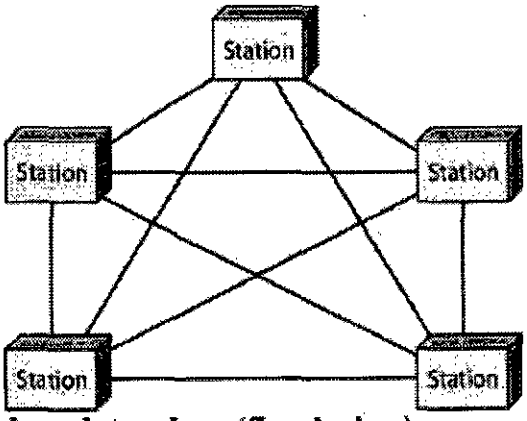


Figure 1.5 A fully connected mesh topology (five devices)

Advantages:

1. The use of dedicated links guarantees that each connection can carry its own data load, thus eliminating the traffic problems that can occur when links must be shared by multiple devices.
2. A mesh topology is robust. If one link becomes unusable, it does not incapacitate the entire system.
3. Privacy or security.
4. Point-to-point links make fault identification and fault isolation easy.

Disadvantages:

1. Every device must be connected to every other device, installation and reconnection are difficult.
2. The sheer bulk of the wiring can be greater than the available space (in walls, ceilings, or floors) can accommodate.
3. The hardware required to connect each link (I/O ports and cable) can be prohibitively expensive.

One practical example of a mesh topology is the connection of telephone regional offices in which each regional office needs to be connected to every other regional office.

b. Star Topology: In a star topology, each device has a dedicated point-to-point link only to a central controller, usually called a hub. The devices are not directly linked to one another. Unlike a mesh topology, a star topology does not allow direct traffic between devices. The controller acts as an exchange: If one device wants to send data to another, it sends the data to the controller, which then relays the data to the other connected device (see Figure 1.6).

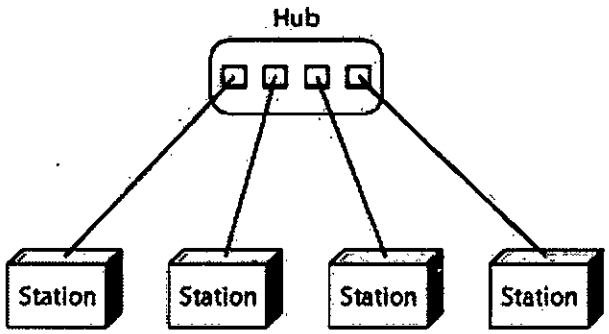


Figure 1.6 A star topology connecting four stations

Advantages:

1. Less expensive than a mesh topology.
2. Easy to install and reconfigure.
3. Far less cabling needs to be housed, and additions, moves, and deletions involve only one connection: between that device and the hub.
4. It includes Robustness. If one link fails, only that link is affected. All other links remain active.

Disadvantages:

1. It is the dependency of the whole topology on one single point, the hub. If the hub goes down, the whole system is dead.
2. A star requires far less cable than a mesh; each node must be linked to a central hub. For this reason, often more cabling is required in a star than in some other topologies (such as ring or bus).

   The star topology is used in local-area networks (LANs). High-speed LANs often use a star topology with a central hub.

c. **Bus Topology:** A bus topology is multipoint. One long cable acts as a backbone to link all the devices in a network (see Figure 1.7). Nodes are connected to the bus cable by drop lines and taps.

A drop line is a connection running between the device and the main cable.

A tap is a connector that either splices into the main cable or punctures the sheathing of a cable to create a contact with the metallic core. As a signal travels along the backbone, some of its energy is transformed into heat. Therefore, it becomes weaker and weaker as it travels farther and farther. For this reason there is a limit on the number of taps a bus can support and on the distance between those taps.

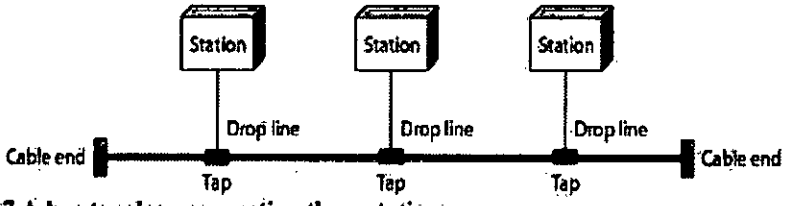


Figure 1.7 A bus topology connecting three stations

Advantages:

1. Ease of installation.
2. Less cabling

Disadvantages:

1. Difficult reconfiguration and fault isolation.
2. Difficult to add new devices.
3. Signal reflection at top can degradation in quality.
4. If any fault in backbone can stops all transmission.

Ethernet LANs can use a bus topology, but they are less popular now.

d. Ring Topology: In a ring topology, each device has a dedicated point-to-point connection with only the two devices on either side of it. A signal is passed along the ring in one direction, from device to device, until it reaches its destination. Each device in the ring incorporates a repeater. When a device receives a signal intended for another device, its repeater regenerates the bits and passes along them (see Figure 1.8).

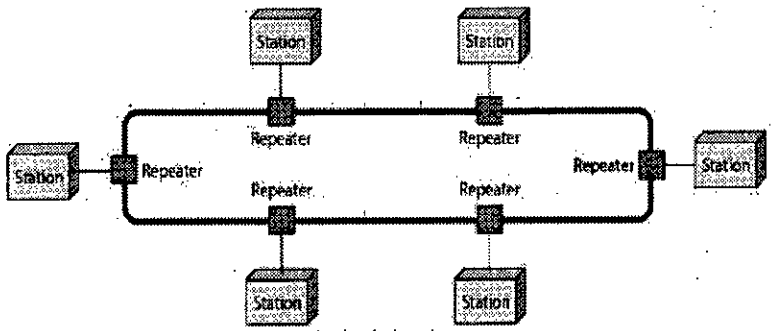


Figure 1.8 A ring topology connecting six stations

Advantages:

1. Easy to install.
2. Easy to reconfigure.
3. Fault identification is easy.

Disadvantages:

1. Unidirectional traffic.
2. Break in a single ring can break entire network.

Ring topologies are found in some office buildings or school campuses. Today high speed LANs made this topology less popular.

e. Hybrid Topology: A network can be hybrid. For example, we can have a main star topology with each branch connecting several stations in a bus topology as shown in Figure 1.9.

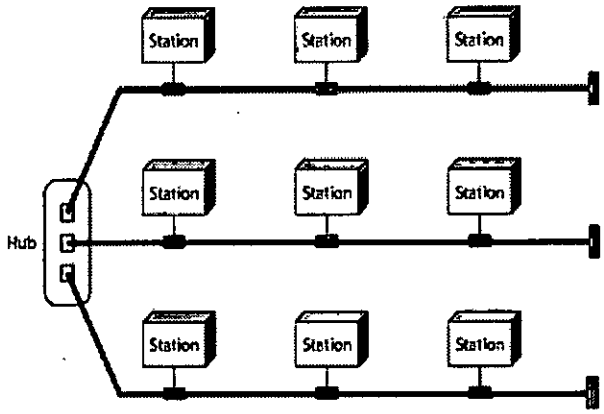


Figure 1.9 A hybrid topology: a star backbone with three bus networks

4. Network models:

Computer networks are created by different entities. Standards are needed so that these heterogeneous networks can communicate with one another. The two best-known standards are the OSI model and the Internet model.

The OSI (Open Systems Interconnection) model defines a seven-layer network. The Internet model defines a five-layer network.

5. Categories of Networks:

Networks are generally referring to two primary categories: local-area networks and wide-area networks.

A LAN normally covers an area less than 2 Meters. A WAN can be worldwide. Networks of a size in between are normally referred to as metropolitan area networks and span tens of miles.

a. Local Area Network:

A local area network (LAN) is usually privately owned and links the devices in a single office, building, or campus (see Figure 1.10). Depending on the needs of an organization and the type of technology used, a LAN can be as simple as two PCs and a printer in someone's home office; or it can extend throughout a company and include audio and video peripherals. Currently, LAN size is limited to a few kilo meters

LANs are designed to allow resources to be shared between personal computers or workstations. The resources to be shared can include hardware (e.g., a printer), software (e.g., an application program) or data. LANs are distinguished from other types of networks by their transmission media and topology. In general, a given LAN will use only one type of transmission medium. The most common LAN topologies are bus, ring, and star. Early LANs had data rates in the 4 to 16 megabits per second (Mbps) range. Today, however, speeds are normally 100 or 1000 Mbps.

Wireless LANs are the newest evolution in LAN technology.

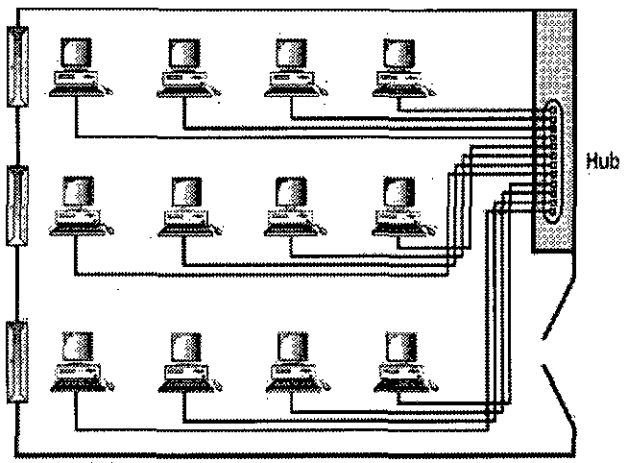


Figure 1.10 An isolated LAN connecting 12 computers to a hub in a closet

b. Wide Area Network

A wide area network (WAN) provides long-distance transmission of data, image, audio and video information over large geographic areas that may comprise a country, a continent or even the whole world. A WAN can be as complex as the backbones that connect the Internet or as simple as a dial-up line that connects a home computer to the Internet. We normally refer to the first as a switched WAN and to the second as a point-to-point WAN (Figure 1.11).

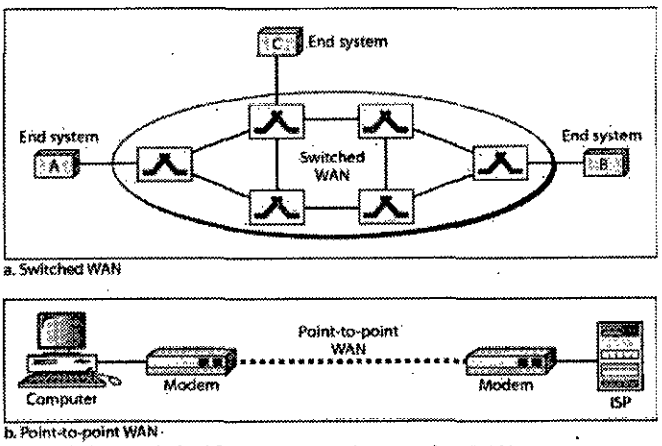


Figure 1.11 WANs: a switched WAN and a point-to-point WAN

The switched WAN connects the end systems, which usually comprise a router (internetworking connecting device) that connects to another LAN or WAN. The point-to-point WAN is normally a line leased from a telephone or cable TV provider that connects a home computer or a small LAN to an Internet service provider (ISP). This type of WAN is often used to provide Internet access.

An early example of a switched WAN is X.25, a network designed to provide connectivity between end users. X.25 is being gradually replaced by a high-speed, more efficient network called Frame Relay. A good example of a switched WAN is the asynchronous transfer mode (ATM) network, which is a network with fixed-size data unit packets called cells. Another example of WANs is the wireless WAN that is becoming more and more popular.

e. Metropolitan Area Networks:

A metropolitan area network (MAN) is a network with a size between a LAN and a WAN. It normally covers the area inside a town or a city. It is designed for customers who need a high-speed connectivity, normally to the Internet, and have endpoints spread over a city or part of city. A good example of a MAN is the part of the telephone company network that can provide a high-speed DSL line to the customer.

6. Interconnection of Networks: Interconnect

When two or more networks are connected, they become an internetwork, or internet. As an example, assume that an organization has two offices, one on the east coast and the other on the west coast. The established office on the west coast has a bus topology LAN; the newly opened office on the east coast has a star topology LAN. The president of the company lives somewhere in the middle and needs to have control over the company from her Home.

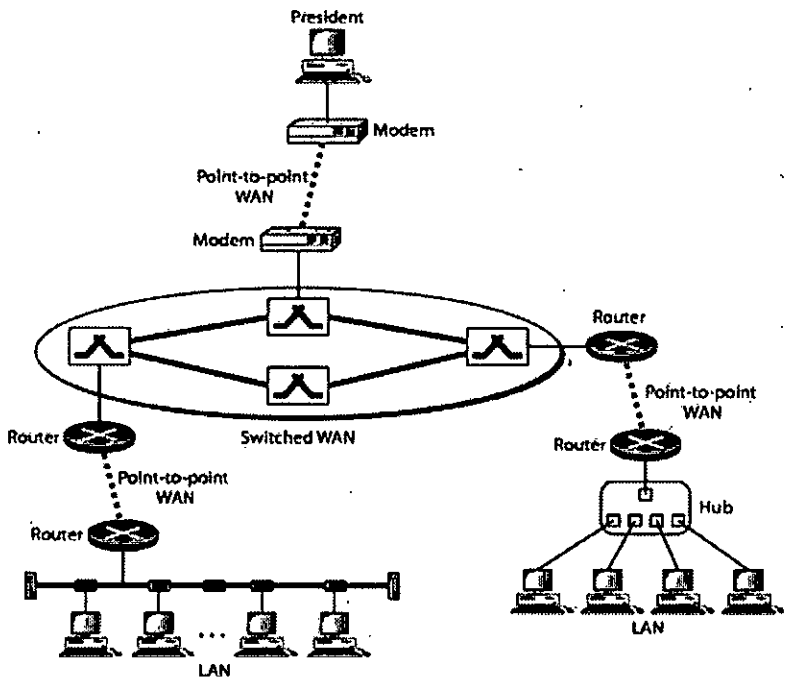


Figure 1.12 A heterogeneous network made of four WANs and two LANs

To create a backbone WAN for connecting these three entities (two LANs and the president's computer), a switched WAN (operated by a service provider such as a telecom company) has been leased. To connect the LANs to this switched WAN, however, three point-to-point WANs are required. These point-to-point WANs can be a high-speed DSL line offered by a telephone company or a cable modem line offered by a cable TV provider as shown in Figure 1.12.

TCP/IP PROTOCOL SUITE:

The TCP/IP protocol suite was developed prior to the OSI model. Therefore, the layers in the TCP/IP protocol suite do not exactly match those in the OSI model.

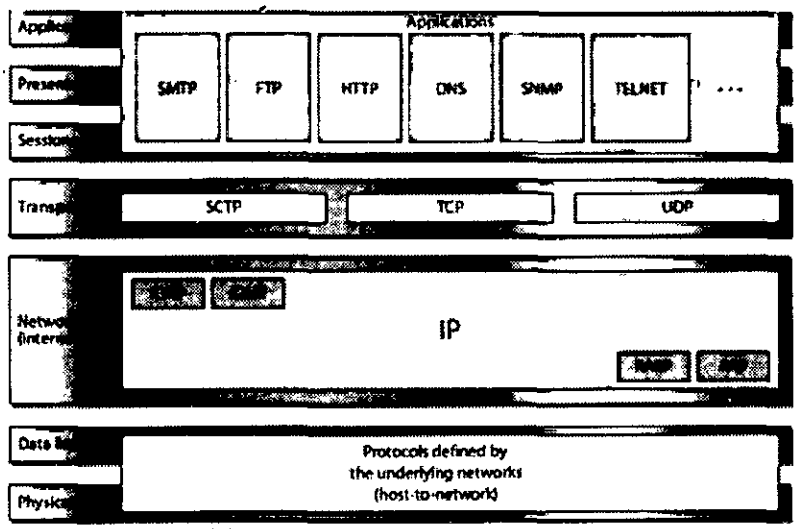


Figure.1.23 TCP/IP and OSI model

The original TCP/IP protocol suite was defined as having four layers: host-to-network, internet, transport, and application.

The TCP/IP protocol suite is made of five layers: physical, data link, network, transport, and application. The first four layers provide physical standards, network interfaces, internetworking, and transport functions that correspond to the first four layers of the OSI model. The three topmost layers in the OSI model, however, are represented in TCP/IP by a single layer called the application layer (see Figure 1.23).

1. Physical and Data Link Layers:

At the physical and data link layers, TCP/IP does not define any specific protocol. It supports all the standard and proprietary protocols. A network in a TCP/IP internetwork can be a local-area network or a wide-area network.

2. Network Layer:

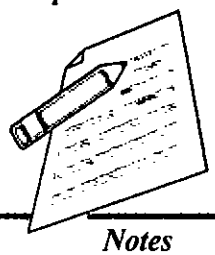
At the network layer (or, more accurately, the internetwork layer), TCP/IP supports the Internetworking Protocol. IP, in turn, uses four supporting protocols: ARP, RARP, ICMP, and IGMP.

a. Internetworking Protocol (IP)

The Internetworking Protocol (IP) is the transmission mechanism used by the TCP/IP protocols. It is an unreliable and connectionless protocol-a best-effort delivery service. The term best effort means that IP provides no error checking or tracking.

b. Address Resolution Protocol

The Address Resolution Protocol (ARP) is used to associate a logical address with a physical address. On a typical physical network, such as a LAN, each device on a link is identified by a physical or station address, usually imprinted on the network interface card (NIC). ARP is used to find the physical address of the node when its Internet address is known.



Notes

c. Reverse Address Resolution Protocol

The Reverse Address Resolution Protocol (RARP) allows a host to discover its Internet address when it knows only its physical address. It is used when a computer is connected to a network for the first time or when a diskless computer is booted.

d. Internet Control Message Protocol

The Internet Control Message Protocol (ICMP) is a mechanism used by hosts and gateways to send notification of datagram problems back to the sender. ICMP sends query and error reporting messages.

e. Internet Group Message Protocol

The Internet Group Message Protocol (IGMP) is used to facilitate the simultaneous transmission of a message to a group of recipients.

3. Transport Layer:

Traditionally the transport layer was represented in TCP/IP by two protocols: TCP and UDP. IP is a host-to-host protocol, meaning that it can deliver a packet from one physical device to another. UDP and TCP are transport level protocols responsible for delivery of a message from a process (running program) to another process. A new transport layer protocol, SCTP, has been devised to meet the needs of some newer applications.

a. User Datagram Protocol

The User Datagram Protocol (UDP) is the simpler of the two standard TCP/IP transport protocols. It is a process-to-process protocol that adds only port addresses, checksum error control, and length information to the data from the upper layer.

b. Transmission Control Protocol

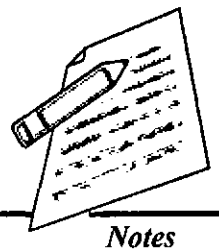
The Transmission Control Protocol (TCP) provides full transport-layer services to applications. TCP is a reliable stream transport protocol. The term stream, in this context, means connection-oriented: A connection must be established between both ends of a transmission before either can transmit data.

c. Stream Control Transmission Protocol

The Stream Control Transmission Protocol (SCTP) provides support for newer applications such as voice over the Internet. It is a transport layer protocol that combines the best features of UDP and TCP.

4. Application Layer:

The application layer in TCP/IP is equivalent to the combined session, presentation and application layers in the OSI model. Many protocols are defined at this layer.



MALWARES

Malware is short for malicious software and used as a single term to refer to virus, spyware, worms etc. Some examples of malware risks are:

1. Virus: Virus is a program written to enter in your computer and damage/ alter your files/data. A virus might corrupt or delete data on your computer. Virus can also replicate themselves.
2. Spam: Spamming is a method of flooding the Internet with copies of the same message. Most spams are commercial advertisements which are sent as unwanted emails to users. Spams are also known as electronic junk mails.
3. Hacking: Computer hacking is the practice of modifying computer hardware and software to accomplish goal outside of the creator's original purpose. In computer security, a hacker is someone who seeks and exploits weaknesses in a computer system or computer network. Hackers may be motivated by a multitude of reason such as profit, challenge, enjoyment or to evaluate those weaknesses to assist in removing them.

SECURITY CONCEPT

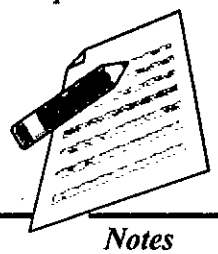
Network devices – such as routers, firewalls, gateways, switches hubs and so on, create the infrastructure of local area networks and the Internet. Securing such devices is fundamental to protecting the environment and outgoing/incoming communications. Some security methods are –

1. Firewall: A firewall is a hardware device or software application installed on the borderline of secured network to examine and control incoming and outgoing network communications. Firewall provides protection from outside attacks.
2. Antivirus: The word 'antivirus' refers to a group of features that are designed to prevent unwanted and potentially malicious files from entering your network. These features all work in different ways, which include checking for a file, size, name, or type or for the presence of a virus or grayware signature.

Cyber Ethics and IT: Cyber Ethics is the philosophic study of ethics pertaining to computers, encompassing user behaviour and what computers are programmed to do, and how this affects individuals and society.

SUMMARY

The basic elements of data communication system i.e., sender, receiver, channel. Three types of transmission modes are simplex, half duplex and full duplex. Bandwidth is the range of frequencies that make up a signal. Computer network is a group of computers which are interconnected to exchange and share information. Hub is a common connection point in a network. Modem converts digital data into analog and analog to digital. Bridge is a device that separates two or more network segments within one logical network. Network protocol defines rules and conventions for communication between network devices. FTP, PPP, TCP/IP, HTTP, HTTPS, SLIP are some of the network protocols. Firewall provides protection from outside attacks.



Notes

EXERCISE

MCQ

1. What is the minimum number of wires needed to send data over it serial communication link layer?

   (A) 1 (B) 2

   (C) 3 (D) 4

   Ans. B

2. Which data communication method is used to send data over a serial communication link?

   (A) simplex (B) half duplex

   (C) full duplex (D) all of these

   Ans . C

3. Which of the following statements is incorrect?

   (A) Teleprocessing combing telecommunication and DP techniques in online activities.

   (B) Multiplexers are designed to accept data from several I/O devices and transmit a unified stream of data on one communication line.

   (C) A half-duplex line is a communication line in which data can move in two directions, but not the same time.

   (D) Batch processing is the preferred processing mode for telecommunication operations.

   Ans. D

4. The interactive transmission of data within a time sharing system may be best suited to…..

   (A) simplex line (B) half duplex lines

   (C) full duplex line (D) bi-flex lines

   Ans. B

5. Tele printers….

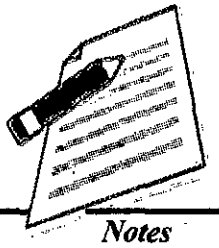
   (A) Are used for printing at remote locations, not for input.

   (B) Offer both high-speed operation and a variety of formatting controls.

   (C) Have a printer for output and a keyboard for input

   (D) are same as tele types.

   Ans. C



6. Which of the following is an example of a bounded medium?
   (A) coaxial cable (B) wave guide
   (C) fiber optic cable (D) all of these
   Ans. D

7. Coaxial cable has conductors with....
   (A) a common axis (B) equal resistance
   (C) the same diameter (D) none of these
   Ans. A

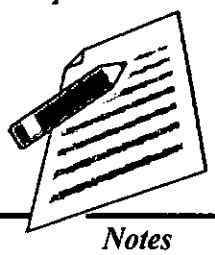
8. The area of coverage of a satellite radio beam is called it's....
   (A) beam width (B) circular polarization
   (C) footprint (D) identity
   Ans. C

9. The amount of uncertainty in a system of the symbol is called.
   (A) bandwidth (B) entropy
   (C) loss (D) quantum
   Ans. B

10. Buffering is....
   (A) the process of temporarily storing the data to allow for small variation in device speeds.
   (B) a method to reduce cross-talks
   (C) storage of data within the transmitting medium until the receiver is ready to receive
   (D) a method to reduce the routing overhead
   Ans. A

Review Questions

1. Where can Bluetooth device be useful?
2. What is RJ45 connector?
3. What is Ethernet and how does it work?
4. What do you mean by network topology? Write the names of different types of network topologies.
5. What are the different types of networks? Write in detail.
6. Write a short note on TCP/IP, PPP, FTP protocols.
7. Write a short note on virus, spam, hacking.
8. What is antivirus software?



Notes

6 FUNDAMENTALS OF INTERNET AND JAVA PROGRAMMING

- Understand the concept of internet.
- Discuss the history of internet.
- Describe the applications of internet.
- Discuss the concept of java programming.

Objective of the chapter:

The basic objective of this chapter is to through some light on the initial concepts of internet and java programming so that the history and applications of internet can be learned.

Introduction

The Internet is the global system of interconnected computer networks that use the Internet protocol suite to link devices worldwide. The purpose of the internet is to communicate between computers that are interconnected with each other. Internet is accessible to every user all over the world.

It is a network of networks that consists of private, public, academic, business, and government networks of local to global scope, linked by a broad array of electronic, wireless, and optical networking technologies.

The Internet carries a vast range of information resources and services, such as the inter-linked hypertext documents and applications of the World Wide Web (WWW), electronic mail, telephony and file sharing.

Browser is a tool used to access the internet using WWW (World Wide Web) and HTTP **(Hyper Text Transfer Protocol).** In the browser, if the user types the domain name such as www.tn.gov.in, the browser calls a protocol name DNS **(Domain Name Server).** DNS is used to get the IP address of the domain names.

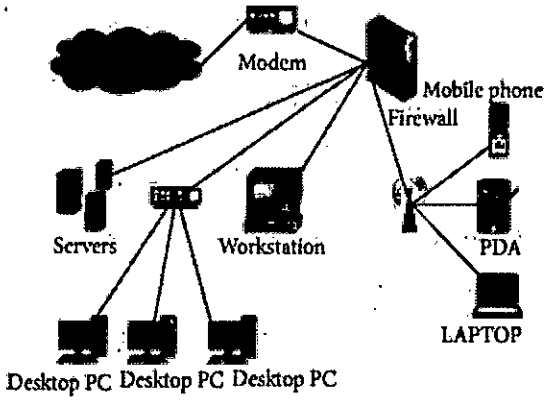


Figure 15.1 Block Diagram of Internet

Evolution of Internet:

Internet evolved in 1969 and evolved many changes in several technologies and Infrastructural levels.

- Internet was started by ARPANET (Advanced Research Project Agency Network), developed by United States. Department of Defence for communication among different government bodies, initially with four nodes.
- In 1972, the four nodes have been developed and it grown to 23 nodes located in different countries making it Internet.
- Invented TCP/IP protocols, DNS, WWW, browsers scripting languages.
- Internet is used as a medium to publish and access the information
- In 1985, The NSFNET was composed of multiple regional networks and peer networks
- In 1986, the NSFNET created a three-tiered network architecture.
- In 1988, updated the links to make it faster
- In 1990, Merit, IBM, and MCI started a new organization known as Advanced Network and Services (ANS).
- By 1991, data traffic had increased tremendously, which necessitated upgrading the NSFNET's backbone network service to T3 (45 Mbps) links.

Internet Evolution:

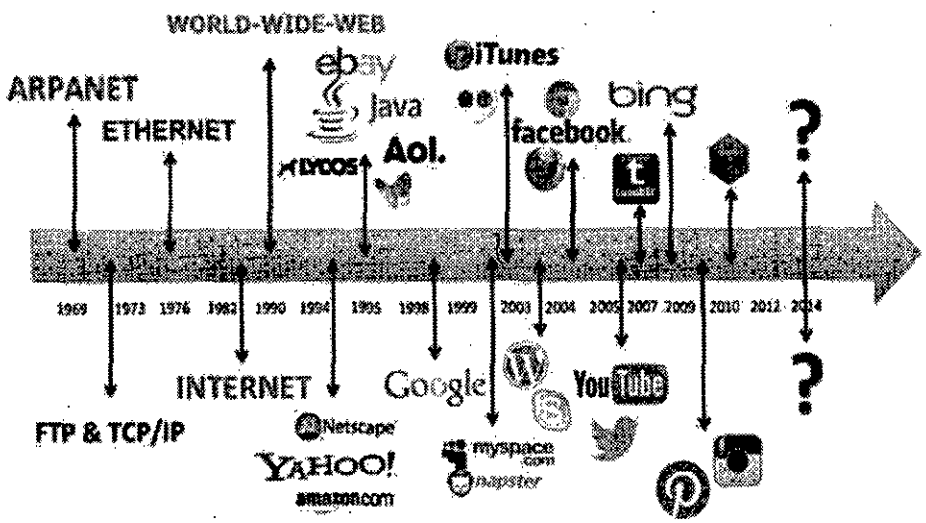
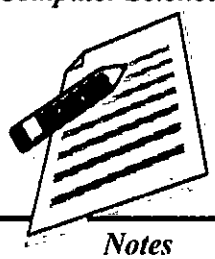


Figure 15.2 Internet History Timeline

Internet covers almost every aspect of life. Internet allows the users to communicate with the people sitting at remote locations. There are various applications available on the web that uses Internet as a medium for communication. One can find various social networking sites such : Facebook, Twitter, Yahoo, Google+, Flickr, Orkut. One can surf for any kind of information over the internet. Information regarding various topics



such as Technology, Health and Science, Social Studies, Geographical Information, Information Technology and Products can be surfed with help of a search engine.

Apart from communication and source of information, internet also serves as a medium for entertainment. Internet also allows the users to use as many services as like E-mail, Internet Banking, Online Shopping, Online Ticket Booking, Online Bill Payment and Data Sharing. Internet provides concept of electronic commerce that allows the business deals to be conducted on electronic systems.

Hardware and Software Requirements for Internet connection:

The following are the methods of connecting a computer to the Internet using software and hardware peripherals.

Three

- Connecting a computer using Wireless Broadband
- Connecting a computer using an Ethernet Cable
- Connecting a Computer Using Dial-Up Community

Hardware Requirement:

- To connect the Internet, any one of the following is mandatory.
- Modem is used to connect Internet through Telephone connection.
- NIC- Network Interface Card (wired/ wireless) facility is the most important hardware required to connect Internet. For example, the Laptop can be connected Internet through the wired/wireless.
- Dongle is used to connect the Internet using cellular network
- Wi-Fi router or Hotspot is used to connect the Internet using wireless network
- Electronic device which supports cellular network
- Internet Connectivity such as Dial-up connection, ISDN, DSL, Cable TV, wired and wireless (Cellular) Network.

Software Requirement

- The operating system should support TCP (Transfer Control Protocol) / IP (Internet Protocol), SMTP (Simple Mail Transfer Protocol), FTP (File Transfer Protocol), HTTP (Hyper Text Transfer Protocol) and HTTPS (Hyper Text Transfer Protocol Secured) protocols.
- Browsers and other Internet clients access to the web applications such as Outlook, Gmail, What's app, Facebook, Twitter and etc.

Connection Types:

The following methods are able to connect internet.

Dial-up Connection:

A dial-up connection is established when two or more data communication devices use a **Public Switched Telephone Network** (PSTN) to connect to an Internet Service Provider (ISP) from computers. Many remote locations depend on Internet dial-up connections because broadband and cable are rare in remote areas with low population. Internet Service Providers often provide dial-up connections, a feasible alternative for budget-conscious subscribers.

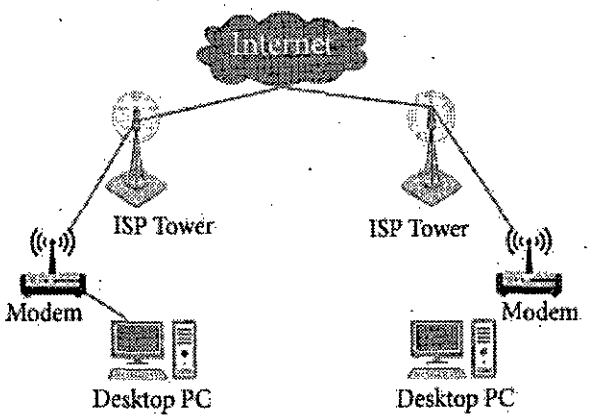


Figure 15.3 Dial-up Connection

ISDN

ISDN is the acronym of **Integrated Services Digital Network.** It establishes the connection using the phone lines (PSTN) which carry digital signals instead of analog signals. It is a set of communication standards for simultaneous digital transmission of data, voice, video, and other services over the traditional circuits of the public switched telephone network. There are two techniques to deliver ISDN services such as Basic Rate Interface (BRI) and Primary Rate Interface (PRI).

The following diagram shows accessing internet using ISDN connection:

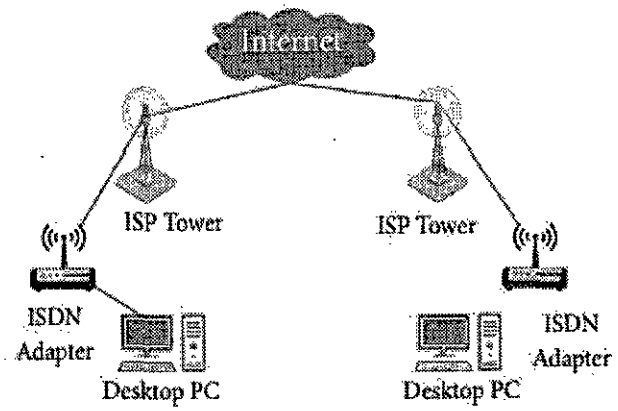


Figure 15.4a Integrated Services Digital Network

DSL:

Digital Subscriber Line (DSL) is a high-speed Internet service for homes and businesses that competes with cable and other forms of broadband Internet. DSL provides high-speed networking over ordinary Telephone lines using broadband modem technology. The technology behind DSL enables Internet and telephone service to work over the same phone line without requiring customers to disconnect either their Voice or Internet connections.

Cable TV Internet Connection (setup box):

The cable TV network can be used for connecting a computer or a local network to the Internet, competing directly with DSL (Digital Subscriber Line) technology.

This type of network is classified as HFC **(Hybrid Fiber-Coaxial),** as it uses both fibre optics and coaxial cables. The connection between the cable TV companies to the distribution points (Optical nodes) is made using fibre optics, with distances up to 25 miles (40 km). Each optical node is typically serves between 500 and 2,000 clients (customers).

The following diagram shows that how internet is accessed using Cable TV connection:

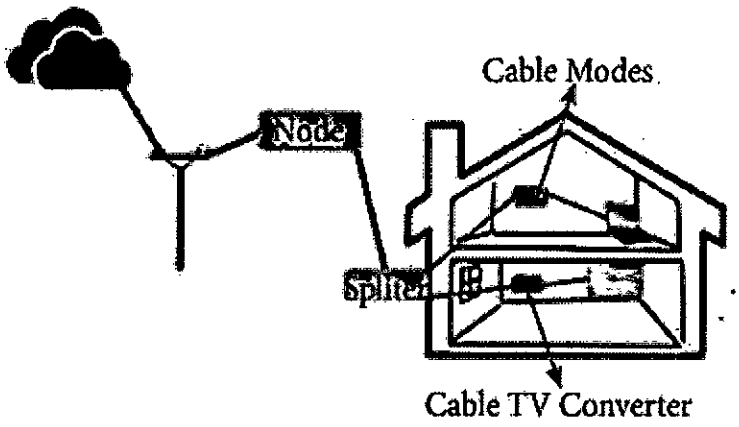
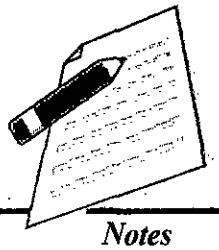


Figure 15.4b Cable TV Connection

Satellite Internet Connection:

Satellite Internet access is Internet access provided through satellite communication for domestic and enterprise usage. The facility of modern consumer grade satellite Internet service is typically provided to individual users through geostationary satellites. It provides fairly high data speeds, along with latest satellites using Ka-band to attain downstream data speeds up to 50 Mbps internet speed.



Wireless Internet Connection:

It is a technology for wireless local area networking with devices based on the IEEE 802.11 standards. Devices that can use Wi-Fi technology include personal computers, video-game consoles, phones and tablets, digital cameras, smart TVs, digital audio players and modern printers. Wi-Fi compatible devices can connect to the Internet via a WLAN and a wireless access point. Such an access point (or hotspot) has a range of about 20 meters (66 feet) indoors and a greater range of outdoors. Hotspot coverage can be as small as a single room with walls that block radio waves, or as large as many square kilometres achieved by using multiple overlapping access points.

Services Available on the Internet:

- Data Transfer
- Internet banking
- E-commerce
- E-Learning
- E-Governance
- Browsing and Chatting
- E-Mail

Data Transfer:

Data transfer is the process of using computing techniques and technologies to transmit or transfer electronic or analog data from one computer node to another. Data is transferred in the form of bits and bytes over an internet digital or analog medium, and the process enables digital or analog communications and its movement between devices. Data transfer is also known as data transmission.

Internet Banking:

Traditionally, customers used to access banking services through Retail/ corporate branch. But in this digital era Online Banking has taken vital role. The online banking is also called as internet banking, virtual banking or e-banking. This is a value added application to connecting the core banking system and provides the self-service bank facilities for customers via online. The **Figure 15.5** is the Screen Shot of the login screen of internet banking.

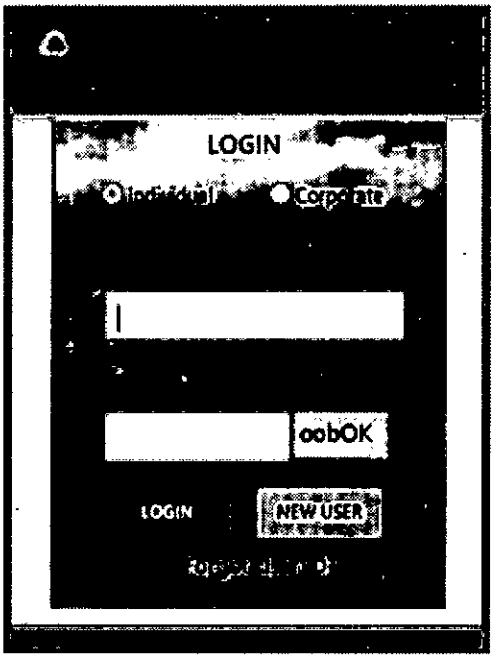


Figure 15.5 login screen of internet banking

Features:

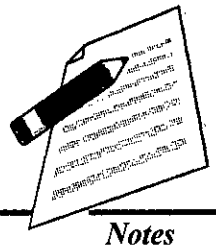
- A bank customer can perform transactional and non-transactional tasks through online banking, including
- Viewing account balances, transactions, statements of customer
- Viewing images of paid cheques, request for cheque books
- Funds transfers between the customer's linked accounts
- Paying third parties, including bill payments and third party fund transfers
- Register utility billers and make bill payments

Advantages

- Permanent online access for the banking transactions.
- Access anywhere using the application via mobile or computer
- Less time consuming, easy to use and safe
- Customer can manage their funds instantly and accurately

E-commerce:

E-commerce application is a transaction of buying or selling goods and services through online. Electronic commerce attraction technologies such as mobile commerce, electronic funds transfer, supply chain management, Internet marketing, online transaction processing, electronic data interchange (EDI), inventory management systems, and automated data collection systems.



E-commerce businesses may also employ some or all of the followings:

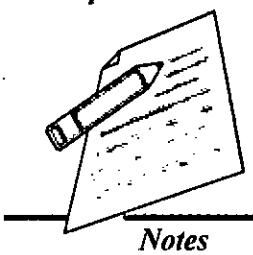
- Online shopping web sites for retail sales direct to consumers
- Providing or participating in online market places, which process third-party business-to-consumer or consumer-to-consumer sales
- Business-to-business buying and selling;
- Gathering and using demographic data through web contacts and social media
- **Business-to-business (B2B)** electronic data interchange
- Marketing to prospective and established customers by e-mail or fax (for example, with newsletters)
- Engaging in retail for launching new products and services
- Online financial exchanges for currency exchanges or trading purposes.

E-Marketing: (Electronic Marketing)

E-Marketing is the process of marketing a product or service using the internet. It also includes marketing done via e-mail and wireless media. It also called Digital Marketing.

Figure 15.6 e-Marketing

Professionals working in e-marketing must design and implement Internet marketing plans. But they also must have a broad understanding of what makes these plans effective. Those working in e-marketing must be able to carry out many tasks:



- Following business market trends
- Consulting with companies about digital marketing needs
- Resolving issues businesses have in reaching customers oriented problems
- Creating e-marketing objectives for business challenges
- Developing marketing strategies competitive business model.
- Choosing cost-effective marketing methods
- Launching digital marketing campaigns and monitoring results

E-Learning :

A learning system based on electronic resources is known as E-learning. The use of computers and the Internet forms the major component of E-learning. The E-learning can also be termed as a network enabled transfer of skills and knowledge and the delivery of education is made to a large number of recipients at the same or different times.

E-Governance:

E-governance is the application of **Information and Communication Technology** (ICT) for delivering government services, exchange of information, communication transactions, integration of various stand-alone systems and services between government-to-citizen (G2C), **Government-to-business (G2B), Government-to-Government (G2G), Government -to-Employees (G2E)** as well as back office processes and interactions within the entire government framework. Through e-governance, government services will be made available to citizens in a convenient, efficient and transparent manner. The three main target groups that can be distinguished in governance concepts are government, citizens and businesses/ interest groups. In e-governance there are no distinct boundaries.

It classify four basic models they are:

- Government-to-Citizen (customer)
- Government-to-Employees
- Government-to-Government
- Government-to-Business

Examples of e- Governance:

- Aadhaar Card is a 12-digit unique identity number issued to all Indian residents based on their biometric and demographic data.
- Inspector General of Registration portal - Tamil Nadu – www.tnreginet. gov.in used for Land and legal registrations

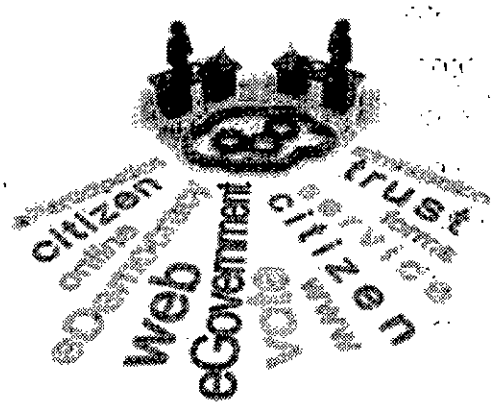


Figure 15.7 e- Governance Methodology

Online Chatting:

Online chat refers to any kind of communication via the Internet that offers a real-time transmission of text messages from sender to receiver. The chat messages are generally short in order to enable other participants to respond quickly. Online chat may address point-to-point communications as well as multicast communications from one sender to many receivers and voice and video chat and web conferencing service.

The Role of WWW as a Service on the Internet:

The World Wide Web abbreviated as WWW or the Web. It is an information space where documents and other web resources are identified by Uniform Resource Locators (URLs), interlinked by hypertext links, and can be accessed through the Internet.

Scientist Tim Berners Lee invented the World Wide Web in 1989. He introduced the first web browser computer program in 1990. The browsers available in general public on the Internet in August 1991.

WWW Operation:

The World Wide Web is the universe of network-accessible information, an expression of human knowledge. All the resources and users on the Internet that are using the Hypertext Transfer Protocol HTTP.

It is a way of exchanging information between computers on the Internet, tying them together into a vast collection of interactive multimedia resources.

Internet and Web is not the same thing: Web uses internet to pass over the information.

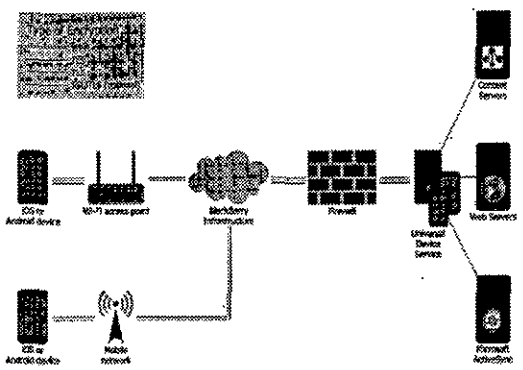
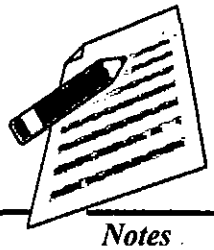


Figure 15.8 World Wide Web Architecture



Notes

Web Page:

Webpage is a document commonly written in Hyper Text Markup Language (HTML) that is accessible through the Internet or other network using an Internet browser. A web page is accessed by entering a URL address and may contain text, graphics and hyperlinks to other web pages and files. The page you are reading now is an example of a web page.

Domain Name:

A domain name is identification a string that defines an area of administrative autonomy, authority or control within the Internet. Domain names are formed by the rules and procedures of the Domain Name System (DNS). Any name registered in the DNS is a domain name. Domain names are used in various networking backgrounds and application-specific naming and addressing purposes. In general, a domain name represents an Internet Protocol (IP) resource, such as a personal computer used to access the Internet, a server computer hosting a web site.

There are several domain names available:

- Generic domain names such as **.com, .edu, .gov, .net.**
- Country level domain names such as **au, in, za, us.**

The following table shows the Generic Top-Level Domain names:

Table 15.1 Top-Level Domain names

Table 15.1 Top-Level Domain names

Domain Name	Meaning
Com	Commercial business
edu	Education
gov	U.S. Government agency
int	International entity
mil	U.S. military

Web Browser:

A web browser also called browser. It is a software application for retrieving, presenting and traversing information resources on the World Wide Web. An information resource (web data) is identified by a Uniform Resource Identifier (URI/URL) that may be a web page, image, video or other piece of content available in web server. Browsers are primarily use the World Wide Web; they can also be used to access information provided by web servers in private networks or files in file systems.

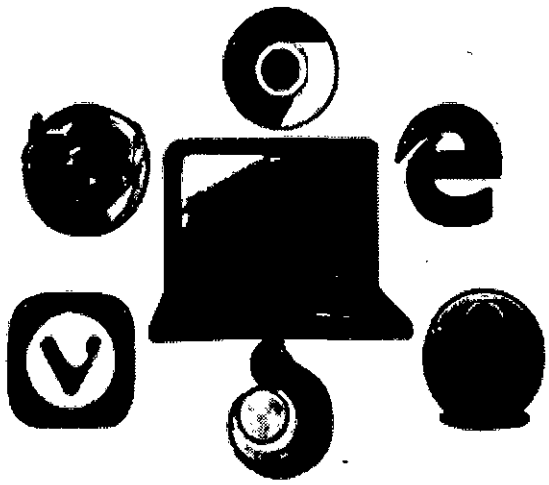


Figure 15.9 Web Browser

Table 15.2 Browsers and Vendors

Browser	Vendor
Internet Explorer	Microsoft
Google Chrome	Google
Mozilla Firefox	Mozilla
Netscape Navigator	Netscape Communications Corp.
Opera	Opera Software
Safari	Apple
Sea Monkey	Mozilla Foundation
K-meleon	K-meleon

Web Server:

A web server is a computer system application that processes requests via HTTP, the basic network protocol used to distribute information on the World Wide Web. The term can refer to the entire system, or specifically to the software that accepts and supervises the HTTP requests

Following table describes the most leading web servers available today:

Table 15.3 Web Server

S.N.	Web Server
1	Apache IITTP Server.
2.	Internet Information Services (IIS)
3.	Sun Java System Web Server

Web Hosting:

Web Facilitating is an administration of give online space to capacity of site pages .These Site pages are made accessible by means of WWW. The organizations which offer site facilitating are known as web host

Examples of Web Hosting Companies:

- Go Daddy
- Amazon Web service
- Digital Ocean
- Free webhostingarea.com

Working of Search Engine

A 'web search engine' is a software system that is designed to search for information on the World Wide Web. Huge information available on internet on various topics. The information may be a mix of web pages, images, and other types of files. Some search engines also mine data available in databases or open directories. Search engines also maintain real-time information by running an algorithm on a web crawler.

There are many different search engines available.

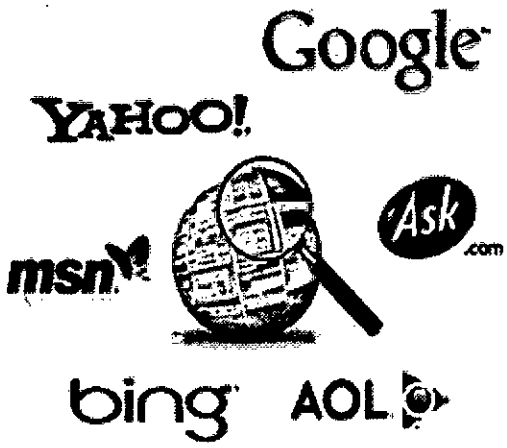
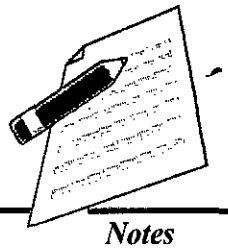


Figure 15.10 Search Engines



A First Simple Program

Now that the basic object-oriented underpinning of Java has been discussed, let's look at some actual Java programs. Let's start by compiling and running the short sample program shown here. As you will see, this involves a little more work than you might imagine.

This is a simple Java program. Call this file "Example.java".

```
*/
Example {
your program begins with a call to main ().
public static void main(String args[]) {
.out.println("This is a simple Java program.");
}
  }
```

NOTE

The descriptions that follow use the standard Java SE 8 Development Kit (JDK 8), which is available from Oracle. If you are using an integrated development environment (IDE), then you will need to follow a different procedure for compiling and executing Java programs. In this case, consult your IDE's documentation for details.

Entering the Program

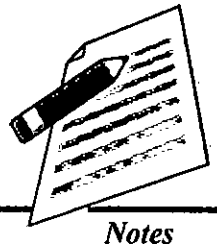
For most computer languages, the name of the file that holds the source code to a program is immaterial. However, this is not the case with Java. The first thing that you must learn about Java is that the name you give to a source file is very important. For this example, the name of the source file should be **Example.java.** Let's see why.

In Java, a source file is officially called a compilation unit. It is a text file that contains (among other things) one or more class definitions. (For now, we will be using source files that contain only one class.) The Java compiler requires that a source file use the **.java** filename extension.

As you can see by looking at the program, the name of the class defined by the program is also **Example.** This is not a coincidence. In Java, all code must reside inside a class. By convention, the name of the main class should match the name of the file that holds the program. You should also make sure that the capitalization of the filename matches the class name. The reason for this is that Java is case-sensitive. At this point, the convention that filenames correspond to class names may seem arbitrary. However, this convention makes it easier to maintain and organize your programs.

Compiling the Program

To compile the **Example** program, execute the compiler, **javac**, specifying the name of the source file on the command line, as shown here:



C:\>javac Example.java

The **javac** compiler creates a file called **Example. class** that contains the byte code version of the program. As discussed earlier, the Java byte code is the intermediate representation of your program that contains instructions the Java Virtual Machine will execute. Thus, the output of **javac** is not code that can be directly executed.

To actually run the program, you must use the Java application launcher called **java**. To do so, pass the class name **Example** as a command-line argument, as shown here:

C:\>java Example

When the program is run, the following output is displayed:

This is a simple Java program.

When Java source code is compiled, each individual class is put into its own output file named after the class and using the **.class** extension. This is why it is a good idea to give your Java source files the same name as the class they contain—the name of the source file will match the name of the **.class** file. When you execute **java** as just shown, you are actually specifying the name of the class that you want to execute. It will automatically search for a file by that name that has the **.class** extension. If it finds the file, it will execute the code contained in the specified class.

Internet Applications

SMS

SMS stands for Short Message Service. It is commonly referred to as "text messaging," It is a service for sending short messages of up to 160 characters (224 characters if using a 5-bit mode) to mobile devices, including cellular phones, smart phones and PDAs.

Voice Mail System (VMS) and answering Machine

Voice mail system can be thought as a message box for phone user to store voice messages and retrieve it through telephone. User can divert all his calls to his voice mail system when he wishes so.

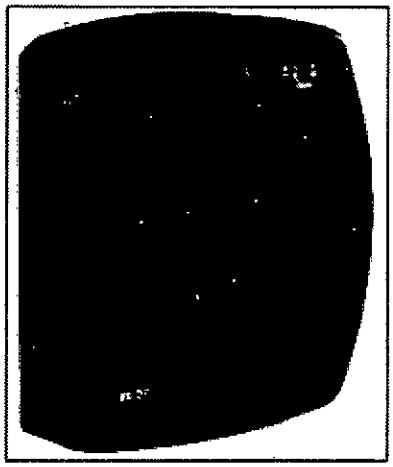
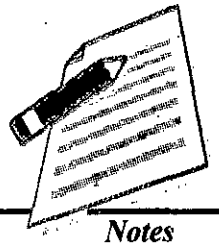


Figure 14.22 Voice Mail System



Main difference between answer machine and voice mail system is that voice mail system is a centralized system where voice mail boxes are managed for many users, while answering machine is an independent individual system connected to a telephone line. Many telephone instruments come with built-in answering machine.

Messages stored in answering machine is played back on the answering machine equipment, cannot be accessed remotely. But voice mail messages can be accessed, listened and managed from anywhere in the world through telephone line.

Answering machine is usually suitable for home use for single line, while voice mail system is more suitable for office use where there are multiple telephone connections as well as extensions through EPABX.

E-Mail

E-mail (electronic mail) is the exchange of computer-stored messages by telecommunication. E-mail messages are usually encoded in ASCII text. However, you can also send non-text files, such as graphic images and sound files, as attachments sent in binary streams. E-mail was one of the first uses of the Internet and is still the most popular use. A large percentage of the total traffic over the Internet is e-mail. E-mail can also be exchanged between online service provider users and in networks other than the Internet, both public and private.

Chat

Chat is a text-based communication that is live or in real-time. For example, when talking to someone in chat any typed text is received by other participants immediately.

Video Conferencing

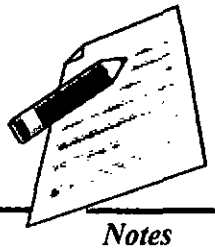
A video conference is a live, visual connection between two or more people residing in separate locations for the purpose of communication. At its simplest, video conferencing provides transmission of static images and text between two locations. It provides transmission of full-motion video images and high-quality audio between multiple locations.

For example, a point-to-point (two-person) video conferencing system works much like a video telephone. Each participant has a video camera, microphone and speakers mounted on his or her computer. As the two participants speak to one another, their voices are carried over the network and delivered to the other's speakers and whatever images appear in front of the video camera appear in a window on the other participant's monitor.

Multipoint videoconferencing allows three or more participants to sit in a virtual conference room and communicate as if they were sitting right next to each other.

A Closer Look at the First Sample Program

Although **Example.java** is quite short, it includes several key features that are common to all Java programs. Let's closely examine each part of the program.



Notes

The program begins with the following lines:

/*

This is a simple Java program. Call this file «Example.java».

*/

This is a comment. Like most other programming languages, Java lets you enter a remark into a program's source file. The contents of a comment are ignored by the compiler. Instead, a comment describes or explains the operation of the program to anyone who is reading its source code. In this case, the comment describes the program and reminds you that the source file should be called **Example.java.** Of course, in real applications, comments generally explain how some part of the program works or what a specific feature does.

Java supports three styles of comments. The one shown at the top of the program is called a multiline comment. This type of comment must begin with /* and end with */. Anything between these two comment symbols is ignored by the compiler. As the name suggests, a multiline comment may be several lines long.

The next line of code in the program is shown here:

Class Example {

This line uses the keyword **class** to declare that a new class is being defined. **Example** is an identifier that is the name of the class. The entire class definition, including all of its members, will be between the opening curly brace ({) and the closing curly brace (}). For the moment, don't worry too much about the details of a class except to note that in Java, all program activity occurs within one. This is one reason why all Java programs are (at least a little bit) object-oriented.

The next line in the program is the single-line comment, shown here:

// Your program begins with a call to main ().

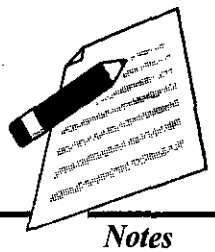
This is the second type of comment supported by Java. A single-line comment begins with a // and ends at the end of the line. As a general rule, programmers use multiline comments for longer remarks and single-line comments for brief, line-by-line descriptions. The third type of comment, a documentation comment, will be discussed in the "Comments" section later in this chapter.

The next line of code is shown here:

public static void main (String args[ ]) {

This line begins the **main ( )** method. As the comment preceding it suggests, this is the line at which the program will begin executing. All Java applications begin execution by calling **main ( )**. The full meaning of each part of this line cannot be given now, since it involves a detailed understanding of Java's approach to encapsulation. However, since most of the examples in the first part of this book will use this line of code, let's take a brief look at each part now.

The **public** keyword is an access modifier, which allows the programmer to control the visibility of class members. When a class member is preceded by **public,** then that member may be accessed by code outside the class in which it is declared.



(The opposite of **public** is **private,** which prevents a member from being used by code defined outside of its class.) In this case, **main ( )** must be declared as **public,** since it must be called by code outside of its class when the program is started. The keyword **static** allows **main ( )** to be called without having to instantiate a particular instance of the class. This is necessary since **main ( )** is called by the Java Virtual Machine before any objects are made. The keyword **void** simply tells the compiler that **main ( )** does not return a value. As you will see, methods may also return values. If all this seems a bit confusing, don't worry. All of these concepts will be discussed in detail in subsequent chapters.

As stated, **main ( )** is the method called when a Java application begins. Keep in mind that Java is case-sensitive. Thus, **Main** is different from **main.** It is important to understand that the Java compiler will compile classes that do not contain a **main ( )** method. But **java** has no way to run these classes. So, if you had typed **Main** instead of **main,** the compiler would still compile your program. However, **java** would report an error because it would be unable to find the **main ( )** method.

Any information that you need to pass to a method is received by variables specified within the set of parentheses that follow the name of the method. These variables are called parameters. If there are no parameters required for a given method, you still need to include the empty parentheses. In **main ( )**, there is only one parameter, albeit a complicated one. **String args [ ]** declares a parameter named **args,** which is an array of instances of the class **String.** (Arrays are collections of similar objects.) Objects of type **String** store character strings. In this case, **args** receives any command-line arguments present when the program is executed. This program does not make use of this information, but other programs shown later in this book will.

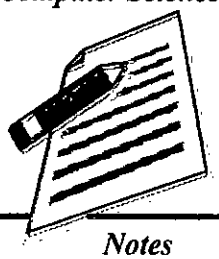
The last character on the line is the {. This signals the start of **main ( )**'s body. All of the code that comprises a method will occur between the method's opening curly brace and its closing curly brace.

One other point: **main ( )** is simply a starting place for your program. A complex program will have dozens of classes, only one of which will need to have a **main ( )** method to get things started. Furthermore, in some cases, you won't need **main ( )** at all. For example, when creating applets—Java programs that are embedded in web browsers—you won't use **main ( )** since the web browser uses a different means of starting the execution of applets.

The next line of code is shown here. Notice that it occurs inside **main ( )**.

System. out. println ("This is a simple Java program.");

This line outputs the string "This is a simple Java program." followed by a new line on the screen. Output is actually accomplished by the built-in **println ( )** method. In this case, **println ( )** displays the string which is passed to it. As you will see, **println ( )** can be used to display other types of information, too. The line begins with **System. Out.** While too complicated to explain in detail at this time, briefly, **System** is a predefined class that provides access to the system, and **out** is the output stream that is connected to the console. As you have probably guessed, console output (and



input) is not used frequently in most real-world Java applications. Since most modern computing environments are windowed and graphical in nature, console I/O is used mostly for simple utility programs, demonstration programs, and server-side code. Later in this book, you will learn other ways to generate output using Java. But for now, we will continue to use the console I/O methods.

Notice that the **println ( )** statement ends with a semicolon. All statements in Java end with a semicolon. The reason that the other lines in the program do not end in a semicolon is that they are not, technically, statements.

The first} in the program ends **main ( )**, and the last} ends the **Example** class definition.

A Second Short Program

Perhaps no other concept is more fundamental to a programming language than that of a variable. As you may know, a variable is a named memory location that may be assigned a value by your program. The value of a variable may be changed during the execution of the program. The next program shows how a variable is declared and how it is assigned a value. The program also illustrates some new aspects of console output. As the comments

at the top of the program state, you should call this file **Example2.java.**

```
/*
Here is another short example. Call this file "Example2.java".
*/
class Example2 {
Public static void main (String args []) {
 int num; // this declares a variable called num
num = 100; // this assigns num the value 100
System.out.println ("This is num: " + num);
 num = num * 2;
System.out.print("The value of num * 2 is ");
System.out.println(num);
 }
}
```

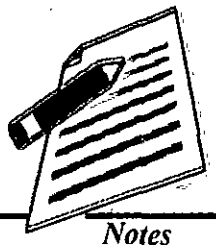
When you run this program, you will see the following output:

```
This is num: 100
The value of num * 2 is 200
```

Let's take a close look at why this output is generated. The first new line in the program is shown here:

```
int num; // this declares a variable called num
```

This line declares an integer variable called **num.** Java (like most other languages) requires that variables be declared before they are used.



Following is the general form of a variable declaration:

type var-name;

Here, type specifies the type of variable being declared, and var-name is the name of the variable. If you want to declare more than one variable of the specified type, you may use a comma-separated list of variable names. Java defines several data types, including integer, character, and floating-point. The keyword **int** specifies an integer type. In the program, the line

num = 100; // this assigns num the value 100

Assigns to **num** the value 100. In Java, the assignment operator is a single equal sign.

The next line of code outputs the value of **num** preceded by the string «This is num:"

System.out.println ("This is num: " + num);

In this statement, the plus sign causes the value of **num** to be appended to the string that precedes it, and then the resulting string is output. (Actually, **num** is first converted from an integer into its string equivalent and then concatenated with the string that precedes it. This process is described in detail later in this book.) This approach can be generalized. Using the + operator, you can join together as many items as you want within a single **println ( )** statement.

The next line of code assigns **num** the value of **num** times 2. Like most other languages, Java uses the * operator to indicate multiplication. After this line executes, **num** will contain the value 200.

Here are the next two lines in the program:

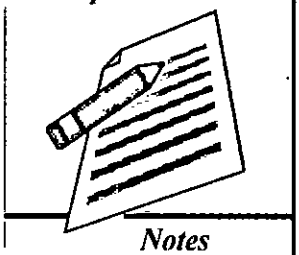
System.out.print ("The value of num * 2 is ");

System.out.println (num);

Several new things are occurring here. First, the built-in method **print ( )** is used to display the string «The value of num * 2 is «. This string is not followed by a newline. This means that when the next output is generated, it will start on the same line. The **print ( )** method is just like **println ( )**, except that it does not output a newline character after each call. Now look at the call to **println ( )**. Notice that **num** is used by itself. Both **print ( )** and **println ( )** can be used to output values of any of Java's built-in types.

SUMMARY

Internet is a network of networks because it is made up of thousands of smaller networks that can exchange information with each other. The collection of web pages interlinked with each other through hyperlinks is known as website. WWW is a system of Internet servers that support documents formatted using HTML. Web browser is a free software application that is used to view web pages, graphics etc.HTML is a mark-up language that specifies the web page layout and hyperlinks through special layout command called tags. XML defines set of rules for encoding documents in



a format which is both human and machine readable and hence can be read by any XML compatible application. The process of authoring or editing a blog is called as blogging. Cookie is the message given to the web browser by a web server.

EXERCISE

MCQ

1. What is internet?

a) a single network
b) a vast collection of different networks
c) interconnection of local area networks
d) interconnection of wide area networks

Answer: b

2. To join the internet, the computer has to be connected to a _____

a) internet architecture board
b) internet society
c) internet service provider
d) different computer

Answer: c

3. Internet access by transmitting digital data over the wires of a local telephone network is provided by _____

a) leased line
b) digital subscriber line
c) digital signal line
d) digital leased line

Answer: b

4. ISP exchanges internet traffic between their networks by _____

a) internet exchange point
b) subscriber end point
c) isp end point
d) internet end point

Answer: a

5. Which of the following protocols is used in the internet?

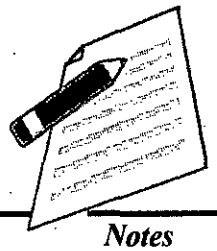
a) HTTP
b) DHCP
c) DNS
d) DNS, HTTP and DNS

Answer: d

6. The size of an IP address in IPv6 is _____

a) 32 bits
b) 64 bits
c) 128 bits
d) 265 bits

Answer: c



7. Internet works on _____
   a) packet switching
   b) circuit switching
   c) both packet switching and circuit switching
   d) data switching

   Answer: a

8. Which one of the following is not an application layer protocol used in internet?
   a) remote procedure call
   b) internet relay chat
   c) resource reservation protocol
   d) local procedure call

   Answer: c

9. Which protocol assigns IP address to the client connected in the internet?
   a) DHCP
   b) IP
   c) RPC
   d) RSVP

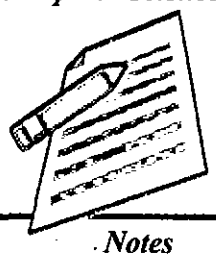
   Answer: a

10. Which one of the following is not used in media access control?
    a) ethernet
    b) digital subscriber line
    c) fibre distributed data interface
    d) packet switching

    Answer: d

Review Questions

1. Give two examples for each of the following: (a) ISP (b) Web browsers (c) Social Networking Websites (d) Instant Messaging Services.
2. Explain URL and its parts with the help of an example.
3. What is the need of IP addresses on a network?
4. How data is transmitted from one computer to another on Internet?
5. Explain the working of Internet.
6. Differentiate between (a) Instant Messaging and Chat rooms. (b) FTP and HTTP.
7. Why Twitter is known as 'SMS of the Internet'?
8. Define blog, blogging and blog post.
9. Explain Video-Conferencing and types of video conferencing techniques.
10. What is XML?



7 INTRODUCTION TO C++

- Understand the concept of C++.
- Discuss the features of C++.
- Describe the applications of C++.
- Discuss the components of C++.

Objective of the chapter:

The basic objective of this chapter is to through some light on the initial concepts of C++ so that the components and applications of C++ can be learned.

Introduction

C++ is one of the most popular programming language developed by Bjarne Stroustrup at AT & T Bell Lab during 1979. C++ supports both procedural and Object Oriented Programming paradigms. Thus, C++ is called as a **hybrid language.** C++ is a superset (extension) of its predecessor C language. Bjarne Stroustrup named his new language as "C with Classes". The name C++ was coined by Rick Mascitti where ++ is the C language increment operator.

Bjarne Stroustrup

Inventor of C++ Programming Language

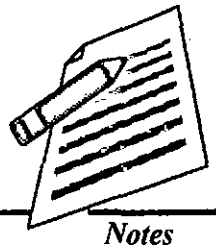
Bjarne is a Danish Computer Scientist was born on 30th December 1950. He has a Master degree in Mathematics and Computer Science in 1975 from Aarhus University, Denmark and Ph.D. in Computer Science in 1979 from the University of Cambridge, England.

History of C++

C++ was developed by Bjarne Stroustrup at AT & T Bell Laboratory during 1979. C++ is originally derived from C language and influenced by many languages like Simula, BCPL, Ada, ML, CLU and ALGOL 68. Till 1983, it was referred "New C" and "C with Classes". In 1983, the name was changed as C++ by Rick Mascitti.

C++ is standardized by the International Organization for Standardization (ISO). The latest standard version published in December 2017 as ISO/IEC 14882:2017 which is informally known as C++17. The first standardized version was published in 1998 as ISO/IEC 14882:1998 which was then enhanced by the C++03 (ISO/IEC 14882:2003), C++11 (ISO/IEC 14882:2011) and C++14 (ISO/ IEC 14882:2014). The Next standard version will be C++20 in 2020.

C# (C-Sharp), D, Java and newer versions of C languages has been influenced by C++.



Benefits of learning C++

- C++ is a highly portable language and is often the language of choice for multi-device, multi-platform app development.
- C++ is an object-oriented programming language and includes **classes, inheritance, polymorphism, data abstraction and encapsulation.**
- C++ has a rich function library.
- C++ allows exception handling, inheritance and function overloading which are not possible in C.
- C++ is a powerful, efficient and fast language. It finds a wide range of applications from GUI applications to 3D graphics for games to real-time mathematical simulations.

History of C++

C++ was developed by Bjarne Stroustrup at AT & T Bell Laboratory during 1979. C++ is originally derived from C language and influenced by many languages like Simula, BCPL, Ada, ML, CLU and ALGOL 68. Till 1983, it was referred "New C" and "C with Classes". In 1983, the name was changed as C++ by Rick Mascitti.

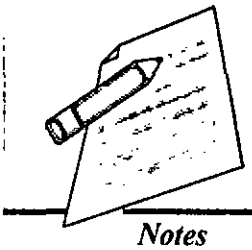
C++ is standardized by the International Organization for Standardization (ISO). The latest standard version published in December 2017 as ISO/IEC 14882:2017 which is informally known as C++17. The first standardized version was published in 1998 as ISO/IEC 14882:1998 which was then enhanced by the C++03 (ISO/IEC 14882:2003), C++11 (ISO/IEC 14882:2011) and C++14 (ISO/ IEC 14882:2014). The Next standard version will be C++20 in 2020.

C# (C-Sharp), D, Java and newer versions of C languages has been influenced by C++.

Character set

Character set is a set of characters which are allowed to write a C++ program. A character represents any alphabet, number or any other symbol (special characters) mostly available in the keyboard. C++ accepts the following characters.

Alphabets	A .... Z, a ..... z
Numeric	0 9
Special Characters	+ - * / ~ ! @ # $ % ^& [ ] ( ) { } = >< _ \ | ? . , : ' " ;
White space	Blank space, Horizontal tab (→), Carriage return (↵), Newline, Form feed
Other characters	C++ can process any of the 256 ASCII characters as data.



Lexical Units (Tokens):

C++ program statements are constructed by many different small elements such as commands, variables, constants and many more symbols called as operators and punctuators. These individual elements are collectively called as **Lexical units** or **Lexical elements or Tokens.** C++ has the following tokens:

- Keywords
- Identifiers
- Literals
- Operators
- Punctuators

TOKEN:

The smallest individual unit in a program is known as a Token or a Lexical unit

1. Keywords

Keywords are the reserved words which convey specific meaning to the C++ compiler. They are the essential elements to construct C++ programs. Most of the keywords are common to C, C++ and Java.

C++ is a case sensitive programming language so, all the keywords must be in lowercase.

Table 9.1 C++ Keywords

Table 9.1 C++ Keywords

asm	auto	break	case	catch
char	class	const	continue	default
delete	do	double	else	enum
extern	float	for	friend	goto
if	inline	int	long	new
operator	private	protected	public	register
return	short	signed	sizeof	static
struct	switch	template	this	throw
try	typedef	union	unsigned	virtual
void	volatile	while		

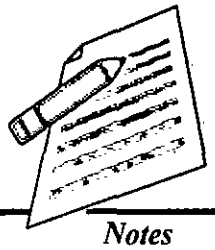
- With revisions and additions, the recent list of keywords also includes:

Using, namespace, bal, static_cast, const_cast, dynamic_cast, true, false

- Identifiers containing a double underscore are reserved for use by C++ implementations and standard libraries and should be avoided by users.

2. Identifiers

Identifiers are the user-defined names given to different parts of the C++ program viz. variables, functions, arrays, classes etc.; these are the fundamental building blocks of a program. Every language has specific rules for naming the identifiers.



Rules for naming an identifier:

- The first character of an identifier must be an alphabet or an underscore (_).
- Only alphabets, digits and underscore are permitted. Other special characters are not allowed as part of an identifier.
- C++ is case sensitive as it treats upper and lower-case characters differently.
- Reserved words or keywords cannot be used as an identifier name.

As per ANSI standards, C++ places no limit on its length and therefore all the characters are significant.

Identifiers	Valid / Invalid	Reason for invalid
Num	Valid	
NUM	Valid	
_add	Valid	
total_sales	Valid	
tamilMark	Valid	
num-add	Invalid	Contains special character (-)
this	Invalid	This is one of the keyword. Keyword cannot be used as identifier names.
2myfile	Invalid	Name must start begins with an alphabet or an underscore

- You may use an underscore in variable names to separate different parts of the name (e.g.: ***total_sales*** is a valid identifier whereas the variable called ***total sales*** is an invalid identifier).
- You may use capital style notation, such as ***tamilMark*** i.e. capitalizing the first letter of the second word.

3. Literals (Constants)

Literals are data items whose values do not change during the execution of a program. Therefore Literals are called as Constants. C++ has several kinds of literals:

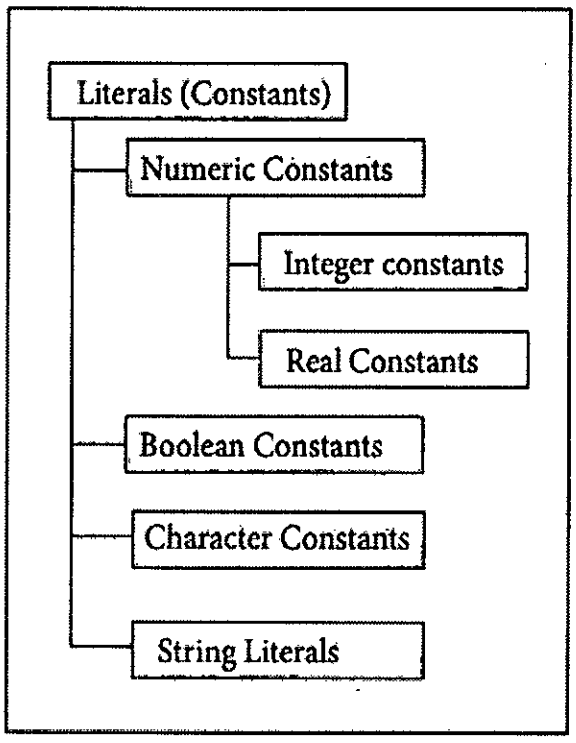


Figure 9.1 Types of Constants

Numeric Constants:

As the name indicates, the numeric constants are numeric values, which are used as constants. Numeric constants are further classified as:

- Integer Constants (or) Fixed point constants.
- Real constants (or) Floating point constants.

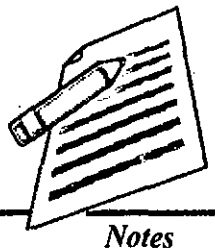
(1) Integer Constants (or) Fixed point constants

Integers are whole numbers without any fractions. An integer constant must have at least one digit without a decimal point. It may be signed or unsigned. Signed integers are considered as negative, commas and blank spaces are not allowed as part of it. In C++, there are three types of integer constants: (i) Decimal (ii) Octal (iii) Hexadecimal

(i) Decimal

Any sequence of one or more digits (0 .... 9)

Valid	Invalid
725	7,500 (Comma is not allowed)
-27	66 5(Blank space is not allowed)
4.56	9$ (Special Character not allowed)



If you assign **4.56** as an integer decimal constant, the compiler will accept only the integer portion of **4.56** i.e. **4**. It will simply ignore **.56.**

If a Decimal constant declared with fractions, then the compiler will take only the integer part of the value and it will ignore its fractional part. This is called as "Implicit Conversion". It will be discussed later.

(ii) Octal

Any sequence of one or more octal values (0 .... 7) that begins with 0 is considered as an Octal constant.

Valid	Invalid
012	05,600(Commas is not allowed)
-027	04.56 (Decimal point is not allowed)**
+0231	0158 (8 is not a permissible digit in octal system)

When you use a fractional number that begins with 0, C++ has consider the number as an integer not an Octal.

(iii) Hexadecimal

Any sequence of one or more Hexadecimal values (0 .... 9, A .... F) that starts with 0x or 0Xis considered as an Hexadecimal constant.

Valid	Invalid
0x123	0x1,A5 (Commas is not allowed)
0X568	0x.14E (Decimal point is not allowed like this)

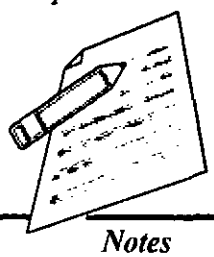
The suffix L or l and U or u added with any constant forces that to be represented as a long or unsigned constant respectively.

(2) Real Constants (or) Floating point constants

A real or floating point constant is a numeric constant having a fractional component. These constants may be written in fractional form or in exponent form.

Fractional form of a real constant is a signed or unsigned sequence of digits including a decimal point between the digits. It must have at least one digit before and after a decimal point. It may have prefix with + or - sign. A real constant without any sign will be considered as positive.

Exponent form of real constants consists of two parts: **(1) Mantissa** and **(2) Exponent.** The mantissa must be either an integer or a real constant. The mantissa followed by a letter E or e and the exponent. The exponent should also be an integer. For example, 58000000.00 may be written as 0.58 × 108 or 0.58E8.



Notes

Mantissa (Before E)	Exponent (After E)
0.58	8

Example:

$5.864\ E\text{-}1 \rightarrow 5.864 \times 10^{1} \rightarrow 58.64$

$5864\ E\text{-}2 \rightarrow 5864 \times 10^{-2} \rightarrow 58.64$

$0.5864\ E\text{-}2 \rightarrow 0.5864 \times 10^{2} \rightarrow 58.64$

Boolean Literals

Boolean literals are used to represent one of the Boolean values (True or false). Internally true has value 1 and false has value 0.

Character constant

A character constant is any valid single character enclosed within single quotes. A character constant in C++ must contain one character and must be enclosed within a single quote.

Valid character constants: 'A', '2', '$'

Invalid character constants: "A"

The value of a single character constant has an equivalent ASCII value. For example, the value of 'A' is 65.

Escape sequences (or) Non-graphic characters

C++ allows certain non-printable characters represented as character constants. Non-printable characters are also called as non-graphical characters. Non-printable characters are those characters that cannot be typed directly from a keyboard during the execution of a program in C++, for example: backspace, tabs etc. These non-printable characters can be represented by using escape sequences. An escape sequence is represented by a backslash followed by one or two characters.

Table 9.2 Escape Sequences

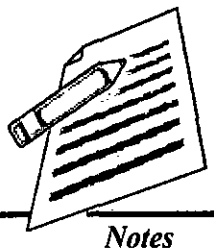


Table 9.2 Escape Sequences

Escape sequence	Non-graphical character
\a	Audible or alert bell
\b	Backspace
\f	Form feed
\n	Newline or linefeed
\r	Carriage return
\t	Horizontal tab
\v	Vertical tab
\\	Backslash
\'	Single quote
\"	Double quote
\?	Question Mark
\On	Octal number
\xHn	Hexadecimal number
\0	Null

Even though an escape sequence contains two characters, they should be enclosed within single quotes because, C++ consider escape sequences as character constants and allocates one byte in ASCII representation.

ASCII (American Standard Code for Information Interchange) was first developed and published in 1963 by the X3 committee, a part of the American Standards Association (ASA).

String Literals

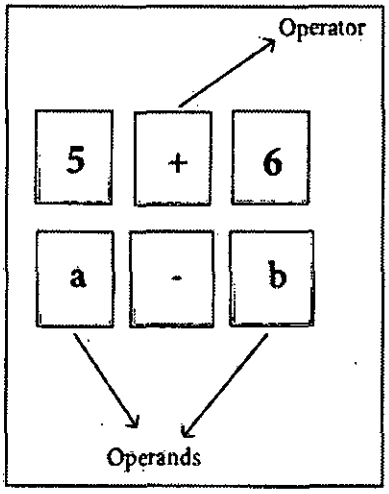
Sequences of characters enclosed within double quotes are called as String literals. By default, string literals are automatically added with a special character **'\0' (Null)** at the end. Therefore, the string "welcome" will actually be represented as "welcome\0" in memory and the size of this string is not 7 but 8 characters i.e., inclusive of the last character \0.

Valid string Literals: "A", "Welcome" "1234"

Invalid String Literals: 'Welcome', '1234'

4. Operators

The symbols which are used to do some mathematical or logical operations are called as **"Operators".** The data items or values that the operators act upon are called as **"Operands".**



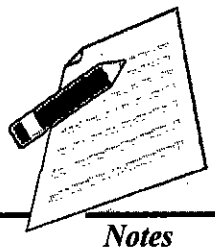
In C++, The operators are classified on the basis of the number of operands.

(i) **Unary Operators** - Require only one operand
(ii) **Binary Operators** - Require two operands
(iii)**Ternary Operators** - Require three operands
C++ Operators are classified as:
(1) Arithmetic Operators
(2) Relational Operators
(3) Logical Operators
(4) Bitwise Operators
(5) Assignment Operators
(6) Conditional Operator
(7) Other Operators

(1) Arithmetic Operators

Arithmetic operators perform simple arithmetic operations like addition, subtraction, multiplication, division etc.

Operator	Operation	Example
+	Addition	10 + 5 = 15
-	Subtraction	10 – 5 = 5
*	Multiplication	10 * 5 = 50
/	Division	10 / 5 = 2 (Quotient of the division)
%	Modulus (To find the reminder of a division)	10 % 3 = 1(Remainder of the division)



- The above mentioned arithmetic operators are binary operators which requires minimum of two operands.

Increment and Decrement Operators

++ (Plus, Plus) Increment operator

-- (Minus, Minus) Decrement operator

An increment or decrement operator acts upon a single operand and returns a new value. Thus, these operators are unary operators. The increment operator adds 1 to its operand and the decrement operator subtracts 1 from its operand. For example,

- **x++** is the same as **x = x+1;**

It adds 1 to the present value of x

- **x--** is the same as to **x = x–1;**

It subtracts 1 from the present value of x

The ++ or -- operators can be placed either as prefix (before) or as postfix (after) to a variable. With the prefix version, C++ performs the increment / decrement before using the operand.

For example: **N1=10, N2=20;**

S = ++N1 + ++N2;

The following Figure explains the working process of the above statement.

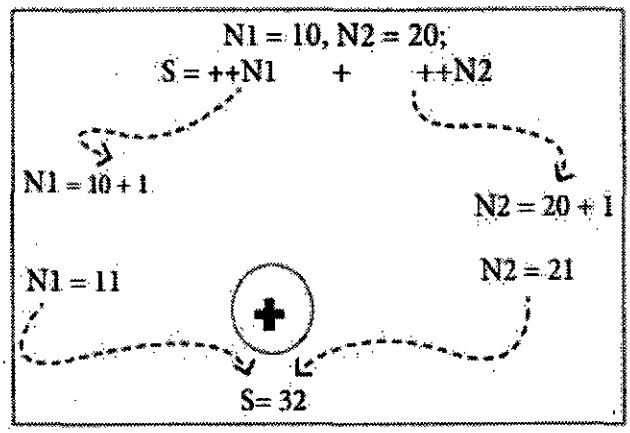


Figure 9.2 Prefix Increment Working process

In the above example, the value of num is first incremented by 1, then the incremented value is assigned to the respective operand.

With the postfix version, C++ uses the value of the operand in evaluating the expression before incrementing / decrementing its present value.

For example: **N1=10, N2=20;**

S = N1++ + ++N2;

The following Figure explains the working process of the above statement.

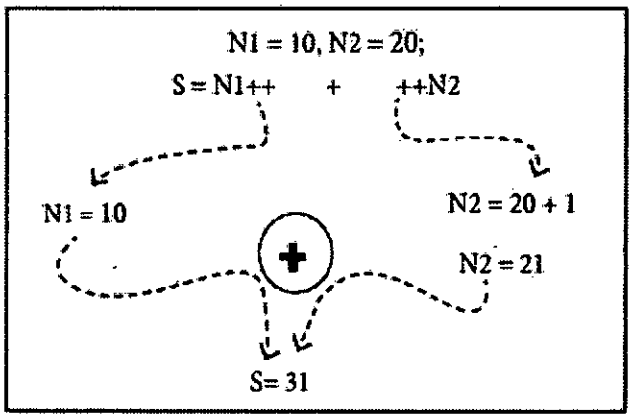


Figure 9.3 Postfix Increment Working process

In the above example, the value assigned to operand N1 is taken into consideration, first and then the value will be incremented by 1.

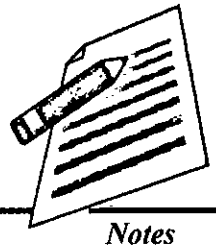
Operator	Operation	Example (Assume n=2; what will be value of a)	
		Prefix	Postfix
++	Increment	a=++n;	a=n++;
		Value of a = 3	Value of a = 2
--	Decrement	a=--n;	a=n--;
		Value of a=1	Value of a=2

(2) Relational Operators

Relational operators are used to determine the relationship between its operands. When the relational operators are applied on two operands, the result will be a Boolean value i.e. 1 or 0 to represents True or False respectively. C++ provides six relational operators. They are,

Operator	Operation	Example
>	Greater than	a > b
<	Less than	a < b
>=	Greater than or equal to	a >= b
<=	Less than or equal to	a <= b
==	Equal to	a == b
!=	Not equal	a != b

- In the above examples, the operand a is compared with b and depending on the relation, the result will be either 1 or 0. i.e., 1 for true, 0 for false.
- All six relational operators are binary operators.



(3) Logical Operators

A logical operator is used to evaluate logical and relational expressions. The logical operators act upon the operands that are themselves called as logical expressions. C++ provides three logical operators.

Table 9.3 Logical Operators

Operator	Operation	Description
&&	AND	The logical AND combines two different relational expressions in to one. It returns 1 (True), if both expression are true, otherwise it returns 0 (false).
||	OR	The logical OR combines two different relational expressions in to one. It returns 1 (True), if either one of the expression is true. It returns 0 (false), if both the expressions are false.
!	NOT	NOT works on a single expression / operand. It simply negates or inverts the truth value. i.e., if an operand / expression is 1 (true) then this operator returns 0 (false) and vice versa

- AND, OR both are binary operators whereas NOT is an unary operator.

Example: a = 5, b = 6, c = 7;

Expression	Result
(a<b) && (b<c)	1 (True)
(a>b) && (b<c)	0 (False)
(a<b) || (b>c)	1 (True)
!(a>b)	1 (True)

(4)Bitwise Operators

Bitwise operators work on each bit of data and perform bit-by-bit operation. In C++, there are three kinds of bitwise operators, which are:

(i) Logical bitwise operators

(ii) Bitwise shift operators

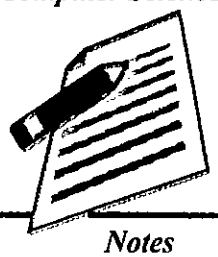
(iii)One's compliment operators

(i) Logical bitwise operators:

& Bitwise AND (Binary AND)

| Bitwise OR (Binary OR)

^ Bitwise Exclusive OR (Binary XOR)



Notes

- Bitwise AND (&) will return 1 (True) if both the operands are having the value 1 (True); Otherwise, it will return 0 (False)
- Bitwise OR (|) will return 1 (True) if any one of the operands is having a value 1 (True); It returns 0 (False) if both the operands are having the value 0 (False)
- Bitwise XOR (^) will return 1 (True) if only one of the operand is having a value 1 (True). If both are True or both are False, it will return 0 (False).

Truth table for bitwise operators (AND, OR, XOR)

A	B	A & B	A | B	A ^ B
1	1	1	1	0
1	0	0	1	1
0	1	0	1	1
0	0	0	0	0

Example:

If a = 65, b=15

Equivalent binary values of 65 = 0100 0001; 15 = 0000 1111

Operator	Operation	Result
&	a & b	a 0 1 0 0 0 0 0 1 b 0 0 0 0 1 1 1 1 a & b 0 0 0 0 0 0 0 1 (a&b) = 0000 0001$_2$ = 1$_{10}$
|	a | b	a 0 1 0 0 0 0 0 1 b 0 0 0 0 1 1 1 1 a | b 0 1 0 0 1 1 1 1 (a|b) = 01001111$_2$ = 79$_{10}$
^	a ^ b	a 0 1 0 0 0 0 0 1 b 0 0 0 0 1 1 1 1 a ^ b 0 1 0 0 1 1 1 0 (a^b) = 0100 1110$_2$ = 78$_{10}$

(ii) The Bitwise shift operators:

There are two bitwise shift operators in C++, **Shift left** (<<) and **Shift right** (>>).

Shift left (<<) – The value of the left operand is moved to left by the number of bits specified by the right operand. Right operand should be an unsigned integer.

Shift right (>>) – The value of the left operand is moved to right by the number of bits specified by the right operand. Right operand should be an unsigned integer.

Example:

If a =15; Equivalent binary value of a is 0000 1111

Operator	Operation	Result
<<	a << 3	a: 0 0 0 0 1 1 1 1 a << 3: 0 1 1 1 1 0 0 0 (a<< 3) = $01111000_2 = 120_{10}$
>>	a >> 2	a: 0 0 0 0 1 1 1 1 a>> 2: 0 0 0 0 1 1 0 0 (a>> 2) = $0000\ 0011_2 = 3_{10}$

(iii) The Bitwise one's compliment operator:

The bitwise One's compliment operator ~**(Tilde)**,inverts all the bits in a binary pattern, that is, all 1's become 0 and all 0's become 1. This is a unary operator.

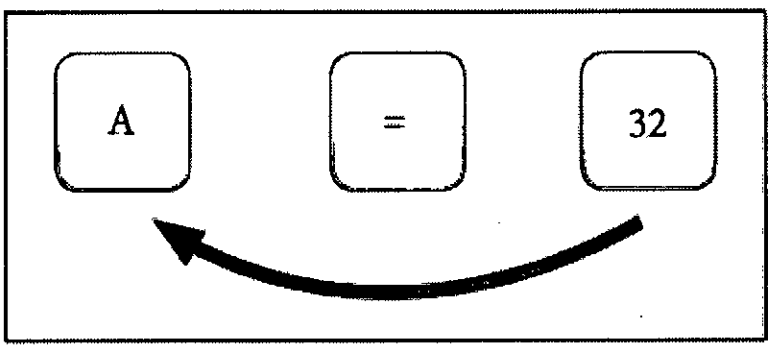
Example:

If a =15; Equivalent binary values of a is 0000 1111

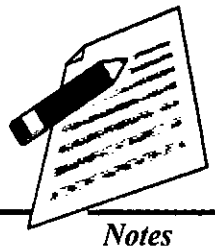
Operator	Operation	Result
~	(~a)	a: 0 0 0 0 1 1 1 1 (~a): 1 1 1 1 0 0 0 0 (~a) = $1111\ 0000_2 = -16_{10}$

(5)Assignment Operator:

Assignment operator is used to assign a value to a variable which is on the left hand side of an assignment statement. = (equal to) is commonly used as the assignment operator in all computer programming languages. This operator copies the value at the right side of the operator to the left side variable. It is also a binary operator.



C++ uses different types of assignment operators. They are called as Shorthand assignment operators.



Operator	Name of Operator	Example
+=	Addition Assignment	a = 10; c = a += 5; (ie, a = a + 5) c = 15
-=	Subtraction Assignment	a = 10; c = a -= 5; (ie. a = a – 5) c = 5
*=	Multiplication Assignment	a = 10; c = a *= 5; (ie. a = a * 5) c = 50
/=	Division Assignment	a = 10; c = a /= 5; (ie. a = a / 5) c = 2
%=	Modulus Assignment	a = 10; c = a %= 5; (ie. a = a % 5) c = 0

(6) Conditional Operator:

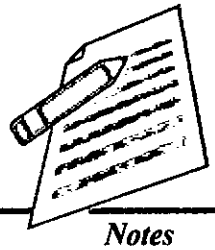
In C++, there is only one conditional operator is used. ?: is a conditional Operator. This is a Ternary Operator. This operator is used as an alternate to if ... else control statement. We will learn more about this operator in later chapters along with if .... else structure.

(7) Other Operators:

The Comma operator	Comma (,) is an operator in C++ used to string together several expressions. The group of expression separated by comma is evaluated from left to right.
Sizeof	This is called as compile time operator. It returns the size of a variable in bytes.
Pointer	* Pointer to a variable & Address of
Component selection	. Direct component selector -> Indirect component selector
Class member operators	:: Scope access / resolution .* Dereference ->* Dereference pointer to class member

Precedence of Operators:

Operators are executed in the order of precedence. The operands and the operators are grouped in a specific logical way for evaluation. This logical grouping is called as an Association.



The order of precedence:

( ) [ ]	Operators within parenthesis are performed first	Higher
++, --	Postfix increment / decrement	
++, --	Prefix increment / decrement	
*, /, %	Multiplication, Division, Modulus	
+, -	Addition, Subtraction	
<, <=, >, >=	Less than, Less than or equal to, Greater than, Greater than or equal to	
==, !=	Equal to, Not equal to	
&&	Logical AND	
||	Logical OR	
?:	Conditional Operator	
=	Simple Assignment	
+=, -=, *=, /=	Shorthand operators	
,	Comma operator	Lower

In C++, one or two operators may be used in different places with different meaning. For example: Asterisk (*) is used for multiplication as well as for pointer to a variable.

5. Punctuators

Punctuators are symbols, which are used as delimiters, while constructing a C++ program. They are also called as **"Separators".** The following punctuators are used in C++; most of these symbols are very similar to C and Java.

Separator	Description	Example

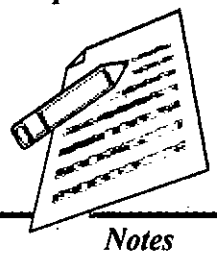
Curly braces { }	Opening and closing curly braces indicate the start and the end of a block of code. A block of code containing more than one executable statement. These statements together are called as "compound statement"	int main ( )
		{
		int x=10, y=20, sum;
		sum = x + y;
		cout << sum;
		}

Parenthesis ( )	Opening and closing parenthesis indicate function calls and function parameters.	clrscr( ); int main ( )
Square brackets [ ]	It indicates single and multidimensional arrays.	int num[5]; char name[50];
Comma ,	It is used as a separator in an expression	int x=10, y=20, sum;

Semicolon ;	Every executable statement in C++ should terminate with a semicolon	int main ( )
		{
		int x=10, y=20, sum;
		sum = x + y;
		cout << sum;
		}

Colon :	It is used to label a statement.	private:

Comments
//
/* */
	Any statement that begins with // are considered as comments. Comments are simply ignored by compilers. i.e., compiler does not execute any statement that begins with a //
	// Single line comment
	/* ..... */ Multiline comment
		/* This is written
		By myself
		to learn CPP */
		int main ( )
		{
		int x=10, y=20, sum;
		// to sum x and y
		sum = x + y;
		cout << sum;
		}



Notes

Execution of C++ program:

For creating and executing a C++ program, one must follow four important steps.

(1) Creating Source code

Creating includes typing and editing the valid C++ code as per the rules followed by the C++ Compiler.

(2) Saving source code with extension .cpp

After typing, the source code should be saved with the extension .cpp

(3) Compilation

This is an important step in constructing a program. In compilation, compiler links the library files with the source code and verifies each and every line of code. If any mistake or error is found, it will inform you to make corrections. If there are no errors, it translates the source code into machine readable object file with an extension .obj

(4) execution

This is the final step of construction of a C++ Program. In this stage, the object file becomes an executable file with extension .exe. Once the program becomes an executable file, the program has an independent existence. This means, you can run your application without the help of any compiler or IDE.

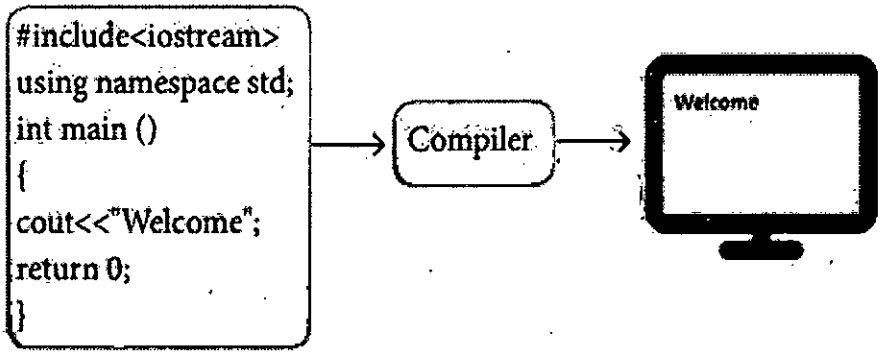


Figure 9.8 Execution

Concept of Data types:

Let us look at the following example,

Name = Ram

Age = 15

Average_Mark = 85.6

In the above example, Name, Age, Average_mark are the fields which hold the values such as Ram, 15, and 85.6 respectively.

In a programming language, **fields** are referred as **variables** and the **values** are referred to as **data.** Each data item in the above example is looking different. That is, "Ram" is a sequence of alphabets and the other two data items are numbers. The first value is a whole number and the second one is a fractional number. In real-world scenarios,

there are lots of different kinds of data we handle in our day-to-day life. The nature or type of the data item varies, for example distance (from your home to school), ticket fare, cost of a pen, marks, temperature, etc.,

In C++ programming, before handling any data, it should be clearly specified to the language compiler, regarding what kind of data it is, with some predefined set of data types.

C++ Data types

In C++, the data types are classified as three main categories

(1) Fundamental data types

(2) User-defined data types and

(3) Derived data types.

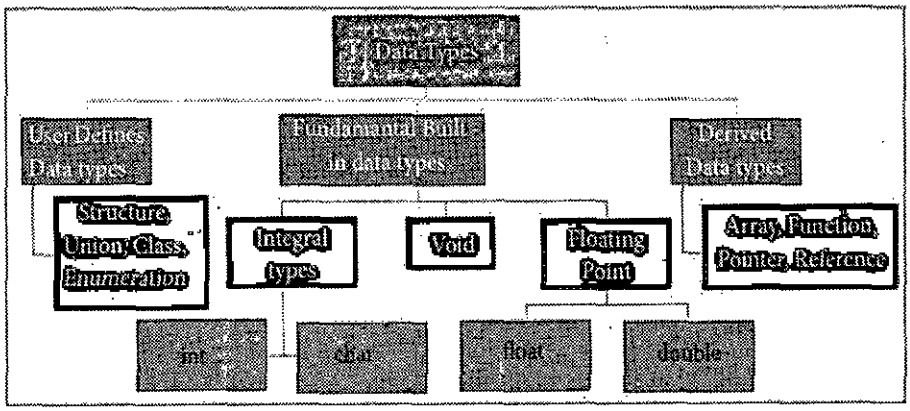


Figure 9.13 Data types in C++

In this chapter, we are going to learn about only the Fundamental data types.

In order to understand the working of data types, we need to know about variables. **The variables are the named memory locations to hold values of specific data types. In C++, the variables should be declared explicitly with their data types before they are actually used.**

Syntax for declaring a variable:

<data type> <variable name>;

Example:

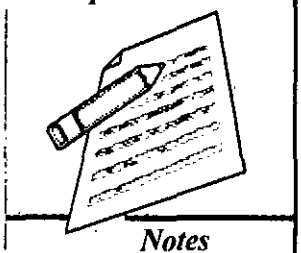
int num1;

To declare more than one variable which are of the same data type using a single statement, it can be declared by separating the variables using a comma.

Example:

int num1, num2, sum;

For example, to store your computer science marks; first you should declare a variable to hold your marks with a suitable data type. Choosing an appropriate data type needs more knowledge and experience. Usually, marks are represented as whole numbers. Thus, your variable for storing the computer science marks should be an integer data type.



Example:

int comp_science_marks;

Now, one variable named **comp_science_marks** is ready to store your marks.

We will learn more about variables later in this chapter.

SUMMARY

The built in or basic data types supported by C++ are integer, floating point and character type. The identifier is a sequence of characters taken from C++ character set. The size of operator can be used to find how many bytes are required for an object to store in memory. The process in which one pre-defined type of expression is converted into another type is called conversion. There are two types of conversion in C++. i.e., implicit conversion and explicit conversion. A number which does not change its value during execution of a program is known as a constant. The variables can be used to hold different values at different times during the execution of a program.

EXERCISE

MCQ

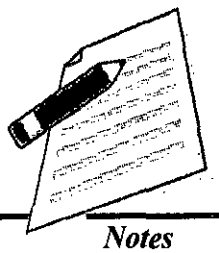
1. Who developed C++?
   (a) Charles Babbage **(b) Bjarne Stroustrup**
   (c) Bill Gates (d) Sundar Pichai

2. What was the original name given to C++?
   (a) CPP (b) Advanced C
   (c) C with Classes (d) Class with C

3. Who coined C++?
   (a) Rick Mascitti (b) Rick Bjarne
   (c) Bill Gates (d) Dennis Ritchie

4. The smallest individual unit in a program is:
   (a) Program (b) Algorithm
   (c) Flowchart **(d) Tokens**

5. Which of the following operator is extraction operator of C++?
   (a) >> (b) <<
   (c) <> (d) ^^



6. Which of the following statements is not true?

   (a) Keywords are the reserved words convey specific meaning to the C++ compiler.

   (b) Reserved words or keywords can be used as an identifier name.

   (c) An integer constant must have at least one digit without a decimal point.

   (d) Exponent form of real constants consists of two parts

7. Which of the following is a valid string literal?

   (a) 'A' (b) 'Welcome'

   (c) 1232 **(d) "1232"**

8. A program written in high level language is called as

   (a) Object code **(b) Source code**

   (c) Executable code (d) All the above

9. Assume a=5, b=6; what will be result of a&b?

   (a) 4 (b) 5

   (c) 1 (d) 0

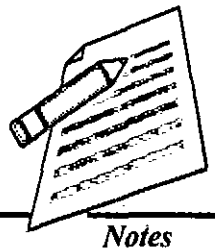
10. Which of the following is called as compile time operators?

    (a) size of (b) pointer

    (c) virtual (d) this

Review Questions

1. What is the difference between a keyword and an identifier?
2. Explain the following terms: (i) Literals (ii) Implicit conversion.
3. How many ways are there in C++ to represent an integer constant?
4. State the rules for comment statement.
5. What is an escape sequence? Describe any two backslash character constant?
6. Explain briefly about standard input/output stream of C++.
7. Write the equivalent C++ expression for the following algebraic expression: (i) 2AB 2BC 2CA 2A + + (ii) 4 2 x 3 (iii) 2 b 4ac 2a − .
8. Evaluate the following C++ expression: int a, b = 2, k = 4; a = b * 3/4 + k/4 + 8 – b + 5/8; 9. Write an appropriate C++ statement for each of the following: (i) Read the values of a, b and c. (ii) Write the values of a and b in one line followed by the value of c on another line. (iii) Write the values of a and b in one line separated by blanks and value of c after two blank lines.
10. Write a program that will find out the square of a given number.



8 GENERAL CONCEPT OF OOP

- Understand the concept of OOP.
- Discuss the features of OOP.
- Describe the applications of OOP.
- Discuss the components of OOP.

Objective of the chapter:

The basic objective of this chapter is to through some light on the initial concepts of OOP so that the components and applications of OOP can be learned.

Introduction

Object-Oriented Programming (OOP) is the term used to describe a programming approach based on classes and objects. The object-oriented paradigm allows us to organize software as a collection of objects that consist of both data and behaviour. This is in contrast to conventional functional programming practice that loosely connects data and behaviour.

Since 1980's the word **'object'** has appeared in relation to programming languages, with almost all languages developed since 1990 having object-oriented features. This chapter introduces general OOP concepts.

Programming Paradigms

Paradigm means organizing principle of a program. It is an approach to programming. There are different approaches available for problem solving using computer. They are Procedural programming, Modular Programming and Object Oriented Programming

Procedural programming

Procedural means a list of instructions were given to the computer to do something. Procedural programming aims more at procedures. This emphasis on doing things.

Important features of procedural programming

- Programs are organized in the form of subroutines or sub programs
- All data items are global
- Suitable for small sized software application

- Difficult to maintain and enhance the program code as any change in data type needs to be propagated to all subroutines that use the same data type. This is time consuming.
- Example: **FORTRAN** and **COBOL**.

Modular programming

Modular programming consists of a list of instructions that instructs the computer to do something. But this **Paradigm consists of multiple modules, each module has a set of functions of related types. Data is hidden under the modules.** Arrangement of data can be changed only by modifying the module

Important features of Modular programming

- Emphasis on algorithm rather than data
- Programs are divided into individual modules
- Each modules are independent of each other and have their own local data
- Modules can work with its own data as well as with the data passed to it.
- Example: **Pascal** and **C**

Object Oriented Programming

Object Oriented Programming paradigm emphasizes on the data rather than the algorithm. It implements programs using **classes and objects.**

Class: A Class is a construct in C++ which is used to bind data and its associated function together into a single unit using the encapsulation concept. Class is a user defined data type. Class represents a group of similar objects.

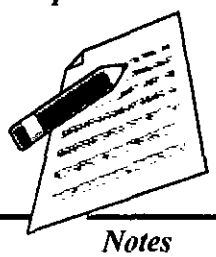
It can also be defined as a template or blueprint representing a group objects that share common properties and relationship.

Objects: Represents data and its associated function together into a single unit. Objects are the basic unit of OOP. Basically an object is created from a class. They are instances of class also called as class variables

An identifiable entity with some characteristics and behaviour is called object.

Important features of Object oriented programming

- Emphasizes on data rather than algorithm
- Data abstraction is introduced in addition to procedural abstraction
- Data and its associated operations are grouped in to single unit
- Programs are designed around the data being operated
- Relationships can be created between similar, yet distinct data types
- Example: **C++, Java, VB.Net, Python etc.**



Basic Concepts of OOP

The Object Oriented Programing has been developed to overcome the drawbacks of procedural and modular programming. It is widely accepted that object-oriented programming is the most important and powerful way of creating software.

The Object-Oriented Programming approach mainly encourages:

• **Modularisation:** where the program can be decomposed into **modules.**

•**Software re-use:** where a program can be composed from existing and new modules.

Main Features of Object Oriented Programming

- Data Abstraction
- Encapsulation
- Modularity
- Inheritance
- Polymorphism

Encapsulation

The mechanism by which the data and functions are bound together into a single unit is known as **Encapsulation.** It implements abstraction.

Encapsulation is about binding the data variables and functions together in class. It can also be called **data binding.**

Encapsulation is the most striking feature of a class. The data is not accessible to the outside world, and only those functions which are wrapped in the class can access it. These functions provide the interface between the object's data and the program.

This encapsulation of data from direct access by the program is called data hiding or information hiding.

Data Abstraction

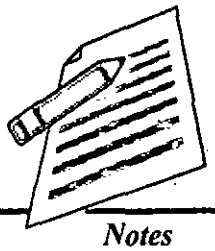
Abstraction refers to showing only the essential features without revealing background details. Classes use the concept of abstraction to define a list of abstract attributes and function which operate on these attributes. They encapsulate all the essential properties of the object that are to be created. The attributes are called **data members** because they hold information. The functions that operate on these data are called **methods or member function.**

Modularity

Modularity is designing a system that is divided into a set of functional units (named modules) that can be composed into a larger application.

Inheritance

Inheritance is the technique of building new classes **(derived class)** from an existing Class **(base class).** The most important advantage of inheritance is **code reusability.**



Polymorphism

Polymorphism is the ability of a message or function to be displayed in more than one form.

Advantages of OOP

Re-usability:

"Write once and use it multiple times" you can achieve this by using class.

Redundancy:

Inheritance is the good feature for data redundancy. If you need a same functionality in multiple classes you can write a common class for the same functionality and inherit that class to sub class.

Easy Maintenance:

It is easy to maintain and modify existing code as new objects can be created with small differences to existing ones.

Security:

Using data hiding and abstraction only necessary data will be provided thus maintains the security of data.

Disadvantages of OOP

Size:

Object Oriented Programs are much larger than other programs.

Effort:

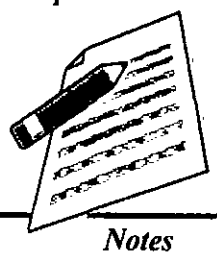
Object Oriented Programs require a lot of work to create.

Speed:

Object Oriented Programs are slower than other programs, because of their size.

SUMMARY

Paradigm means organizing principle of a program. It is an approach to programming. Procedural or Modular programming means a list of instructions were given and each instructions tell the computer to do something. Procedural programming aims more ate procedures. In this Programs are organized in the form of subroutines or sub programs Modular programming combines related procedures in a module and hides data under modules. Object Oriented programming Paradigm emphasizes on the data rather than the algorithm. It implements programs using classes and objects. Class is a user defined data type. Class represents a group of similar objects. Objects are the basic unit of OOP. It represents data and associated function together in to a single unit. The mechanism by which the data and functions are bound together into a single unit is known as ENCAPSULATION. It implements abstraction . Abstraction



Notes

refers to showing only the essential features without revealing background details. Modularity is designing a system that is divided into a set of functional units that can be composed into a larger application. Polymorphism is the ability of a message or function to be displayed in more than one form. Inheritance is the technique of building new classes (derived class) from an existing class. The most important advantage of inheritance is code reusability. Inheritance is transitive in nature.

EXERCISE

MCQ

1. The term is used to describe a programming approach based on classes and objects is

   (A) OOP (B) POP
   (C) ADT (D) SOP

2. The paradigm which aims more at procedures.

   (A) Object Oriented Programming
   (B) Procedural programming
   (C) Modular programming
   (D) Structural programming

3. Which of the following is a user defined data type?

   (A) class (B) float
   (C) int (D) object

4. The identifiable entity with some characteristics and behaviour is.

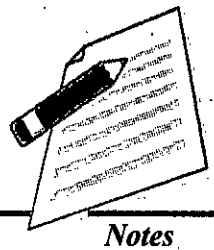
   (A) class **(B) object**
   (C) structure (D) member

5. The mechanism by which the data and functions are bound together into a single unit is known as

   (A) Inheritance **(B) Encapsulation**
   (C) Polymorphism (D) Abstraction

6. Insulation of the data from direct access by the program is called as

   (A) Data hiding (B) Encapsulation
   (C) Polymorphism (D) Abstraction



7. Which of the following concept encapsulate all the essential properties of the object that are to be created?
   (A) class (B) Encapsulation
   (C) Polymorphism **(D) Abstraction**

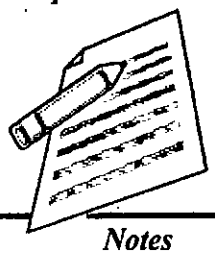
8. Which of the following is the most important advantage of inheritance?
   (A) data hiding **(B) code reusability**
   (C) code modification (D) accessibility

9. "Write once and use it multiple times" can be achieved by
   (A) redundancy **(B) reusability**
   (C) modification\ (D) composition

10. Which of the following supports the transitive nature of data?
    (A) Inheritance (B) Encapsulation
    (C) Polymorphism (D) Abstraction

Review Questions

1. What are the features of OOP?
2. Explain the following terms briefly: (a) Data abstraction (b) Data encapsulation (c) Polymorphism (d) Inheritance
3. Describe the various benefits of OOP.



Notes

9 CONTROL STATEMENTS

- Understand the concept of statements.
- Discuss the features of statements.
- Describe the concept of control statements.
- Discuss the types of control statements.

Objective of the chapter:

The basic objective of this chapter is to through some light on the initial concepts of control statements so that the types and applications of control statements can be learned.

Introduction

A computer program is a set of statements or instructions to perform a specific task. These statements are intended to perform specific action. The action may be of variable declarations, expression evaluations, assignment operations, decision making, looping and so on.

There are two kinds of statements used in C++.

(i) Null statement

(ii) Compound statement

(i) Null statement

The **"null** or **empty statement"** is a statement containing only a semicolon. It takes the flowing form:

; // it is a null statement

Null statements are commonly used as placeholders in iteration statements or as statements on which to place labels at the end of compound statements or functions.

(ii) Compound (Block) statement

C++ allows a group of statements enclosed by pair of braces {}. This group of statements is called as a compound statement or a block.

The general format of compound statement is:

statement1;
statement2;
statement3;
}

For example

{

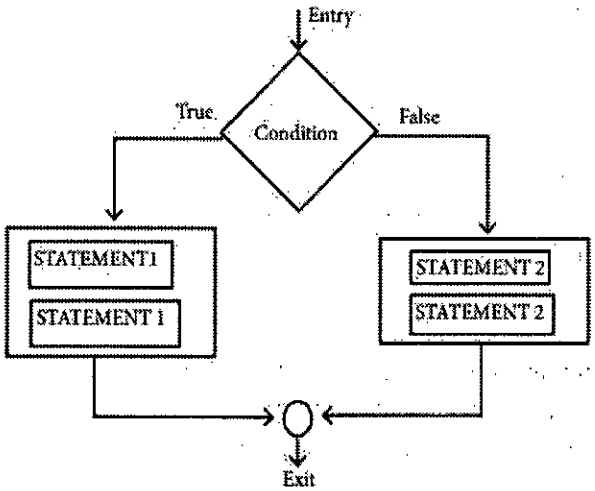
```
int x, y;
x = 10;
y = x + 10;
}
```

The compound statement or block is a treated as a single unit and may be appear anywhere in the program.

Control Statements

Control statements are statements that alter the sequence of flow of instructions.

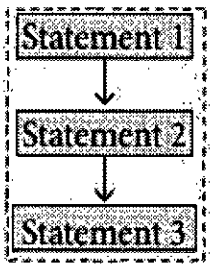
Selection statement



In a program, statements may be executed sequentially, selectively or iteratively. Every programming language provides statements to support sequence, selection (branching) and iteration.

If the Statements are executed sequentially, the flow is called as sequential flow. In some situations, if the statements alter the flow of execution like branching, iteration, jumping and function calls, this flow is called as control flow.

Sequence statement

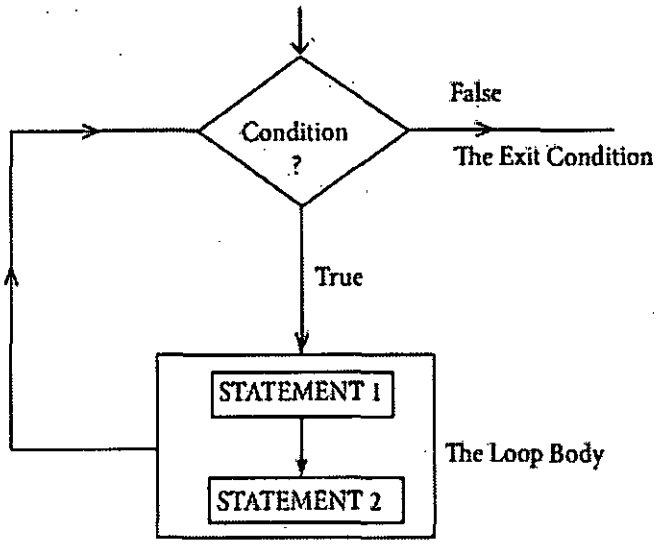


The **sequential statement** are the statements, that are executed one after another only once from top to bottom. These statements do not alter the flow of execution. These statements are called as sequential flow statements. They are always end with a semicolon (;).

The selection statement means the statement (s) are executed depends upon a condition. If a condition is true, a true block (a set of statements) is executed otherwise a false block is executed. This statement is also called **decision statement or selection statement** because it helps in making decision about which set of statements are to be executed.

Iteration statement

The **iteration statement** is a set of statement are repetitively executed depends upon a conditions. If a condition evaluates to true, the set of statements (true block) is executed again and again. As soon as the condition becomes false, the repetition stops. This is also known as **looping statement** or iteration statement.



The set of statements that are executed again and again is called the **body of the loop.** The condition on which the execution or exit from the loop is called **exit-condition or *test-condition*.**

Generally, all the programming languages support this type of statements to write programs depend upon the problems. C++ also supports this type of statements. These statements will be discussed in coming sections.

In selection statements and iteration statements are executed depends upon the conditional expression. The conditional expression evaluates either true or false.

Selection statements

In a program a decision causes a onetime jump to a different part of a program. Decisions in C++ are made in several ways, most importantly with if .. else ... statement which chooses between two alternatives. Another decision statement , switch creates branches for multiple alternatives sections of code, depending on the value of a single variable.

if statement

The if statement evaluates a condition, if the condition is true then a true-block (a statement or set of statements) is executed, otherwise the true-block is skipped. The general syntax of the statement is:

if (expression)

true-block;

statement-x;

In the above syntax, **if** is a keyword that should contain expression or condition which is enclosed within parentheses. If the expression is true (nonzero) then the true-block is executed and followed by statement-x are also executed, otherwise, the control passes to statement-x. The true-block may consist of a single statement, a compound statement or empty statement. The control flows of if statement and the corresponding flow chart are shown below.

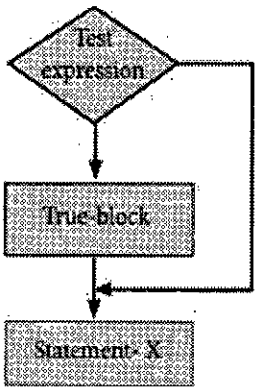
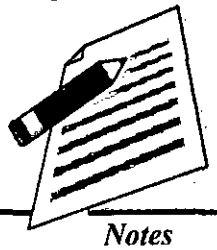


Illustration 10.1 C++ program to calculate total expenses using if statement

```
#include<iostream>
using namespace std;
int main()
{
    int qty, dis=0;
    float rate, tot;
    cout<<»\nEnter the quantity «;
   cin>>qty;
    cout<<»\nEnter the rate «;
    cin>>rate;
    if ( qty> 500)
    dis=10;
    tot = (qty * rate) - ( qty * rate * dis / 100);
    cout<<»The total expenses is «<< tot;
    return 0;
```




```
}
```

Output

First Run

Enter the quantity 550

Enter the rate 10

The total expenses is 4950

Second Run

Enter the quantity 450

Enter the rate 10

The total expenses is 4500

In the first execution of the program, the test condition evaluates to true, since qty> Therefore, the variable dis which is initialized to 0 at the time of declaration, now gets a new value 10. The total expenses are calculated using a new dis value.

In the second execution of the program, the test condition evaluates to false, since qty> 500. Thus, the variable dis which is initialized to 0 at the time of declaration, remains 0. Hence, the expression after the minus sign evaluates to 0. So, the total expenses are calculated without discount.

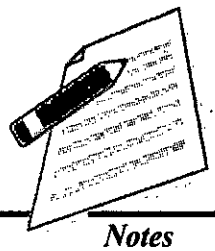
Illustration 10.2 C++ program to check whether a person is eligible to vote using if statement

```
#include <iostream>
using namespace std;
int main()
{
    int age;
    cout<< «\n Enter your age: «;
    cin>> age; if(age>=18)
    cout<< «\n You are eligible for voting ....»;
    cout<< «This statement is always executed.»;
    return 0;
}
```

```
#include<iostream>
using namespace std;
int main()
{
    int age;
    cout<< "\n Enter your age: ";
    cin>> age;
    if(age>=18)
        cout<< "\n You are eligible for voting ....";
    cout<< "This statement is always executed.";
    return 0;
}
```

The pair of braces is not required because if condition followed by only one statement

Output



Enter your age: 23

You are eligible for voting....

This statement is always executed.

Illustration 10.3 C++ program to calculate bonus using if statement

```
#include<iostream>
using namespace std;
int main()
{
int bonus,yr_of_ser;
cout<<"\nEnter your year of service ";
cin>>yr_of_ser;
if ( yr_of_ser> 3 )
{bonus=2000;cout<<»\n Your bonus is « <<bonus;}
cout<<"\nCongratulations...";
return 0;
}
```

Output

Enter your year of service 5

Your bonus is 2000

Congratulations...

if-else statement

In the above examples of if, you have seen so for allow you to execute a set of statement is a condition evaluates to true. What if there is another course of action to be followed if the condition evaluates to false. There is another form of if that allows for this kind of either or condition by providing an else clause. The syntax of the if-else statement is given below:

if (expression)

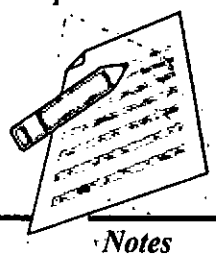
{True-block;}

else

{False-block;}

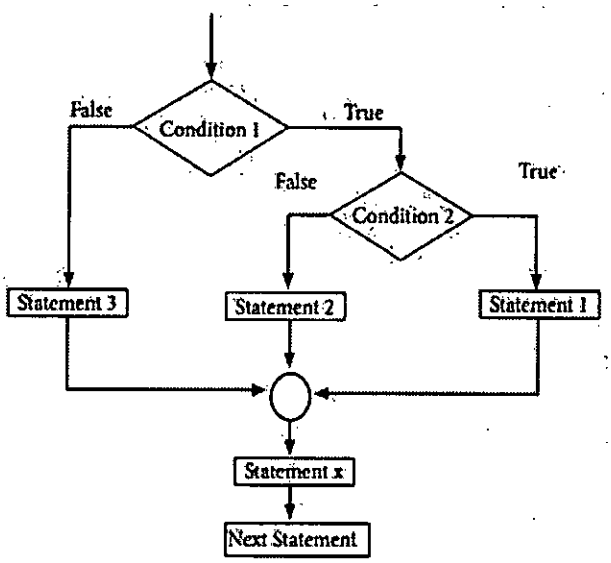
Statement-x

In if-else statement, first the expression or condition is evaluated either true of false. If the result is true, then the statements inside true-block are executed and false-block is skipped. If the result is false, then the statement inside the false-block is executed i.e., the true-block is skipped.

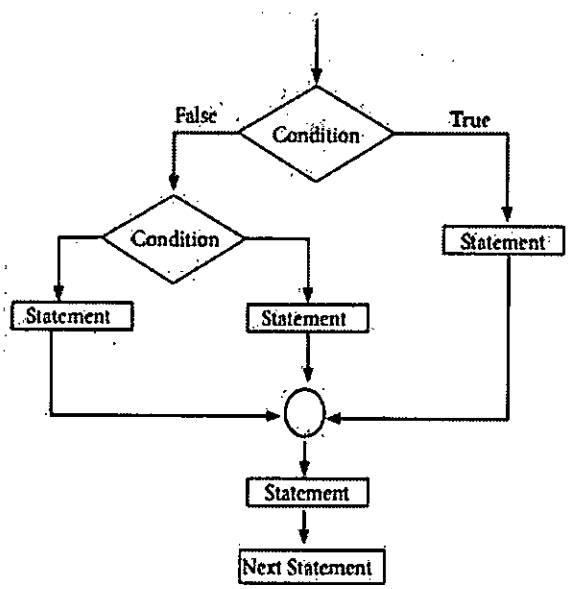



```
    }
    else
    {
    False_Part_Statements;
    }
}
```

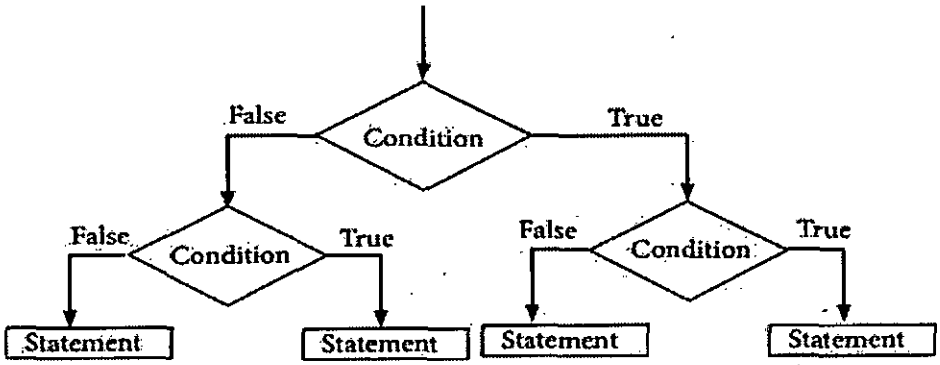
In the first syntax of the nested if mentioned above the expression-1 is evaluated and the expression result is false then control passes to statement-m. Otherwise, expression-2 is evaluated, if the condition is true, then Nested-True-block is executed, next statement-n is also executed. Otherwise Nested-False-Block, statement-n and statement-m are executed. The working procedure of the above said if. Else structures are given as flowchart below:



Flowchart 10.1 if nested inside if Part



Flowchart 10.2 If nested inside else part



Flowchart 10.3 If nested inside both if part and else part

Illustration 10.5 – C++ program to calculate commission according to grade using nested if statement

```
#include <iostream>
using namespace std;
int main()
{
    int sales, commission;
    char grade;
    cout << «\n Enter Sales amount: «;
    cin >> sales;
    cout << «\n Enter Grade: «;
    cin >> grade;
    if (sales > 5000)
    {
        commission = sales * 0.10;
        cout << «\n Commission: « << commission;
    }
    else
    {
        commission = sales * 0.05;
        cout << «\n Commission: « << commission;
    }
    cout << «\n Good Job ..... «;
    return 0;
}
```

Output:

Enter Sales amount: 6000

Enter Grade: A

Commission: 600

Good Job .....

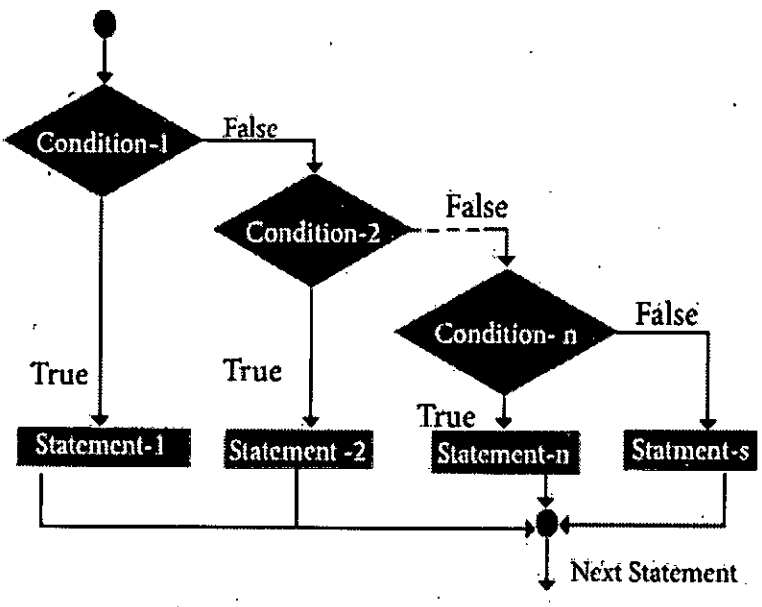
if -else-if ladder

The if-else ladder is a multi-path decision making statement. In this type of statement 'if' is followed by one or more else if statements and finally end with an else statement.

The syntax of if-else ladder:

```
if (expression 1)
{
    Statemet-1
}
else
    if( expression 2)
    {
       Statemet-2
    }
    else
       if ( expression 3)
       {
       Statemet-3
       }
       else
       {
            Statement-4
       }
```

When the respective expression becomes true, the statement associated with block is executed, and the rest of the ladder is bypassed. If none of the conditions is true, then the final else statement will be executed.



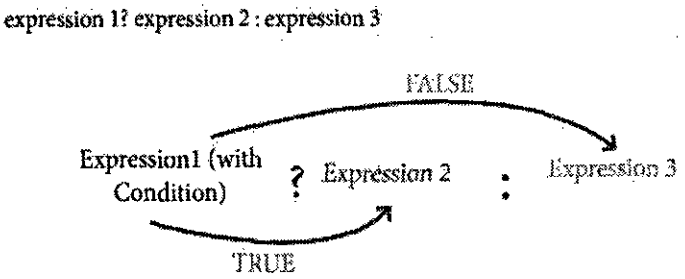
Flowchart 10.4 if-else ladder flow chart

Illustration 10.6 C++ program to find your grade using if-else ladder.

```
#include <iostream>
using namespace std;
int main ()
{
int marks;
cout<<" Enter the Marks :";
cin>>marks;
if( marks >= 60 )
    cout<< «Your grade is 1st class !!» <<endl;
      else if( marks >= 50 && marks < 60)
           cout<< «your grade is 2nd class !!» <<endl;
                else if( marks >= 40 && marks < 50)
                     cout<< «your grade is 3rd class !!» <<endl;
else
    cout<< «You are fail !!» <<endl;
return 0;
}
```

Output

Enter the Marks :60
Your grade is 1st class !!
When the marks are greater than or equal to 60, the message **"Your grade is 1st class !!"** is displayed and the rest of the ladder is bypassed. When the marks are between 50 and 59, the message **"Your grade is 2nd class !!"** is displayed, and the other ladder is bypassed. When the mark between 40 to 49, the message **"Your grade is 3nd class !!"** is displayed, otherwise, the message «You are fail !!» is displayed.
The ?: Alternative to if- else
The conditional operator (or Ternary operator) is an alternative for 'if else statement'. The conditional operator that consists of two symbols (?:). It takes three arguments. The control flow of conditional operator is shown below
The syntax of the conditional operator is:
expression 1? expression 2 : expression 3



In the above syntax, the expression 1 is a condition which is evaluated, if the condition is true (Non-zero), then the control is transferred to expression 2, otherwise, the control passes to expression 3.

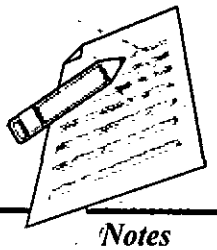


Illustration 10.7 – C++ program to find greatest of two numbers using conditional operator

```
#include <iostream>
using namespace std;
int main()
{
    int a, b, largest;
    cout << «\n Enter any two numbers: «;
    cin >> a >> b;
    largest = (a>b)? a : b;
    cout << «\n Largest number : « << largest;
    return 0;
}
```

Output:

```
Enter any two numbers: 12 98
Largest number : 98
```

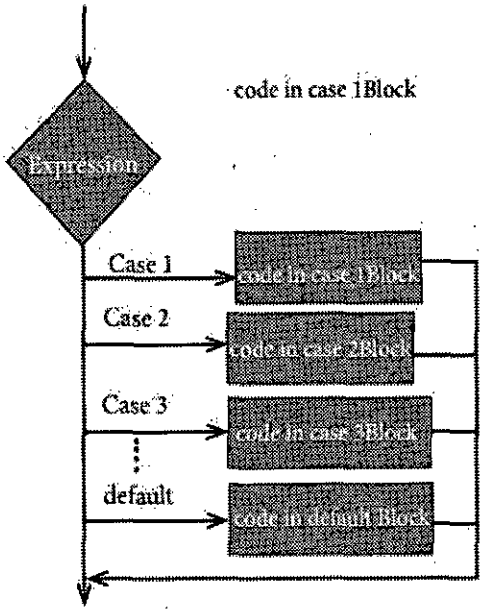
Switch statement

The switch statement is a multi-way branch statement. It provides an easy way to dispatch execution to different parts of code based on the value of the expression. The switch statement replaces multiple if-else sequences.

The syntax of the switch statement is;

```
switch(expression)
{
    case constant 1:
        statement(s);
        break;
    case constant 2:
        statement(s);
        break;
    .
    .
    .
    .
    default:
        statement(s);
}
```

In the above syntax, the expression is evaluated and if its value matches against the constant value specified in one of the case statements, that respective set of statements are executed. Otherwise, the statements under the default option are executed. The workflow of switch statement and flow chart are shown below.



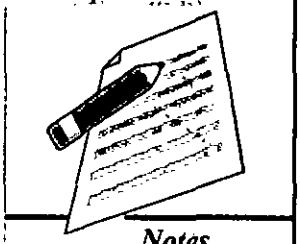
Flowchart10.5: workflow of switch and flow chart

Rules:

1. The expression provided in the switch should result in a constant value otherwise it would not be valid.
2. Duplicate case values are not allowed.
3. The default statement is optional.
4. The break statement is used inside the switch to terminate a statement sequence. When a break statement is reached, the switch terminates, and the flow of control jumps to the next line following the switch statement.
5. The break statement is optional. If omitted, execution will continue on into the next case. The flow of control will fall through to subsequent cases until a break is reached.
6. Nesting of switch statements is also allowed.

Illustration 10.8 – C++ program to demonstrate switch statement

```
#include <iostream>
using namespace std;
int main()
{
    int num;
    cout << «\n Enter week day number: «;
    cin >> num;
    switch (num)
    {
       case 1 : cout << «\n Sunday»; break;
       case 2 : cout << «\n Monday»; break;
```



Notes

```
        case 3 : cout << «\n Tuesday»; break;
        case 4 : cout << «\n Wednessday»; break;
        case 5 : cout << «\n Thursday»; break;
        case 6 : cout << «\n Friday»; break;
        case 7 : cout << «\n Saturday»; break;
        default: cout << «\n Wrong input....»;
    }
}
```

Output:

Enter week day number: 6

Friday

Illustration 10.9 – C++ program to demonstrate switch statement

```
#include <iostream>
using namespace std;
int main()
{
    char grade;
    cout << «\n Enter Grade: «;
    cin >> grade;
switch(grade)
    {
    case ‹A› : cout << «\n Excellent...»;
    break;
    case ‹B› :
    case ‹C› : cout << «\n Welldone ...»;
    break;
    case ‹D› : cout << «\n You passed ...»;
    break;
    case ‹E› : cout << «\n Better try again ...»;
    break;
    default : cout << «\n Invalid Grade ...»;
}
cout << "\n Your grade is " << grade;
return 0;
}
```

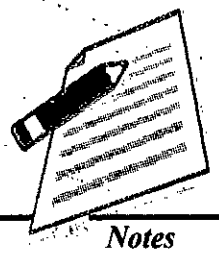
Output:

Enter Grade: C

Welldone ...

Your grade is C

Switch vs if-else



"if-else" and "switch" both are selection statements. The selection statements, transfer the flow of the program to the particular block of statements based upon whether the condition is "true" or "false". However, there are some differences in their operations. These are given below:

Key Differences Between if-else and switch

1. Expression inside if statement decide whether to execute the statements inside if block or under else block. On the other hand, expression inside switch statement decides which case to execute.
2. An if-else statement uses multiple statements for multiple choices. On other hand, switch statement uses single expression for multiple choices.
3. If-esle statement checks for equality as well as for logical expression. On the other hand, switch checks only for equality.
4. The if statement evaluates integer, character, pointer or floating-point type or Boolean type. On the other hand, switch statement evaluates only character or an integer data type.
5. Sequence of execution is like either statement under if block will execute or statements under else block statement will execute. On the other hand the expression in switch statement decide which case to execute and if do not apply a break statement after each case it will execute till the end of switch statement.
6. If expression inside if turn out to be false, statement inside else block will be executed. If expression inside switch statement turns out to be false then default statements are executed.
7. It is difficult to edit if-else statements as it is tedious to trace where the correction is required. On the other hand, it is easy to edit switch statements as they are easy to trace.

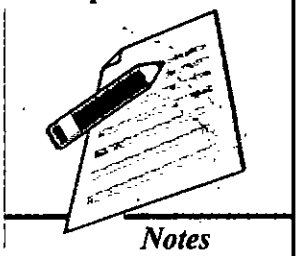
The if statement is more flexible than switch statement.

Some important things to know about switch

There are some important things to know about switch statement. They are

1. A switch statement can only work for quality of comparisons.
2. No two case labels in the same switch can have identical values.
3. If character constants are used in the switch statement, they are automatically converted to their equivalent ASCII codes.
4. The switch statement is more efficient choice than if in a situation that supports the nature of the switch operation.

Tips: The switch statement is more efficient than if-else statement.



Nested switch

When a switch is a part of the statement sequence of another switch, then it is called as nested switch statement. The inner switch and the outer switch constant may or may not be the same.

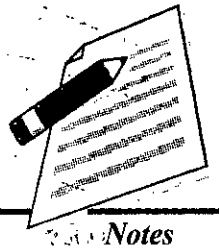
The syntax of the nested switch statement is;

```
switch (expression)
{
    case constant 1:
    statement(s);
    break;
        switch(expression)
        {
        case constant 1:
              statement(s);
                    break;
        case constant 2:
              statement(s);
              break;
              .
              .
              .
        default :
        statement(s);
    }
case constant 2:
statement(s);
break;
.
.
.
default :
statement(s);
}
```

The below program illustrates nested switch statement example. The outer switch checks for zero or non-zero and the inner switch checks for odd or even.

Illustration 10.10 C++ program to check for zero or non-zero and odd or even using nested switch statement

```
#include <iostream>
using namespace std;
int main()
```




```
{
int a = 8;
cout<<"The Number is : " <<a <<endl;
switch (a)
    {
    case 0 :
    cout<<»The number is zero» <<endl;
    break;
    default:
    cout<<»The number is a non-zero integer» <<endl;
    int b = a % 2;
    switch (b)
    {
    case 0:
    cout<<»The number is even» <<endl;
    break;
    case 1:
    cout<<»The number is odd» <<endl;
    break;
    }
    }
return 0;
}
```

Output

The Number is : 8

The number is a non-zero integer

The number is even

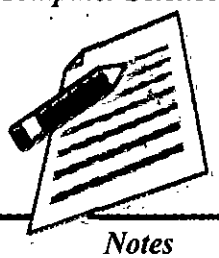
Parts of a loop

Every loop has four elements that are used for different purposes. These elements are

- Initialization expression
- Test expression
- Update expression
- The body of the loop

Initialization expression(s):

The control variable(s) must be initialized before the control enters into loop. The initialization of the control variable takes place under the initialization expressions. The initialization expression is executed only once in the beginning of the loop.



Notes

Test Expression:

The test expression is an expression or condition whose value decides whether the loop-body will be execute or not. If the expression evaluates to true (i.e., 1), the body of the loop executed, otherwise the loop is terminated.

In an entry-controlled loop, the test-expression is evaluated before the entering into a loop whereas in an exit-controlled loop, the test-expression is evaluated before exit from the loop.

Update expression:

It is used to change the value of the loop variable. This statement is executed at the end of the loop after the body of the loop is executed.

The body of the loop:

A statement or set of statements forms a body of the loop that are executed repetitively. In an entry-controlled loop, first the test-expression is evaluated and if it is nonzero, the body of the loop is executed otherwise the loop is terminated. In an exit-controlled loop, the body of the loop is executed first then the test-expression is evaluated. If the test-expression is true the body of the loop is repeated otherwise loop is terminated

for loop

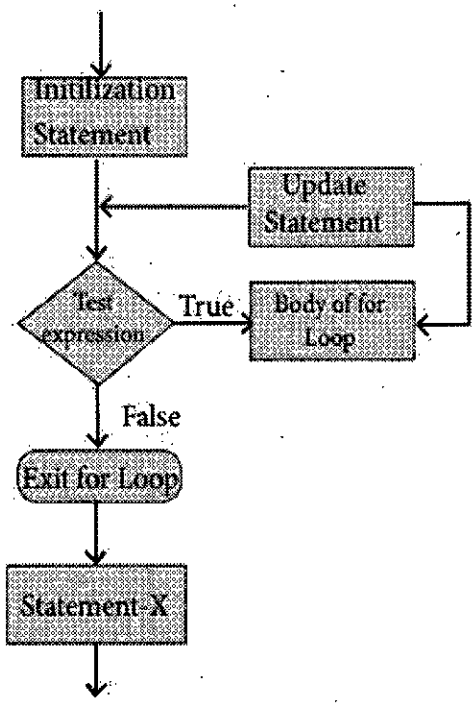
The for loop is the easiest looping statement which allows code to be executed repeatedly. It contains three different statements (initialization, condition or test-expression and update expression(s)) separated by semicolons.

The general syntax is:

```
for (initialization(s); test-expression; update expression(s))
{
    Statement 1;
    Statement 2;
    .....
}
```

Statement-x;

The initialization part is used to initialize variables or declare variable which are executed only once, then the control passes to test-expression. After evaluation of test-expression, if the result is false, the control transferred to statement-x. If the result is true, the body of the loop is executed, next the control is transferred to update expression. After evaluation of update expression part, the control is transferred to the test-expression part. Next the steps 3 to 5 are repeated. The workflow of for loop and flow chart are shown below.



Flowchart 10.6: Workflow of for loop and flow chart

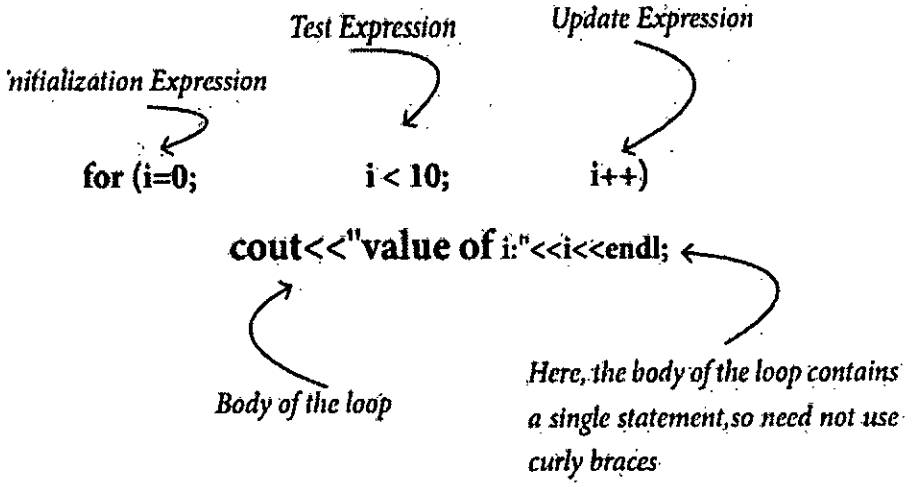
Illustration 10.11 C++ program to display numbers from 0 to 9 using for loop

```
#include <iostream>
using namespace std;
int main ()
{
int i;
for(i = 0; i< 10; i ++ )
    cout<< «value of i : « <<i<<endl;
return 0;
}
```

Output

```
value of i : 0
value of i : 1
value of i : 2
value of i : 3
value of i : 4
value of i : 5
value of i : 6
value of i : 7
value of i : 8
value of i : 9
```

The following lines describe the working of the above given for loop:



In the above program, first the variable i is initialized, next i is compared with 10, if i is less than ten, the value of i is incremented. In this way, the numbers 0 to 9 are displayed. Once i become 10, it is no longer < 10. So, the control comes out of the for loop.

Illustration 10.12 C++ program to sum the numbers from 1 to 10 using for loop

```
#include <iostream>
using namespace std;
int main ()
{
int i, sum=0;
for(i=1; i<=10;i++)
{
    sum=sum+i;
}
cout<<"The sum of 1 to 10 is "<<sum;
return 0;
}
```

Output

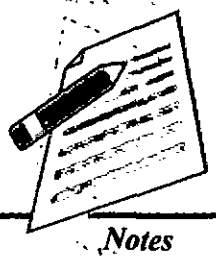
The sum of 1 to 10 is 55

Variations of for loop

The for is one of the most important looping statement in C++ because it allows a several variations. These variations increase the flexibility and applicability of for loop. These variations will be discussed below:

Multiple initialization and multiple update expressions

Multiple statements can be used in the initialization and update expressions of for loop. These multiple initialization and multiple update expressions are separated by commas. For example,




```
#include<iostream>
using namespace std;
int main()
{
        int i, j;
                for(i=0, j=10 ; i<j  ;  i++,j--)
                {
                cout<<"\nThe value of i is"<<i<<" The value of j is "j;

}
return 0;
}
```

Multiple initialization expressions (separated by commas)

Multiple update expressions (separated by commas)

Output

The value of i is 0 The value of j is 10

The value of i is 1 The value of j is 9

The value of i is 2 The value of j is 8

The value of i is 3 The value of j is 7

The value of i is 4 The value of j is 6

In the above example, the initialization part contains two variables i and j and update expression contains i++ and j++. These two variables are separated by commas which is executed in sequential order i.e., during initialization firstly i=0 followed by j=10. Similarly, in update expression, firstly i++ is evaluated followed by j++ is evaluated.

Prefer prefix operator over postfix

Generally, the update expression contains increment/decrement operator (++ or --). In this part, always prefer prefix increment/decrement operator over postfix when to be used alone. The reason behind this is that when used alone, prefix operators are executed faster than postfix.

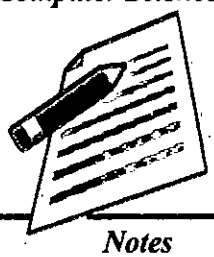
Optional expressions

Generally, the for loop contains three parts, i.e., initialization expressions, test expressions and update expressions. These three expressions are optional in a for loop.

Case 1

Illustration 10.13 (a) C++ program to sum the numbers from 1 to 10

```
#include <iostream>
using namespace std;
int main ()
{
int i, sum=0; n;
cout<<"\n Enter The value of n";
cin>>n;
```




```
i =1;
for ( ; i<=10;i++)
    {
    sum += i;
   }
cout<<"\n The sum of 1 to " <<n<<"is "<<sum;
return 0;
}
```

Output

Enter the value of n 5

The sum of 1 to 5 is 15

In the above example, the variable i is declared and sum is initialized at the time of variable declaration. The variable i is assigned to 0 before the for loop but still the semicolon is necessary before test expression. In a for loop, if the initialization expression is absent then the control is transferred to test expression/conditional part.

Case 2

Illustration 10.13 (b) C++ program to sum the numbers from 1 to 10

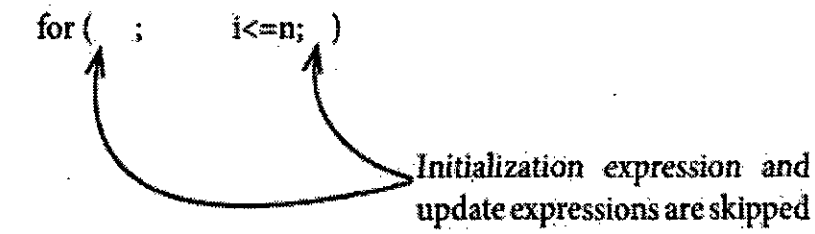
```
#include <iostream>
using namespace std;
int main ()
{
int i, sum=0, n;
cout<<"\n Enter The value of n";
cin>>n;
i =1;
for ( ; i<=10; )
{
    sum += i;
    ++i;
}
cout<<"\n The sum of 1 to " <<n<<"is "<<sum;
return 0;
}
```

Output

Enter the value of n 5

The sum of 1 to 5 is 15

In the above code, the update expression is not done, but a semicolon is necessary before the update expression.

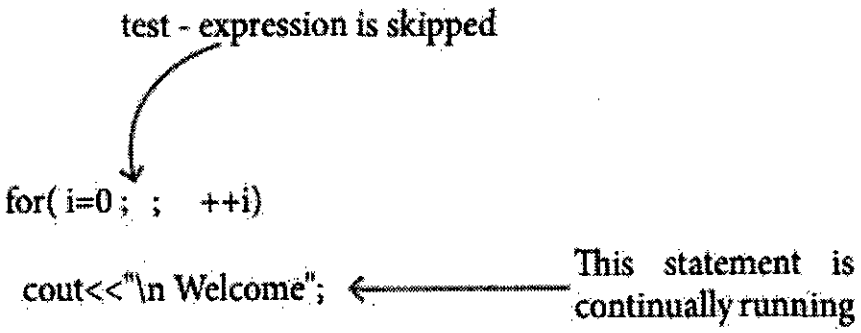


In the above code, neither the initialization nor the update expression is done in the for loop.

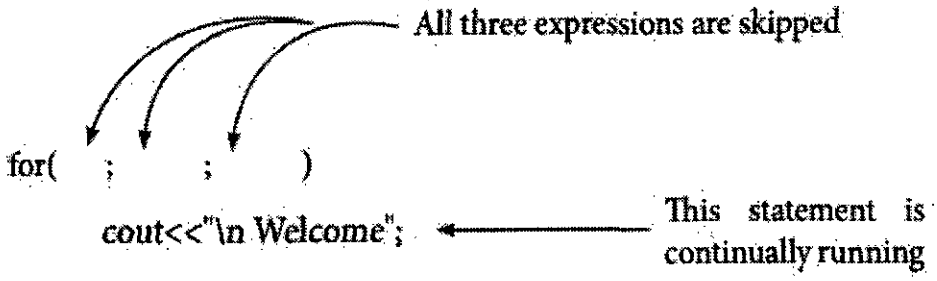
If both or any one of expressions are absent then the control is transferred to conditional part.

Case 3

An infinite loop will be formed if a test-expression is absent in a for loop. For example,



Similarly, the following for loop also forms an infinite loop.

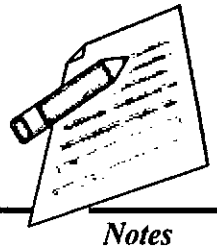


Empty loop

Empty loop means a loop has no statement in its body is called an empty loop. Following for loop is an empty loop:

In the above code, the for loop contains a null statement, it is an empty loop.

Similarly, the following for loop also forms an empty loop.



Notes

```
int i;
for( i=0      ;      i<=5;   ++i);   <-- The body of for loop contains a null statement
 {                                   <-- The body of for loop is not executed because semicolon(;) is ended at the end of for loop.
        cout<<"\nWe are Indians";
 }
```

In the above code, the body of a for loop enclosed by braces is not executed at all because a semicolon is ended after the for loop.

Declaration of variable in a for loop

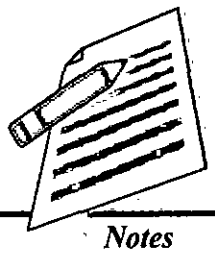
In C++, the variables can also be declared within a for loop. For instance,

```
int main ()
{
  int sum = 0;          <-- Variable (i)is declared within the for loop.
  for(int i=0; i<=5; ++i )
  {
       sum = sum + i;   <-- The variable i can be accessed only within the body of loop.
  }
  cout<<"\nThe variable i cannot access here";
  cout<<"\n The variable sum can access here";
}
```

A variable declared inside the block of main () can be accessed anywhere inside main() i.e., the scope of variable in main()

SUMMARY

Statements are the instructions given to the computer to perform any kind of action. Compound statement is a grouping of statements in which each individual statement ends with a semicolon. Sometimes we require a set of statements to be executed a number of times by changing the value of one or more variables each time to obtain a different result. This type of program execution is called looping. In the while loop, the loop repetition test is performed before each execution of the loop body; the loop body is not executed at all if the initial test fails. In the do-while loop, the loop termination test is performed after each execution of the loop body. Hence the loop body is always executed at least once. Continue statement is used in loops and causes a program to skip the rest of the body of the loop. The execution of the program can be stopped at any point with exit() statement.



EXERCISE

MCQ

1. What is the alternate name of null statement?
   (A) No statement **(B) Empty statement**
   (C) Void statement (D) Zero statement

2. In C++, the group of statements should enclosed within:
   (A){} (B)[]
   (C)() (D)<>

3. The set of statements that are executed again and again in iteration is called as:
   (A) condition **(B) loop**
   (C) statement (D) body of loop

4. The multi way branching statement:
   (A) if (B) if ... else
   (C) switch (D) for

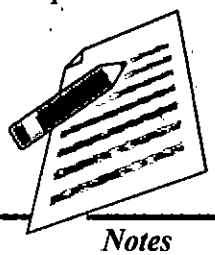
5. How many types of iteration statements?
   (A) 2 **(B) 3**
   (C) 4 (D) 5

6. How many times the following loop will execute? for (int i=0; i<10; i++)
   (A) 0 **(B) 10**
   (C) 9 (D) 11

7. Which of the following is the exit control loop?
   (A) for (B) while
   (C) do...while (D) if...else

8. Identify the odd one from the keywords of jump statements:
   (A) break **(B) switch**
   (C) go to (D) continue

9. Which of the following is the exit control loop?
   (A) do-while (B) for
   (C) while (D) if-else



10. A loop that contains another loop inside its body:

    (A) Nested loop (B) Inner loop

    (C) Inline loop (D) Nesting of loop

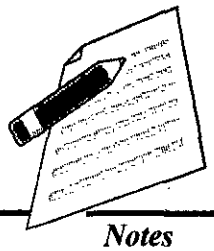
Review Questions

1. Write a program to display the following menu and accept a choice number. If an invalid choice is entered, then appropriate error message must be displayed. Otherwise the choice number entered must be displayed.

   MENU

   1. Create a data file 2. Display a data file
   3. Edit a data file 4. Exit Choice.

2. Write a program that will accept a mark and assign letter grade according to the following table. Grade Mark A > = 90, B > = 80 but < 90, C > = 70 but < 80, D > = 60 but < 70, F < 60.
3. Write a program that will print sum of N integers entered by the user.
4. Write a program to generate the members of fibonacci series upto 500.
5. Write a program to reverse a given number.
6. Write a program that will print the factorial of any number.
8. Write a program to print all the tables between 2 and 20 upto 10.



Notes

10 FUNCTIONS

- Understand the concept of functions.
- Discuss the features of functions.
- Describe the types of functions.
- Discuss the applications of functions.

Objective of the chapter:

The basic objective of this chapter is to through some light on the initial concepts of functions so that the types and applications of functions can be learned.

Introduction

The most important criteria in writing and evaluating the algorithm is the time it takes to complete a task. To have a meaningful comparison of algorithms, the duration of computation time must be independent of the programming language, compiler, and computer used. As you aware that algorithms are expressed using statements of a programming language. If a bulk of statements to be repeated for many numbers of times then subroutines are used to finish the task.

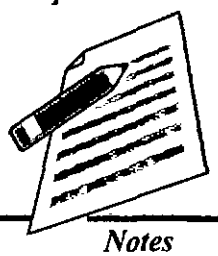
Subroutines are the basic building blocks of computer programs. Subroutines are small sections of code that are used to perform a particular task that can be used repeatedly. In Programming languages these subroutines are called as Functions.

A large program can typically be split into small sub-programs (blocks) called as functions where each sub-program can perform some specific functionality. Functions reduce the size and complexity of a program, makes it easier to understand, test, and check for errors. The functions which are available by default known as **"Built-in"** functions and user can create their own functions known as **"User-defined"** functions.

- Built-in functions – Functions which are available in C++ language standard library.
- User-defined functions – Functions created by users.

Need for Functions

To reduce size and complexity of the program we use Functions. The programmers can make use of sub programs either writing their own functions or calling them from standard library.



Notes

1. Divide and Conquer

- Complicated programs can be divided into manageable sub programs called functions.
- A programmer can focus on developing, debugging and testing individual functions.
- Many programmers can work on different functions simultaneously.

2. Reusability:

- Few lines of code may be repeatedly used in different contexts. Duplication of the same code can be eliminated by using functions which improves the maintenance and reduce program size.
- Some functions can be called multiple times with different inputs.

Types of Functions

Functions can be classified into two types,

1. Pre-defined or Built-in or Library Functions
2. User-defined Function.

C++ provides a rich collection of functions ready to be used for various tasks. The tasks to be performed by each of these are already written, debugged and compiled, their definitions alone are grouped and stored in files called **header files.** Such ready-to-use sub programs are called **pre-defined functions or built-in functions.**

C++ also provides the facility to create new functions for specific task as per user requirement. The name of the task and data required (arguments) are decided by the user and hence they are known as **User-defined functions.**

C++ Header Files and Built-in Functions

Header files provide function prototype and definitions for library functions. Data types and constants used with the library functions are also defined in them. A header file can be identified by their file extension **.h.** A single header file may contain multiple built-in functions. For example: **stdio.h** is a header file contains pre-defined **"standard input/output"** functions.

Standard input/output (stdio.h)

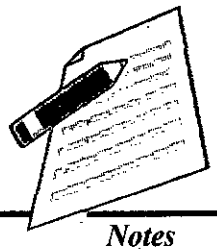
This header file defines the standard I/O predefined functions **getchar(), putchar(), gets(), puts()** and etc.

1. getchar() and putchar() functions

The predefined function **getchar()** is used to get a single character from keyboard and **putchar()** function is used to display it.

Program 11.1 C++ code to accept a character and displays it

```
#include<iostream>
#include<stdio.h>
```




```
using namespace std;
int main()
{
cout<<"\n Type a Character : ";
char ch = getchar();
 cout << "\n The entered Character is: ";
  putchar(ch);
  return 0;
}
```

Output:

Type a Character : T

The entered Character is: T

2. gets() and puts() functions

Function gets() reads a string from standard input and stores it into the string pointed by the variable. Function **puts()** prints the string read by **gets()** function in a newline.

Program 11.2 C++ code to accepts and display a string

```
#include<iostream>
#include<stdio.h>
using namespace std;
int main()
{
    char str[50];
    cout<<»Enter a string : «;
    gets(str);
    cout<<»You entered: «
    puts(str);
    return(0);
}
```

Output :

Enter a string : Computer Science

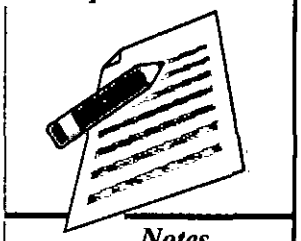
You entered: Computer Science

Character functions (ctype.h)

This header file defines various operations on characters. Following are the various character functions available in C++. The header file **ctype.h** is to be included to use these functions in a program.

1. isalnum()

This function is used to check whether a character is **alphanumeric or not.** This function returns non-zero value if c is a digit or a letter, else it returns 0.



Notes

Syntax:

int isalnum (char c)

Example :

int r = isalnum('5');

cout << isalnum('A') <<'\t'<<r;

But the statements given below assign 0 to the variable n, since the given character is neither an alphabet nor a digit.

char c = '$';

int n = isalnum(c);

cout<<c;

Output:

0

Program 11.3

```
#include<iostream>
#include<stdio.h>
#include<ctype.h>
using namespace std;
int main()
{
 char ch;
 int r;
  cout<<"\n Type a Character :";
  ch = getchar();
  r = isalnum(ch);
cout<<"\nThe Return Value of isalnum(ch) is :"<<r;
}
```

Output-1:

Type a Character :A

The Return Value of isalnum(ch) is :1

Output-2:

Type a Character :?

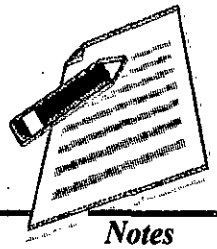
The Return Value of isalnum(ch) is :0

2. isalpha()

The isalpha() function is used to check whether the given character is an alphabet or not.

Syntax:

int isalpha(char c);



This function will return 1 if the given character is an alphabet, and 0 otherwise 0. The following statement assigns 0 to the variable n, since the given character is not an alphabet.

int n = isalpha('3');

But, the statement given below displays 1, since the given character is an alphabet.

cout << isalpha('a');

Program 11.4

```
#include<iostream>
#include<stdio.h>
#include<ctype.h>
using namespace std;
int main()
{
 char ch;
 cout << "\n Enter a charater: ";
 ch = getchar();
  cout<<"\n The Return Value of isalpha(ch) is :" << isalpha(ch) ;
}
```

Output-1:

Enter a charater: A

The Return Value of isalpha(ch) is :1

Output-2:

Enter a charater: 7

The Return Value of isalpha(ch) is :0

3. isdigit()

This function is used to check whether a given character is a digit or not. This function will return 1 if the given character is a digit, and 0 otherwise.

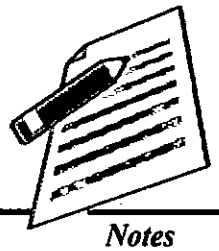
Syntax:

int isdigit(char c);

When the following program is executed, the value of the variable n will be 1,since the given character is not a digit.

Program 11.5

```
using namespace std;
#include<iostream>
#include<ctype.h>
int main()
```




```
{
 char ch;
 cout << "\n Enter a Character: ";
 cin >> ch;
  cout<<"\n The Return Value of isdigit(ch) is :" << isdigit(ch) ;
}
```

Output-1

Enter a Character: 3

The Return Value of isdigit(ch) is :1

Output-2

Enter a Character: A

The Return Value of isdigit(ch) is :0

***Return 0; (Not Compulsory in latest compilers)**

4. islower()

This function is used to check whether a character is in lower case (small letter) or not. These functions will return a non-zero value, if the given character is a lower case alphabet, and 0 otherwise.

Syntax:

int islower(char c);

After executing the following statements, the value of the variable n will be 1 since the given character is in lower case.

char ch = 'n';

int n = islower(ch);

But the statement given below will assign 0 to the variable n, since the given character is an uppercase alphabet.

int n = islower('P');

5. isupper()

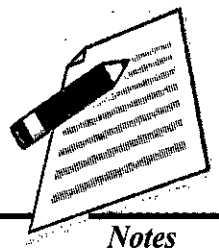
This function is used to check the given character is uppercase. This function will return 1 if true otherwise 0. For the following examples value **1** will be assigned to n and 0 for m.

int n=isupper('A');

int m=isupper('a');

6. toupper()

This function is used to convert the given character into its uppercase. This function will return the upper case equivalent of the given character. If the given character itself is in upper case, the output will be the same.



Syntax:

char toupper(char c);

The following statement will assign the character constant 'K' to the variable c.

char c = toupper('k');

But, the output of the statement given below will be 'B' itself.

cout <<toupper('B');

7. tolower()

This function is used to convert the given character into its lowercase. This function will return the lower case equivalent of the given character. If the given character itself is in lower case, the output will be the same.

Syntax:

char tolower(char c);

The following statement will assign the character constant 'k' to the variable c.

char c = tolower('K');

But, the output of the statement given below will be 'b' itself.

cout <<tolower('b');

String manipulation (string.h)

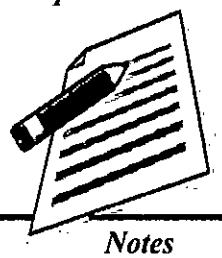
The library string.h (also referred as cstring) has several common functions for dealing with strings stored in arrays of characters. The string.h header file to be included before using any string function.

1. strcpy()

The **strcpy()** function takes two arguments: target and source. It copies the character string pointed by the source to the memory location pointed by the target. The null terminating character **(\0)** is also copied.

Program 11.6

```
#include <string.h>
#include <iostream>
using namespace std;
int main()
{
char source[] = "Computer Science";
char target[20]="target";
cout<<"\n String in Source Before Copied :"<<source;
 cout<<"\n String in Target Before Copied :"<<target; strcpy(target,source);
  cout<<"\n String in Target After strcpy function Executed :"<<target; return 0;
}
```



Output:

String in Source Before Copied :Computer Science
String in Target Before Copied :target
String in Target After strcpy function Executed :Computer Science

2. strlen()

The **strlen()** takes a null terminated byte string source as its argument and returns its length. The length does not include the null(\0) character.

Program 11.7

```
#include <string.h>
#include <iostream>
using namespace std;
int main()
{
char source[ ] = "Computer Science";
cout<<"\nGiven String is "<<source<<" its Length is "<<strlen(source);
return 0;
}
```

Output:

Given String is Computer Science its Length is 16

3. strcmp()

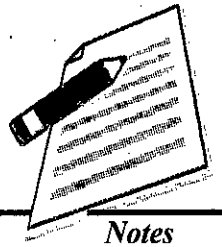
The **strcmp()** function takes two arguments: string1 and string2. It compares the contents of string1 and string2 lexicographically.

The strcmp() function returns a:

- Positive value if the first differing character in string1 is greater than the corresponding character in string2. (ASCII values are compared)
- Negative value if the first differing character in string1 is less than the corresponding character in string2.
- 0 if string1 and string2 are equal.

Program 11.8

```
#include <string.h>
#include <iostream>
using namespace std;
int main()
{
char string1[] = "Computer";
```




```
char string2[] = "Science";
 int result;
 result = strcmp(string1,string2);
 if(result==0)
 {
cout<<"String1 : "<<string1<<" and String2 : "<<string2 <<"Are Equal";
 }
 if (result<0)
 {
 cout<<"String1 :"<<string1<<" and String2 : "<<string2 <<" Are Not Equal";
 }
}
```

Output

String1 : Computer and String2 : Science Are Not Equal

4. strcat()

The **strcat()** function takes two arguments: target and source. This function appends copy of the character string pointed by the source to the end of string pointed by the target.

Program 11.9

```
#include <string.h>
#include <iostream>
using namespace std;
int main()
{
char target[50] = "Learning C++ is fun";
 char source[50] = " , easy and Very useful";
 strcat(target, source);
 cout << target ;
return 0;
}
```

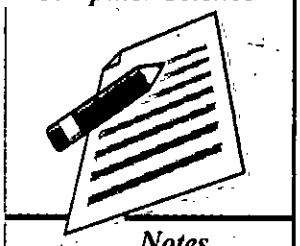
Output

Learning C++ is fun , easy and Very useful

5. strupr()

The **strupr()** function is used to convert the given string into Uppercase letters.

Program 11.10




```
using namespace std;
#include<iostream>
#include<ctype.h>
#include<string.h>
int main()
{
 char str1[50];
  cout<<"\nType any string in Lower case :";
  gets(str1);
 cout<<"\n Converted the Source string "<<str1<<into Upper Case is "<<strupr(str1);
return 0;
}
```

Output:

Type any string in Lower case : computer science

Converted the Source string computer science into Upper Case is COMPUTER SCIENCE

6. strlwr()

The **strlwr()** function is used to convert the given string into Lowercase letters.

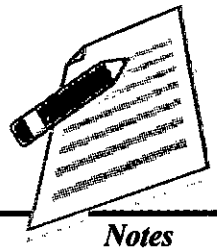
Program 11.11

```
using namespace std;
#include<iostream>
#include<ctype.h>
#include<string.h>
int main()
{
 char str1[50];
 cout<<"\nType any string in Upper case :";
 gets(str1);
 cout<<"\n Converted the Source string "<<str1<<into Lower Case is "<<strlwr(str1);
}
```

Output:

Type any string in Upper case : COMPUTER SCIENCE

Converted the Source string COMPUTER SCIENCE into lower Case is computer science



Mathematical functions (math.h)

Most of the mathematical functions are defined in math.h header file which includes basic mathematical functions.

1. cos() function

The cos() function takes a single argument in radians. The cos() function returns the value in the range of [-1, 1]. The returned value is either in double, float, or long double.

Program 11.12

```
#include <iostream>
#include <math.h>
using namespace std;
int main()
{
 double x = 0.5, result;
 result = cos(x);
 cout << "COS("<<x<<")= "<<result;
}
```

Output:

```
COS(0.5)= 0.877583
```

2. sqrt() function

The sqrt() function returns the square root of the given value of the argument. The sqrt() function takes a single non-negative argument. If a negative value is passed as an argument to sqrt() function, a domain error occurs.

Program 11.13

```
#include <iostream>
#include <math.h>
using namespace std;
int main()
{
 double x = 625, result;
 result = sqrt(x);
 cout << "sqrt("<<x<<") = "<<result;
 return 0;
}
```

Notes

Output:

sqrt(625) = 25

3. sin() function

The **sin()** function takes a single argument in radians. The **sin()** function returns the value in the range of [-1, 1]. The returned value is either in double, float, or long double.

4. pow() function

The **pow()** function returns base raised to the power of exponent. If any argument passed to **pow()** is long double, the return type is promoted to long double. If not, the return type is double. The **pow()** function takes two arguments:

base - the base value

exponent - exponent of the base

Program 11.14

```
#include <iostream>
#include <math.h>
using namespace std;
int main()
{
 double base, exponent, result;
 base = 5;
  exponent = 4;
  result = pow(base, exponent);
 cout << "pow("<<base << "^" << exponent << ") = " << result;
 double x = 25;;
 result = sin(x);
 cout << "\nsin("<<x<<")= "<<result;
  return 0;
}
```

Output:

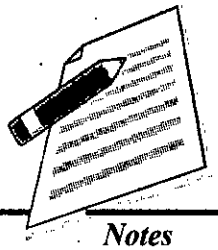
pow(5^4) = 625

sin(25)= -0.132352

Generating Random Numbers

The **srand()** function in C++ seeds the pseudo random number generator used by the **rand()** function. The seed for **rand()** function is 1 by default. It means that if no **srand()** is called before **rand()**, the **rand()** function behaves as if it was seeded with **srand(1)**. The **srand()** function takes an unsigned integer as its parameter which is used as seed by the **rand()** function. It is defined in<**cstdlib**>or <**stdlib.h**>header file.

Program 11.15




```
#include<iostream>
#include<cstdlib.h>
using namespace std;
int main()
{
 int random = rand(); /* No srand() calls before rand(), so seed = 1*/
 cout << "\nSeed = 1, Random number = " << random; srand(10);
  /* Seed = 10 */
   random = rand();
   cout << "\n\nSeed = 10, Random number = " << random;
   return 0;
}
```

OUTPUT :

Seed = 1, Random number = 41
Seed = 10, Random number = 71

User-defined Functions

We can also define new functions to perform a specific task. These are called as user-defined functions. User-defined functions are created by the user. A function can optionally define input parameters that enable callers to pass arguments into the function. A function can also optionally return a value as output. Functions are useful for encapsulating common operations in a single reusable block, ideally with a name that clearly describes what the function does.

Function Definition

In C++, a function must be defined before it is used anywhere in the program. The general syntax of a function definition is:

Return_Data_Type Function_name(parameter list)

{

Body of the function

}

Note:

1. The Return_Data_Type is any valid data type of C++.
2. The Function_name is a user-defined identifier.
3. The parameter list, which is optional, is a list of parameters, i.e. a list of variables preceded by data types and separated by commas.
4. The body of the function comprises C++ statements that are required to perform the intended task of this function.

Function Prototype

C++ program can contain any number of functions. But, it must always have **only one main() function** to begin the program execution. We can write the definitions of functions in any order as we wish. We can define the main() function first and all other functions after that or we can define all the needed functions prior to main(). Like a variable declaration, a function must be declared before it is used in the program. The declaration statement may be given outside the main() function.

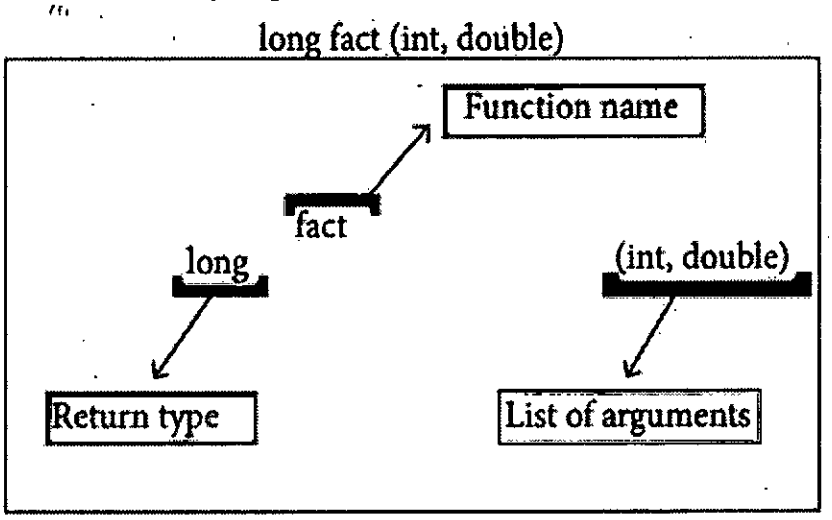


Figure 11.1

The prototype above provides the following information to the compiler:

- The return value of the function is of type long.
- **fact** is the name of the function.
- the function is called with two arguments:

The first argument is of int **data** type.

The second argument is of **double** data type.

int display(int , int) // function prototype//

The above function prototype provides details about the return data type, name of the function and a list of formal parameters or arguments.

Use of void command

void type has two important purposes:

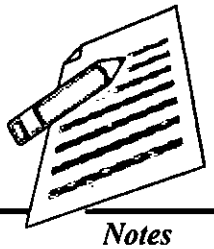
- To indicate the function does not return a value
- To declare a generic pointer.

void data type indicates the compiler that the function does not return a value, or in a larger context void indicates that it holds nothing.

For Example:

void fun(void)

The above function prototype tells compiler that the function **fun()** neither receives values from calling program nor return a value to the calling program.



Accessing a function

The user-defined function should be called explicitly using its name and the required arguments to be passed. The compiler refers to the function prototype to check whether the function has been called correctly. If the argument type does not match exactly with the data type defined in the prototype, the compiler will perform type conversion, if possible. If type conversion is impossible, the compiler generates an error message.

Example :

1. **display() :** calling the function without a return value and without any argument
2. **display (x, y) :** calling the function without a return value and with arguments
3. **x = display() :** calling the function with a return value and without any argument
4. **x = display (x, y) :** calling the function with a return value and with arguments

1. Formal Parameters and Actual Parameters or Arguments

Arguments or parameters are the means to pass values from the calling function to the called function. The variables used in the function definition as parameters are known as formal parameters. The constants, variables or expressions used in the function call are known as actual parameters.

1	display()	calling the function without a return value and without any argument
2	display (x, y)	calling the function without a return value and with arguments
3	x = display()	calling the function with a return value and without any argument
4	x = display (x, y)	calling the function with a return value and with arguments

2. Default arguments

In C++, one can assign default values to the formal parameters of a function prototype. The Default arguments allows to omit some arguments when calling the function.

When calling a function,

- For any missing arguments, complier uses the values in default arguments for the called function.
- The default value is given in the form of variable initialization.

Example : **void default value(int n1=10, n2=100);**

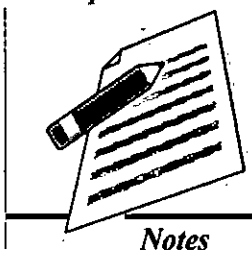
- The default arguments facilitate the function call statement with partial or no arguments. Example : **defaultvalue(x, y);**

defaultvalue(200,150);

defaultvalue(150);

defaultvalue(x,150);

The default values can be included in the function prototype from right to left, i.e., we cannot have a default value for an argument in between the argument list.



Example : **void defaultvalue(int n1=10, n2);//invalid prototype**
void defaultvalue(int n1, n2 = 10);//valid prototype

3. Constant Arguments

The constant variable can be declared using **const** keyword. The **const** keyword makes variable value stable. The constant variable should be initialized while declaring. The **const** modifier enables to assign an initial value to a variable that cannot be changed later inside the body of the function.

Syntax :

<returntype><functionname> (const <datatype variable=value>)

Example:

- int minimum(const int a=10);
- float area(const float pi=3.14, int r=5);

Program 11.16

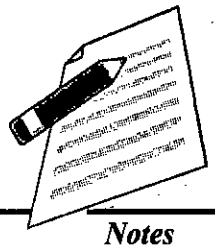
```
#include <iostream>
using namespace std;
double area(const double r,const double pi=3.14)
{
 return(pi*r*r);
}
int main ()
{
 double rad,res;
 cout<<"\nEnter Radius :";
  cin>>rad;
  res=area(rad);
 cout << "\nThe Area of Circle ="<<res;
return 0;
}
```

Output:

```
Enter Radius :5
The Area of Circle =78.5
```

If the variable value "r" is changed as **r=25;** inside the body of the function **"area"** then compiler will throw an error as **"assignment of read-only parameter 'r'"**

```
double area(const double r,const double pi=3.14)
{
 r=25;
return(pi*r*r);
}
```



Methods of calling functions

In C++, the arguments can be passed to a function in two ways. Based on the method of passing the arguments, the function calling methods can be classified as Call by Value method and Call by Reference or Address method.

Call by value Method

This method copies the value of an actual parameter into the formal parameter of the function. In this case, changes made to formal parameter within the function will have no effect on the actual parameter.

Program 11.17

```
#include<iostream>
using namespace std;
void display(int x)
{
int a=x*x;
cout<<"\n\nThe Value inside display function (a * a):"<<a;
}
int main()
{
 int a;
cout<<"\nExample : Function call by value:";
 cout<<"\n\nEnter the Value for A :";
 cin>>a;
  display(a);
  cout<<"\n\nThe Value inside main function "<<a;
 return(0);
}
```

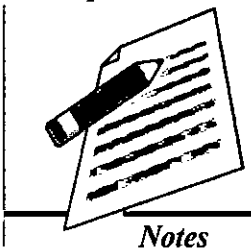
Output :

```
Example : Function call by value
Enter the Value for A : 5
The Value inside display function (a * a) : 25
The Value inside main function 5
```

Call by reference or address Method

This method copies the address of the actual argument into the formal parameter. Since the address of the argument is passed ,any change made in the formal parameter will be reflected back in the actual parameter.

Program 11.18




```
#include<iostream>
using namespace std;
void display(int &x) //passing address of a//
{
 x=x*x;
 cout<<"\n\nThe Value inside display function (n1 x n1) :"<<x ;
}
int main()
{
int n1;
cout<<"\nEnter the Value for N1 :";
cin>>n1;
cout<<"\nThe Value of N1 is inside main function Before passing : "<< n1; display(n1);
cout<<"\nThe Value of N1 is inside main function After passing (n1 x n1) : "<< n1;
return(0);
}
```

Output :

Enter the Value for N1 :45

The Value of N1 is inside main function Before passing : 45

The Value inside display function (n1 x n1) :2025

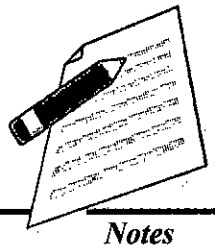
The Value of N1 is inside main function After passing (n1 x n1) : 2025

Note that the only change in the **display()** function is in the function header. The **&** symbol in the declaration of the parameter **x** means that the argument is a reference variable and hence the function will be called by passing reference. Hence when the argument **num1** is passed to the **display()** function, the variable **x** gets the address of **num1** so that the location will be shared. In other words, the variables **x** and **num1** refer to the same memory location. We use the name **num1** in the main() function, and the name **x** in the **display()** function to refer the same storage location. So, when we change the value of **x**, we are actually changing the value of **num1**.

Inline function

Normally the call statement to a function makes a compiler to jump to the functions (the definition of the functions are stored in STACKS) and also jump back to the instruction following the call statement. This reduces the speed of program execution. Inline functions can be used to reduce the overheads like STACKS for small function definition.

An inline function looks like normal function in the source file but inserts the function's code directly into the calling program. To make a function inline, one has to insert the keyword **inline** in the function header.



Syntax :

inline return type function name(datatype parametername1, ... datatype parameter nameN)

Advantages of inline functions:

- Inline functions execute faster but requires more memory space.
- Reduce the complexity of using STACKS.

Program 11.19

```
#include <iostream>
using namespace std;
inline float simple interest(float p1,float n1, float r1)
{
float si1=(p1*n1*r1)/100;
return(si1);
}
int main ()
{
float si,p,n,r;
cout<<"\nEnter the Principle Amount Rs. :";
 cin>>p;
  cout<<"\nEnter the Number of Years :";
  cin>>n;
  cout<<"\nEnter the Rate of Interest :";
  cin>>r;
 si=simple interest(p,n,r);
  cout << "\nThe Simple Interest = Rs."<<si;
 return 0;
}
```

Output:

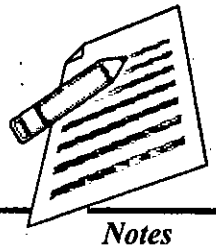
Enter the Principle Amount Rs. :60000

Enter the Number of Years :10

Enter the Rate of Interest :5

The Simple Interest = Rs.30000

Though the above program is written in the normal function definition format during compilation the function code **(p1*n1*r1)/100** will be directly inserted in the calling statement i.e.**si=simple interest(p,n,r);** this makes the calling statement to change as **si= (p1*n1*r1)/100;**



Different forms of User-defined Function declarations

1. A Function without return value and without parameter

The following program is an example for a function with no return and no arguments passed .

The name of the function is **display()**, its return data type is void and it does not have any argument.

Program 11.20

```
#include<iostream>
using namespace std;
void display()
{
cout<<"First C++ Program with Function";
}
int main()
{
display(); // Function calling statement//
return(0);
}
```

Output :

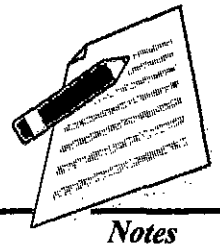
```
First C++ Program with Function
```

2. A Function with return value and without parameter

The name of the function is **display()**, its return type is int and it does not have any argument. The **return** statement returns a value to the calling function and transfers the program control back to the calling statement.

Program 11.21

```
#include<iostream>
using namespace std;
int display()
{
 int a, b, s;
 cout<<"Enter 2 numbers: ";
 cin>>a>>b;
  s=a+b;
 return s;
}
int main()
{
 int m=display();
```




```
 cout<<"\nThe Sum="<<m;
return(0);
}
```

Output :

Enter 2 numbers: 10 30

The Sum=40

3. A Function without return value and with parameter

The name of the function is **display()**, its return type is void and it has two parameters or arguments **x** and **y** to receive two values. The **return** statement returns the control back to the calling statement.

Program 11 .22

```
#include<iostream>
using namespace std;
void display(int x, int y)
{
 int s=x+y;
  cout<<"The Sum of Passed Values: "<<s;
}
int main()
{
  int a,b;
  cout<<"\nEnter the First Number      :";
   cin>>a;
   cout<<"\nEnter the Second Number :";
   cin>>b;
   display(a,b);
   return(0);
}
```

Output :

Enter the First Number :50

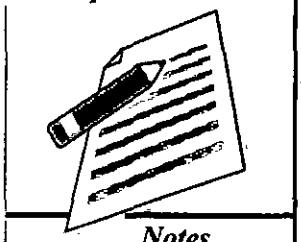
Enter the Second Number :45

The Sum of Passed Values: 95

4. A Function with return value and with parameter

The name of the function is display(), its return type is int and it has two parameters or arguments **x** and **y** to receive two values. The return statement returns the control back to the calling statement.

Program 11.23



Notes

```
#include<iostream>
using namespace std;
int display(int x, int y)
{
 int s=x+y;
   return s;
}
int main()
{ int a,b;
 cout<<"\nEnter the First Number
 cin>>a;
 cout<<"\nEnter the Second Number :";
 cin>>b;
 int s=display(a,b);
cout<<"\nExample: Function with Return Value and with Arguments";
cout<<"\nThe Sum of Passed Values: "<<s; return(0);
}
```

Output :

Enter the First Number :45

Enter the Second Number :67

Example: Function with Return Value and with Arguments

The Sum of Passed Values: 112

Returning from function

Returning from the function is done by using the **return** statement.

The **return** statement stops execution and returns to the calling function. When a **return** statement is executed, the function is terminated immediately at that point.

The return statement

The **return** statement is used to return from a function. It is categorized as a jump statement because it terminates the execution of the function and transfer the control to the called statement. A **return** may or may not have a value associated with it. If return has a value associated with it, that value becomes the return value for the calling statement. Even for void function return statement without parameter can be used to terminate the function.

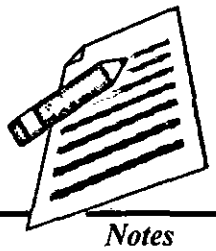
Syntax:

return expression/variable;

Example : return(a+b); return(a);

return; // to terminate the function

The Returning values:



The functions that return no value is declared as void. The data type of a function is treated as int, if no data type is explicitly mentioned. For example,

For Example :

int add (int, int);

add (int, int);

In both prototypes, the return value is int, because by default the return value of a function in C++ is of type **int**. Look at the following examples:

Sl.No	Function Prototype	Return type
1	int sum(int, float)	int
2	float area(float, float)	float
3	char result()	char
4	double fact(int n)	double

Returning Non-integer values

A string can also be returned to a calling statement.

Program 11.24

```
#include<iostream>
#include<string.h>
using namespace std;
char *display()
{
return ("chennai");
}
int main()
{
 char s[50];
 strcpy(s,display());
  cout<<"\nExample:Function with Non Integer Return"<<s;
  return(0);
}
```

Output :

Example: Function with Non Integer Return Chennai

The Returning by reference

Program 11.25

```
#include<iostream>
using namespace std;
```


```
int main()
{
 int n1=150;
  int &n1ref=n1;
   cout<<"\nThe Value of N1 = "<<n1<<" and n1Reference = "<<n1ref;
   n1ref++;
 cout<<"\nAfter n1 increased the Value of N1 = "<<n1;
  cout<<" and n1Reference = "<<n1ref;
  return(0);
}
```

Output:

The Value of N1 = 150 and n1Reference = 150

After n1 increased the Value of N1 = 151 and n1Reference = 151

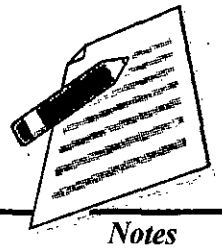
Recursive Function

A function that calls itself is known as recursive function. And, this technique is known as recursion.

Example 1: Factorial of a Number Using Recursion

Program 11.26

```
#include <iostream>
using namespace std;
int factorial(int); // Function prototype //
int main()
{
 int no;
 cout<<"\nEnter a number to find its factorial: ";
  cin >> no;
  cout << "\nFactorial of Number " << no <<" = " << factorial(no);
 return 0;
}
int factorial(int m)
{
 if (m > 1)
 {
  return m*factorial(m-1);
   }
  else
    {
  return 1;
}
}
```



Output :

Enter a number to find its factorial: 5

Factorial of Number 5 = 120

Note: Function prototype is mandatory since the function factorial() is given after the main() function.

Example 2: Finding HCF of any to number using Recursion

Program 11.27

```
#include <iostream>
using namespace std;
//Function to find HCF //
int hcf(int n1, int n2)
{
if (n2 != 0)
 return hcf(n2, n1 % n2);
 else
 return n1;
}
int main()
{
 int num1, num2;
 cout << "Enter two positive integers: ";
 cin >> num1 >> num2;
  cout << "Highest Common Factor (HCF) of " << num1;
  cout<< " & " << num2 << " is: " << hcf(num1, num2); return 0;
 }
```

Output:

Enter two positive integers: 350 100

Highest Common Factor (HCF) of 350 & 100 is: 50

Scope Rules of Variables

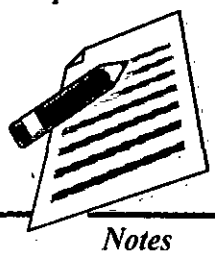
Scope refers to the accessibility of a variable. There are four types of scopes in C++.

They are: **Local scope, Function scope, File scope and Class scope.**

Introduction

A scope is a region or life of the variable and broadly speaking there are three places, where variables can be declared,

- Inside a block which is called local variables.
- Inside a function is called function variables.
- Outside of all functions which is called global variables.



- Inside a class is called class variable or data members.

Local Scope:

- A local variable is defined within a block. A block of code begins and ends with curly braces { }.
- The scope of a local variable is the block in which it is defined.
- A local variable cannot be accessed from outside the block of its declaration.
- A local variable is created upon entry into its block and destroyed upon exit.

Program 11.28 (a)

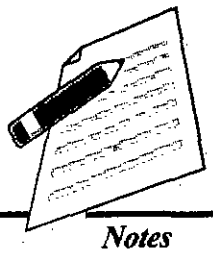
```
//Demo to test Local Scope//
#include<iostream>s
using namespace std;
int main ( )
{
int a, b ;
a = 10;
b = 20;
if (a > b)
{
int temp; //local to this if block//
temp = a;
a = b;
b = temp;
}
cout <<"\n Descending order .... \n";
cout <<a <<"\t"<<b;
return(0);
}
```

Output:

```
Descending order ....
10  20
```

Program 11.28 (b)

```
//Demo to test Local Scope//
#include<iostream>s
using namespace std;
int main ( )
{
int a, b ;
a = 10;
b = 20;
if (a > b)
```




```
{
int temp; //local to this if block//
temp = a;
a = b;
b = temp;
}
cout <<temp;
return(0);
}
```

In function 'int main()':

[Error] 'temp' was not declared in this scope

On compilation the Program 11.28(b), the compiler prompts an error message: The variable **temp** is not accessible. Because the life time of a local variable is the life time of a block in its state of execution. Local variables die when its block execution is completed.

Function Scope:

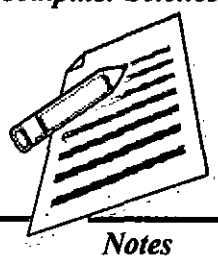
- The scope of variables declared within a function is extended to the function block, and all sub-blocks therein.
- The life time of a function scope variable, is the life time of the function block. The scope of formal parameters is function scope.

Program 11.29 (a)

```
//Demo to test Function Scope//
#include<iostream>
using namespace std;
void add(int x, int y)
{
 int m=x+y; //'m' declared within function add()//
 cout<<"\nThe Sum = "<<m;
}
int main ( )
{
int a, b ;
a = 10;
b = 20;
add(a,b);
return(0);
}
```

Program 11.29 (b)

```
//Demo to test Function Scope//
#include<iostream>
```




```
using namespace std;
void add(int x, int y)
{
 int m=x+y; //'m' declared within function add()//
 cout<<»\nThe Sum = «<<m;
}
int main ( )
{
int a, b ;
a = 10;
b = 20;
add(a,b);
cout<<m; //'m' declared within function add()//
return(0);
}
```

Output:

The Sum = 30

Note : In function 'int main()':

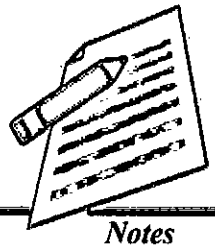
[Error] 'm' was not declared in this scope

File Scope:

- A variable declared above all blocks and functions (including main ( )) has the scope of a file. The life time of a file scope variable is the life time of a program.
- The file scope variable is also called as **global variable.**

Program 11.30

```
//Demo to test File or global Scope//
#include<iostream>
using namespace std;
int file_var=20; //Declared within File//
void add(int x, int y)
{
 int m=x+y+file_var;
 cout<<"\n The Sum = "<<m;
}
int main ( )
{
int a, b ;
a = 10;
b = 20;
add(a,b);
cout<<"\nThe File Variable = "<<file_var;
return(0);
}
```



Output:

The Sum = 50

The File Variable =20

Class Scope:

- A class is a new way of creating and implementing a user defined data type. Classes provide a method for packing together data of different types.
- Data members are the data variables that represent the features or properties of a class.

class student
{
private :
int mark1, mark2, total;
};
	The class student contains mark1, mark2 and total are data variables. Its scope is within the class student only.

Note: The class scope will be discussed later in chapter **"Classes and Object".**

Scope resolution operator

- The scope operator reveals the hidden scope of a variable. The scope resolution operator (::) is used for the following purposes.
- To access a Global variable when there is a Local variable with same name. An example using Scope Resolution Operator.

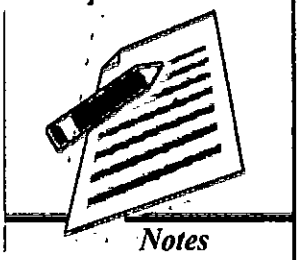
Program 11.31

```
//Program to show that we can access a global variable
//using scope resolution operator :: when there is a local
//variable with same name //
#include<iostream>
using namespace std;
int x=45; // Global Variable x
int main()
{
    int x = 10; // Local Variable x
    cout << «\nValue of global x is « << ::x;
    cout << «\nValue of local x is « << x;
    return 0;
}
```

Output:

Value of global x is 45

Value of local x is 10



SUMMARY

A large program can typically be split into smaller sized blocks called as functions. Functions can be classified into Pre-defined or Built-in or Library Functions and User-defined Functions. User-defined functions are created by the user. The void function tells the compiler that the function returns nothing. The return statement returns a value to the calling function and transfers the program control back to the calling function. Default the data type of a function in C++ is of type int. A function that calls itself is known as recursive function. Scope refers to the accessibility of a variable. There are four types of Scopes. They are: Local scope, Function scope, File scope and Class scope. The scope operator (::) reveals the hidden scope of a variable.

EXERGISE

MCQ

1. Which of the following header file defines the standard I/O predefined functions ?

   A) stdio.h B) math.h
   C) string.h D) ctype.h

2. Which function is used to check whether a character is alphanumeric or not.

   A) isalpha() B) isdigit()
   C) isalnum() D) islower()

3. Which function begins the program execution ?

   A) isalpha() B) isdigit()
   C) main() D) islower()

4. Which of the following function is with a return value and without any argument ?

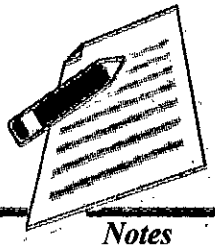
   A) x=display(int, int) **B) x=display()**
   C) y=display(float) D) display(int)

5. Which is return data type of the function prototype of add(int, int); ?

   A) int B) float
   C) char D) double

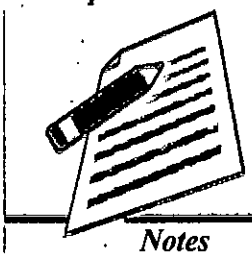
6. Which of the following is the scope operator ?

   A)> B)&
   C)% **D)::**



Review Questions

1. Write a program that will ask the user to enter a single character. Find out whether it is alphabetic, digit or special character.
2. Write a program that will ask the user to enter his/her name. Convert all uppercase letter to lowercase letter and vice-versa.
3. Write a program that will ask the user to enter a character. Check if it is alphabetic or not. If alphabetic, check whether it is in uppercase or lowercase.
4. What is the difference between the following ? (i) getchar ( ) & putchar ( ) (ii) gets ( ) & puts ( )
5. Write a program that will count the number of characters in a sentence entered by user.
6. Write a program that will count the number of words in a sentence entered by the user.
7. Write a program that will count the number of A's, E's, I's, O's and U's in a sentence.
8. Differentiate between the following : (i) Actual and Formal parameters (ii) Call by value and call by reference.
9. What do you mean by inline function ?
10. What are the advantages of function prototype in C++?



11 ARRAY

- Understand the concept of array.
- Discuss the features of array.
- Describe the types of array.
- Discuss the applications of array.

Objective of the chapter:

The basic objective of this chapter is to through some light on the initial concepts of array so that the types and applications of array can be learned.

Introduction

The variables are used to store data. These variables are the one of the basic building blocks in C++. A single variable is used to store a single value that can be used anywhere in the memory. In some situations, we need to store multiple values of the same type. In that case, it needs multiple variables of the same data type. All the values are stored randomly anywhere in the memory.

For example, to store the roll numbers of the 100 students, it needs 100 variables named as roll1, roll2, roll3,.....roll100 . It becomes very difficult to declare 100 variables and store all the roll numbers. In C++, the concept of Array helps to store multiple values in a single variable. Literally, the meaning of **Array is "More than one". In other words, array is an easy way of storing multiple values of the same type referenced by a common name". An array is also a derived data type in C++. "An array is a collection of variables of the same type that are referenced by a common name".** In an array, the values are stored in a fixed number of elements of the same type sequentially in memory. Therefore, an integer array holds a sequence of integers; a character array holds a sequence of characters, and so on. The size of the array is referred to as its dimension.

Types of Arrays:

There are different types of arrays used in C++. They are:

- One-dimensional arrays
- Two-dimensional arrays
- Multi-dimensional arrays

One-dimensional array

This is the simplest form of an array. A one dimensional array represents values that are stored in a single row or in a single column.

Declaration

Syntax:

<data type><array_name> [<array_size>];

data_type declares the basic type of the array, which is the type of each element in the array.

array_name specifies the name with which the array will be referenced.

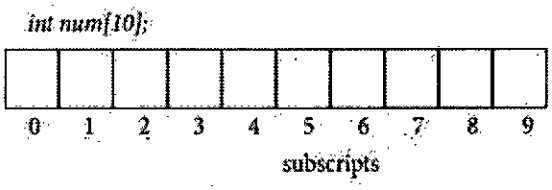
array_size defines how many elements the array will hold. Size should be specified with square brackets [ ].

Example:

int num[10];

In the above declaration, an array named "num" is declared with 10 elements (memory space to store 10 different values) as integer type.

To the above declaration, the compiler allocated 10 memory locations (boxes) in the common name "num" as given below



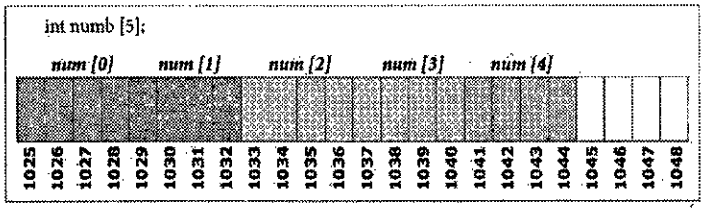
Each element (Memory box) has a unique index number starting from 0 which is known as "subscript". The subscript always starts with 0 and it should be an unsigned integer value. Each element of an array is referred by its name with subscript index within the square bracket. For example, num[3] refers to the 4th element in the array.

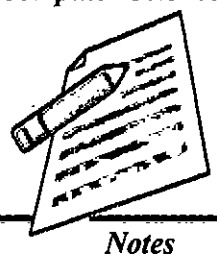
Some more array declarations with various data types:

```
char emp_name[25];    //    character array named emp_name with size 25
float salary[20];  // floating-point array named salary with size 20
int a[5], b[10], c[15]; // multiple arrays are declared of type int
```

Memory representation of an one dimensional array

The amount of storage required to hold an array is directly related with type and size. The following figure shows the memory allocation of an array with five elements.





The above figure clearly shows that, the array num is an integer array with 5 elements. As per the Dev-C++ compiler, 4 bytes are allocated for every int type variable. Here, there are totally 5 elements in the array, where for each element, 4 bytes will be allocated. Totally, 20 bytes will be allocated for this array.

Datatype	Turbo C++	Dev C++
char	1	1
int	2	4
float	4	4
long	4	4
double	8	8
long double	10	10

The memory space allocated for an array can be calculated using the following formula:

Number of bytes allocated for type of array × Number of elements

Initialization

An array can be initialized at the time of its declaration. Unless an array is initialized, all the array elements contain garbage values.

Syntax:

<datatype> <array_name> [size] = {value-1,value-2,..... ,value-n};

Example

int age[5]={19,21,16,1,50};

In the above example, the array name is 'age' whose size is 5. In this case, the first element 19 is stored in age[0], the second element 21 is stored in age[1] and so on as shown in figure 12.1

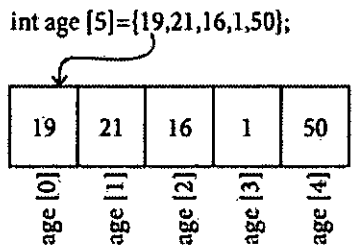


Figure 12.1

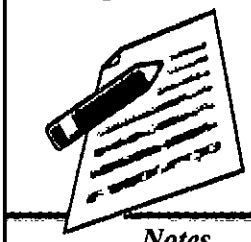
While declaring and initializing values in an array, the values should be given within the curly braces i.e. { ..... }

The size of an array may be optional when the array is initialized during declaration.

Example:

int age[]={ 19,21,16,1,50};

In the above initialization, the size of the array is not specified directly in the declaration with initialization. So, the size is determined by compiler which depends on the total number of values. In this case, the size of the array is five.



More examples of array initialization:

float x[5] = {5.6, 5.7, 5.8, 5.9, 6.1};

char vowel[6] = {'a', 'e', 'i', 'o', 'u', '\0'};

Accepting values to an array during run time :

Multiple assignment statements are required to insert values to the cells of the array during runtime. The for loop is ideally suited for iterating through the array elements.

// Input values while execution

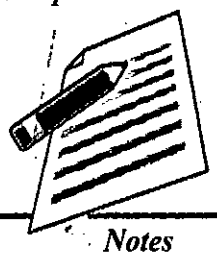
```
#include <iostream>
using namespace std;
int main()
{
int num[5];
for(int i=0; i<5; i++)
 {
  cout<< "\n Enter value " << i+1 << "= ";
 cin>>num[i];
    }
}
```

In the above program, a for loop has been constructed to execute the statements within the loop for 5 times. During each iteration of the loop, cout statement prompts you to "Enter value ....." and cin gets the value and stores it in num[i];

The following table shows the execution of the above code block.

Iteration	i <5	cout << "\n Enter value " << i+1 << "= ";	cin>>num [i];	Received value stored in memory		i++ (i=i+1)
1	5 > 0 (T)	Enter value 1 =	num[0] = 5	num[0]	5	1
2	5 > 1 (T)	Enter value 2 =	num[1] = 10	num[1]	10	2
3	5 > 2 (T)	Enter value 3 =	num[2] = 15	num[2]	15	3
4	5 > 3 (T)	Enter value 4 =	num[3] = 20	num[3]	20	4
5	5 > 4 (T)	Enter value 4 =	num[25 = [4	num[4]	25	5
6	5 > 5 (F)	Exit from Loop				

In for loop, the index *i* is declared with an initial value 0 (zero). Since in most of the cases, the initial value of the loop index will be used as the array subscript representation.



Accessing array elements

Array elements can be used anywhere in a program as we do in case of a normal variable. The elements of an array are accessed with the array name followed by the subscript.index within the square bracket.

Example:

```
cout<<num[3];
```

In the above statement, num[3] refers to the 4th element of the array and cout statement displays the value of num[3].

The subscript in bracket can be a variable, a constant or an expression that evaluates to an integer.

```
// Accessing array elements
#include <iostream>
using namespace std;
int main()
{
int num[5] = {10, 20, 30, 40, 50};
int t=2;
cout<<num[2] <<endl; // S1
 cout<<num[3+1] <<endl; // S2
  cout<<num[t=t+1]; // S3
}
```

output:

```
30
50
40
```

In the above program, statement **S1** displays the value of the 3rd element (subscript index 2). **S2** will display the value of the 5th element (i.e. Subscript value is 3+1 = 4). In the same way statement **S3** will display the value of the 4th element.

The following program illustrates the writing and reading of array elements.

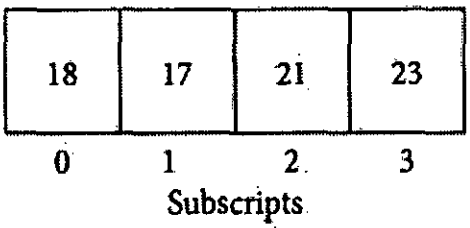
```
//Program to read and write the values from an array
#include <iostream>
using namespace std;
int main()
{
int age[4];//declaration of array
cout<< "Enter the age of four persons:" <<endl;
for(int i = 0; i < 4; i++)//loop to write array elements
cin>> age[i];
```

```
cout<<"The ages of four persons are:";
 for(int j = 0; j< 4; j++)
  cout<< age[j]<<endl;
}
```

The following table shows the execution of the above code lines

Iteration	i < 4	cin>> age [i];	Value received	age	i++ (i=i+1)
1	4 > 0 (T)	cin>> age[0];	18	age [0] = 18	1
2	4 > 1 (T)	cin>> age[1];	17	age [1] = 17	2
3	4 > 2 (T)	cin>> age[2];	21	age [2] = 21	3
4	4 > 3 (T)	cin>> age[3];	23	age [3] = 23	4
5	4 > 4 (F)	Exit from loop			

After the successful execution of the above statements, the given values will be stored in memory like,



Second for loop:

```
for (int j = 0; j < 4; j++)
cout<< age[j]<<endl;
```

The above statements are used to read the values from the memory and display the values.

The following table shows the execution of the above code.

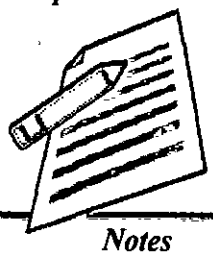
Iteration	j< 4	cout << age[j];	values read from memory	Output	j++
1	4 > 0 (T)	cout << age[0];	18	18	1
2	4 > 1 (T)	cout << age[1];	17	17	2
3	4 > 2 (T)	cout << age[2];	21	21	3
4	4 > 3 (T)	cout << age[3];	23	23	4
5	4 > 4 (F)	Exit from loop			

So, the final output will be:

Enter the age of four persons:

18

17



21

23

The ages of four persons are:

18

17

21

23

Traversal:

Accessing each element of an array at least once to perform any operation is known as "Traversal". Displaying all the elements in an array is an example of "traversal".

Traversal of an array

```
#include <iostream>
using namespace std;
int main()
{
 int num[5];
 for (int i=0; i<5; i++)
  {
   cout<< "\n Enter value " << i+1 <<"= ";
  cin>>num[i]; // Reading from keyboard
  //Traversing the array elements sequentially and adding 1 to each element num[i]
= num[i] + 1;
    }
 cout<< "\n After incrementing, the values in array num..." <<endl;
  for (int j=0; j<5; j++)
   {
   //Traversing the array elements sequentially and printing each one of them
cout<<num[j] <<endl;
    }
}
```

Output:

Enter value 1= 10

Enter value 2= 20

Enter value 3= 30

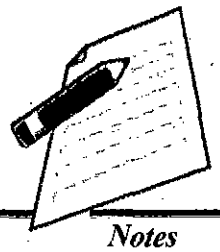
Enter value 4= 40

Enter value 5= 50

After incrementing, the values in array num...

11

21



Notes

31

41

51

Program to read the marks of 10 students and to find the average of all those marks.

```
#include <iostream>
using namespace std;
int main()
{
int marks[10], sum=0;
float avg;
for(int i=0; i<10; i++)
    {
 cout<< "\n Enter Mark " << i+1 << "= ";
  cin>> marks[i];
   sum=sum+marks[i];
    }
 avg=sum/10.0;
 cout<< "\n The Total Marks: " << sum;
cout<< "\n The Average Mark: " <<avg;
}
```

Output:

Enter Mark 1= 41

Enter Mark 2= 98

Enter Mark 3= 65

Enter Mark 4= 75

Enter Mark 5= 35

Enter Mark 6= 82

Enter Mark 7= 64

Enter Mark 8= 5

Enter Mark 9= 58

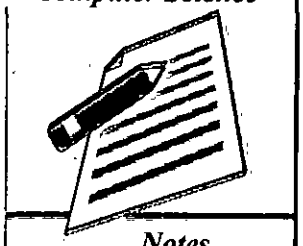
Enter Mark 10= 68

The Total Marks: 591

The Average Mark: 59

C++ program to inputs 10 values and count the number of odd and even numbers

```
#include <iostream>
using namespace std;
int main()
```



Notes

```
{
int num[10], even=0, odd=0;
 for (int i=0; i<10; i++)
  {
  cout<< "\n Enter Number " << i+1 <<"= ";
   cin>>num[i];
    if (num[i] % 2 == 0)
     ++even;
      else
     ++odd;
     }
     cout << «\n There are «<< even <<» Even Numbers»;
     cout << «\n There are «<< odd <<» Odd Numbers»;
}
```

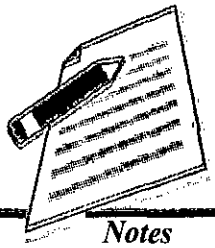
Output:

```
Enter Number 1= 78
Enter Number 2= 51
Enter Number 3= 32
Enter Number 4= 66
Enter Number 5= 41
Enter Number 6= 68
Enter Number 7= 27
Enter Number 8= 65
Enter Number 9= 28
Enter Number 10= 94
There are 6 Even Numbers
There are 4 Odd Numbers
```

(HOTS : Rewrite the above program using the conditional operator instead of if)

Program to read the prices of 10 products in an array and then print the sum and average of all the prices

```
#include <iostream>
using namespace std;
int main()
{
    float price[10], sum=0, avg=0, prod=1;
    for(int i=0; i<10; i++)
```




```
    {
        cout<< «\n Enter the price of item « << i+1 <<»= «;
        cin>> price[i];
        sum+=price[i];
    }
avg=sum/10.0;
cout<< "\n Sum of all prices: " << sum;
cout<< "\n Average of all prices: " <<avg;
}
```

Program to accept the sales of each day of the month and print the average sales for each month

```
#include <iostream>
using namespace std;
int main()
{
 int days;
 float sales[5], avgSales=0, totalSales=0;
  cout<< "\n Enter No. of days: ";
  cin>> days;
  for (int i=0; i<days; i++)
     {
   cout<< "\n Enter sales on day - " << i+1 <<": ";
   cin>> sales[i];
   totalSales+=sales[i];
}
avg=total sales/days;
cout<< "\n Average Sales = " <<avgSales;
return 0;
}
```

Searching in a one dimensional array:

Searching is a process of finding a particular value present in a given set of numbers. The linear search or sequential search compares each element of the list with the value that has to be searched until all the elements in the array have been traversed and com

```
{
    for (int i=0; i<size; i++)
    {
        if (arr[i] == value)
        return i; // return index value
    }
```

```
return -1;
}
int main()
{
    int num[10], val, id;
    for (int i=0; i<10; i++)
    {
    cout<< «\n Enter value « << i+1 <<»= «;
    cin>>num[i];
    }
    cout<< «\n Enter a value to be searched: «;
    cin>>val;
    id=Search(num,10,val);
    if(id==-1)
    cout<< «\n Given value is not found in the array..»; else
    cout<< «\n The value is found at the position» << id+1; return 0;
}
```

The above program reads an array and prompts for the values to be searched. It calls Search( ) function which receives array, size and value to be searched as parameters. If the value is found, then it returns the array index to the called statement; otherwise, it returns -1.

Strings

A string is defined as a sequence of characters where each character may be a letter, number or a symbol. Each element occupies one byte of memory. Every string is terminated by a null ('\0', ASCII code 0) character which must be appended at the end of the string. In C++, there is no basic data type to represent a string. Instead, it implements a string as an one-dimensional character array. When declaring a character array, it also has to hold a null character at the end, and so, the size of the character array should be one character longer than the length of the string.

Character Array (String) creation

To create any kind of array, the size (length) of the array must be known in advance, so that the memory locations can be allocated according to the size of the array. Once an array is created, its length is fixed and cannot be changed during run time. This is shown in figure12.2

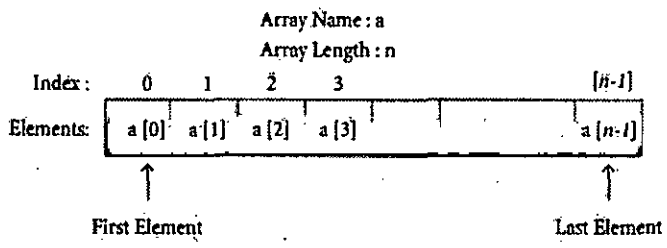
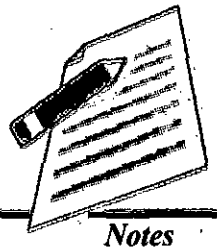


Figure 12.2



Syntax

Array declaration is:

char array_name[size];

In the above declaration, the size of the array must be an unsigned integer value.

For example,

char country[6];

Here, the array reserves 6 bytes of memory for storing a sequence of characters. The length of the string cannot be more than 5 characters and one location is reserved for the null character at the end.

Program to demonstrate a character array.

```
#include <iostream>
using namespace std;
int main()
{
    char country[6];
    cout<< «Enter the name of the country: «;
    cin>>country;
    cout<<» The name of the country is «<<country;
}
```

OUTPUT

Enter country the name: INDIA

The country name is INDIA

Initialization

The character array can be initialized at the time of its declaration. The syntax is shown below:

char array_name[size]={ list of characters separated by comma or a string } ;

For example,

char country[6]="INDIA";

In the above example, the text "INDIA" has 5 letters which is assigned as initial value to array country. The text is enclosed within double quotes. The memory representation is shown in Figure 13.3

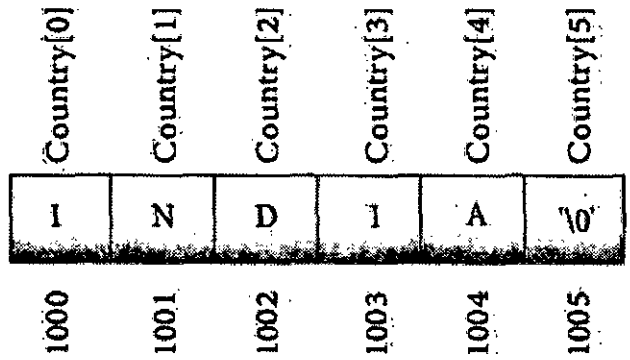


Figure 12.3

In the above memory representation, each character occupies one byte in memory. At the end of the string, a null character is automatically added by the compiler.

C++ also provides other ways of initializing the character array:

char country[6]={'I', 'N', 'D', 'I', 'A', '\0'};

char country[]="INDIA";

char country[]={'I', 'N', 'D', 'I', 'A', '\0'};

If the size of the array is not explicitly mentioned, the compiler automatically calculate the size of the array based on the number of elements in the list and allocates space accordingly.

In the initialization of the string, if all the characters are not initialized, then the rest of the characters will be filled with NULL.

Example:

char str[5]={'5','+','A'};

str[0]; ---> 5

str[1]; ---> +

str[2]; ---> A

str[3]; ---> NULL

str[4]; ---> NULL

During initialization, the array of elements cannot be initialized more than its size.

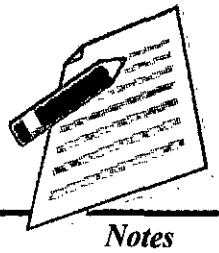
For example

char str[2]={'5','+','A','B'}; // Invalid

In the above example, the compiler displays "initialize-string for array of chars is too long" error message.

Write a program to demonstrate various methods of initializing the character arrays

```
#include <iostream>
using namespace std;
```




```
int main()
{
    char arr1[6]=»INDIA»;
    char arr2[6]={‹I›,›N›,›D›,›I›,›A›,›\0›};
    char arr3[]=»TRICHY»;
    char arr4[]={‹T›,›R›,›I›,›C›,›H›,›Y›,›\0›};
    char arr5[8]=»TRICHY»;
    cout<<»arr1 is :» <<arr1<< « and its size is «<<sizeof(arr1)<<endl;
    cout<<»arr2 is :» <<arr2<< « and its size is «<<sizeof(arr2)<<endl;
    cout<<»arr3 is :» <<arr3<< « and its size is «<<sizeof(arr3)<<endl;
    cout<<»arr4 is :» <<arr4<< « and its size is «<<sizeof(arr4)<<endl;
    cout<<»The elements of arr5»<<endl;
    for(int i=0;i<8;i++)
    cout<<arr5[i]<<» «;
    return 0;
}
```

Output

arr1 is :INDIA and its size is 6

arr2 is :INDIA and its size is 6

arr3 is :TRICHY and its size is 7

arr4 is :TRICHY and its size is 7

The elements of arr5

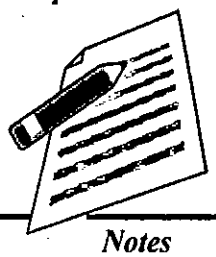
T R I C H Y

Read a line of Text

In C++, cin.get() is used to read a line of text including blank spaces. This function takes two arguments. The first argument is the name of the string and second argument is the maximum size of the array.

Write a program to display a line of text using get() function.

```
//str10.cpp
//To read a line of text
#include <iostream>
using namespace std; int main()
{
char str[100];
cout<< "Enter a string: ";
cin.get(str, 100);
cout<< "You entered: " <<str<<endl;
return 0;
}
```



Output

Enter a string: I am a student

You entered: I am a student

In the above program, str is the name of the string and 100 is the maximum size of the character array that represents the string str.

In C++, getline() is also used to read a line of text from the input stream. It can read the characters till it encounters a newline character or a delimiter specified by the user. This function is available in the <string> header.

Write a Program to check palindrome or not

```
#include<iostream>
using namespace std;
int main( )
{
    int i, j, len, flag =1;
    char a [20];
    cout<<»Enter a string:»;
    cin>>a;
    for(len=0;a[len]!=›\0›;++len)
for(!=0,j=len-1;i<len/2;++i,--j)
    {
    if(a[j]!=a[i])
       flag=0;
    }
    if(flag==1)
       cout<<»\n The String is palindrome»;
    else
       cout<<»\n The String is not palindrome»;
    return 0;
}
```

Output:

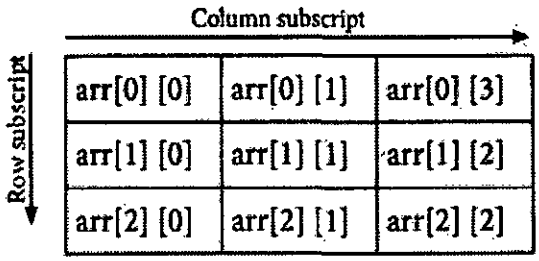
Enter a string : madam

The String is palindrome

Two-dimensional array

Two-dimensional (2D) arrays are collection of similar elements where the elements are stored in certain number of rows and columns. An example m × n matrix where m denotes the number of rows and n denotes the number of columns is shown in Figure12.4

```
int arr[3][3];
```



The array arr can be coneptually viewed in matrix form with 3 rows and coloumns. point to be noted here is since the subscript starts with 0 arr [0][0] represents the first element.

Figure 12.4

Declaration of 2-D array

The declaration of a 2-D array is

data-type array_name[row-size][col-size];

In the above declaration, data-type refers to any valid C++ data-type, array_name refers to the name of the 2-D array, row-size refers to the number of rows and col-size refers to the number of columns in the 2-D array.

For example

int A[3][4];

In the above example, A is a 2-D array, 3 denotes the number of rows and 4 denotes the number of columns. This array can hold a maximum of 12 elements.

Array size must be an unsigned integer value which is greater than 0. In arrays, column size is compulsory but row size is optional.

Other examples of 2-D array are:

int A[3][3];

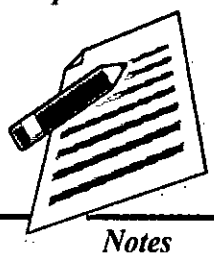
float x[2][3];

char name[5][20];

Initialization of Two-Dimensional array

The array can be initialized in more than one way at the time of 2-D array declaration. For example

```
int matrix[4][3]={
{10,20,30},// Initializes row 0
{40,50,60},// Initializes row 1
{70,80,90},// Initializes row 2
{100,110,120}// Initializes row 3
};
```



int matrix[4][3]={10,20,30,40,50,60,70,80,90,100,110,120};

Array's row size is optional but column size is compulsory.

For example

```
int matrix[][3]={
{10,20,30},// row 0
{40,50,60},// row 1
{70,80,90},// row 2
{100,110,120}// row 3
};
```

Accessing the two-dimensional array

Two-dimensional array uses two index values to access a particular element in it, where the first index specifies the row value and second index specifies the column value.

```
matrix[0][0]=10;// Assign 10 to the first element of the first row
matrix[0][1]=20;// Assign 20 to the second element of the first row
matrix[1][2]=60;// Assign 60 to the third element of the second row
matrix[3][0]=100;// Assign 100 to the first element of the fourth row
```

Write a program to perform addition of two matrices

```
#include<iostream>
#include<conio>
using namespace std;
int main()
{
    int row, col, m1[10][10], m2[10][10], sum[10][10];
    cout<<»Enter the number of rows : «;
    cin>>row;
    cout<<»Enter the number of columns : «;
    cin>>col;
    cout<< «Enter the elements of first matrix: «<<endl;
    for (int i = 0;i<row;i++ )
    for (int j = 0;j <col;j++ )
    cin>>m1[i][j];
    cout<< «Enter the elements of second matrix: «<<endl;
    for (int i = 0;i<row;i++ )
    for (int j = 0;j<col;j++ )
    cin>>m2[i][j];
    cout<<»Output: «<<endl;
    for (int i = 0;i<row;i++ )
    for (int j = 0;j<col;j++ )
```

```
    {
    sum[i][j]=m1[i][j]+m2[i][j];
    cout<<sum[i][j]<<» «;
    }
    cout<<endl<<endl;
    }
getch();
return 0;
}
```

```
Enter the number of rows :  2
Enter the number of column : 2
Enter the elements of first matrix:
1
1
1
1
Enter the elements of second matrix:
1
1
1
1
Output:
2  2
2  2
```

Memory representation of 2-D array

Normally, the two-dimensional array can be viewed as a matrix. The conceptual view of a 2-D array is shown below:

int A[4][3];

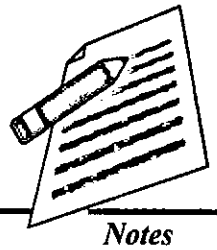
A[0][0]	A[0][1]	A[0][2]
A[1][0]	A[1][1]	A[1][2]
A[2][0]	A[2][1]	A[2][2]
A[3][0]	A[3][1]	A[3][2]

In the above example, the 2-D array name A has 4 rows and 3 columns.

Like one-dimensional, the 2-D array elements are stored in continuous memory.

There are two types of 2-D array memory representations. They are:

- Row-Major order
- Column-Major order



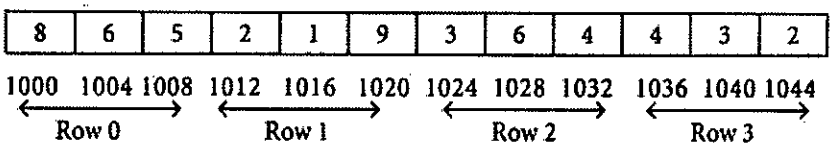
For example

int A[4][3]={

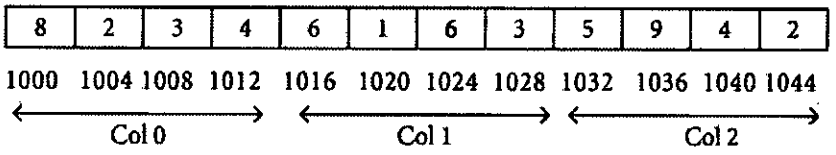
{ 8,6,5}, { 2,1,9}, {3,6,4}, {4,3,2},

Row Major order

In row-major order, all the elements are stored row by row in continuous memory locations, that is, all the elements in first row, then in the second row and so on. The memory representation of row major order is as shown below;



Column Major order



Array of strings

An array of strings is a two-dimensional character array. The size of the first index (rows) denotes the number of strings and the size of the second index (columns) denotes the maximum length of each string. Usually, array of strings are declared in such a way to accommodate the null character at the end of each string. For example, the 2-D array has the declaration:

char Name[6][10];

In the above declaration, the 2-D array has two indices which refer to the row size and column size that is 6 refers to the number of rows and 10 refers to the number of columns.

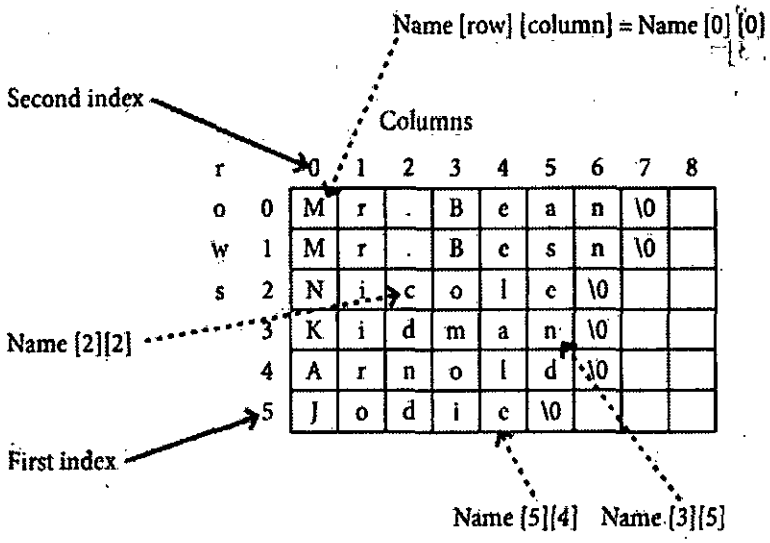
Initialization

For example

char Name[6][10] = {"Mr. Bean", "Mr. Bush", "Nicole", "Kidman", "Arnold", "Jodie"};

In the above example, the 2-D array is initialized with 6 strings, where each string is a maximum of 9 characters long, since the last character is null.

The memory arrangement of a 2-D array is shown below and all the strings are stored in continuous locations.



r o w s	0	1	2	3	4	5	6	7	8
0	M	r	.	B	e	a	n	\0	
1	M	r	.	B	e	s	n	\0	
2	N	i	c	o	l	e	\0		
3	K	i	d	m	a	n	\0		
4	A	r	n	o	l	d	\0		
5	J	o	d	i	e	\0			

C++ program to demonstrate array of strings using 2d character array

```
#include<iostream>
using namespace std;
int main( )
{
    // initialize 2d array
    char colour [4][10]={«Blue»,»Red»,»Orange», «yellow»};
    // printing strings stored in 2d array
    for (int i=0; i <4; i++)
    cout << colour [i] << «\n»;
}
```

Output:

Blue
Red
Orange
Yellow

Passing Arrays to functions

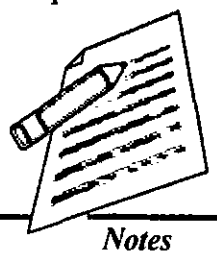
In C++, arrays can be passed to a function as an argument. To pass an array to a function in C++, the function needs the array name as an argument.

Passing a two-dimensional array to a function

Write a program to display marks of 5 students by passing one-dimensional array to a function.

C++ program to display marks of 5 students (one dimensional array)

```
#include <iostream>
using namespace std;
```



Notes

```
void display (int m[5]);
int main()
{
    int marks[5]={88, 76, 90, 61, 69};
    display(marks);
    return 0;
}
void display (int m[5])
{
    cout << «\n Display Marks: « << endl;
    for (int i=0; i<5; i++)
    {
        cout << «Student « << i+1 << «: « << m[i]<<endl;
    }
}
```

Output:

Display Marks:
Student 1: 88
Student 2: 76
Student 3: 90
Student 4: 61
Student 5: 69

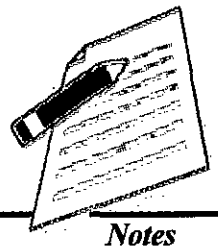
SUMMARY

Array is a collection of values of same type. On one dimensional array you can perform the operations like traversing, searching, sorting. Traversing of an array means accessing all the elements of the array one by one. Searching is the process of finding an element within the array list. Sorting is a method to arrange the list either in ascending or descending order. Two dimensional array has two subscripts or index. You can initialize array elements at run time.

EXERCISE

MCQ

1. Which of the following is the collection of variables of the same type that an referenced by a common name ?
   a) int
   b) float
   c) **Array**
   d) class



2. Array subscripts is always starts with which number ?
   a) -1 **b) 0**
   c) 2 d) 3

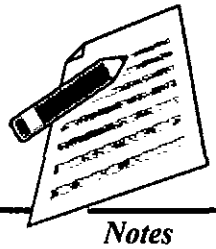
3. int age[]={6,90,20,18,2}; How many elements are there in this array?
   a) 2 **b) 5**
   c) 6 d) 4

4. cin>>n[3]; To which element does this statement accepts the value?
   a) 2 **b) 3**
   c) 4 d) 5

5. By default, the string and with which character?
   a) \o b) \t
   c) \n d) \b

Review Questions

1. What is the need of an array?
2. What do you mean by searching?
3. What do you mean by sorting?
4. Write a program that will read 20 float values in a one dimensional array and find out the following: (i) Number of values greater than zero. (ii) Number of values equal to zero. (iii) Number of values less than zero.
5. Write a program that will find out the sum of two diagonals of two dimensional array of A [N] [N].
6. Write a program to search whether the element entered by the user is present in a one dimensional array of 10 elements or not.



Notes

12 STRUCTURE, TYPEDEF & ENUMERATED DATA TYPE

- Understand the concept of structure.
- Discuss the features of structure.
- Describe the types of structure.
- Discuss the applications of structure.
- Discuss the concept of enumerated data type.

Objective of the chapter:

The basic objective of this chapter is to through some light on the initial concepts of structure so that the types and applications of structure can be learned.

Introduction

Structure is a user-defined which has the combination of data items with different data types. This allows to group of variables of mixed data types together into a single unit.

Purpose of Structures

In any situation when more than one variable is required to represent objects of uniform data-types, array can be used. If the elements are of different data types, then array cannot support. If more than one variable is used, they can be stored in memory but not in adjacent locations. It increases the time consumption while searching. The structure provides a facility to store different data types as a part of the same logical element in one memory chunk adjacent to each other.

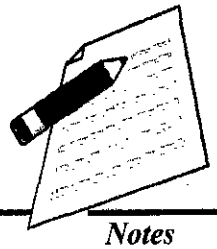
Declaring and defining structures

Structure is declared using the keyword 'struct'. The syntax of creating a structure is given below.

```
struct structure_name {
    type member_name1;
    type member_name2;
} reference_name;
```

Objects declared along with structure definition are called global objects

An optional field reference_name can be used to declare objects of the structure type directly.



Example:

```
struct Student
{
    long rollno;
    int age;
    float weight;
};
```

In the above declaration of the struct, three variables rollno, age and weight are used. These variables(data element)within the structure are called members (or fields). In order to use the Student structure, a variable of type Student is declared and the memory allocation is shown in figure 12.5

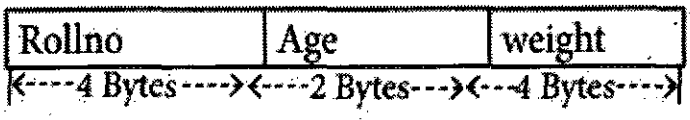


Fig 12.5 Memory Allocation

```
struct Student balu; // create a Student structure for Balu
```

This defines a variable of type Student named as Balu. Similar to normal variables, struct variable allocates memory for that variable itself. It is possible to define multiple variables of the same struct type:

```
struct Student frank; // create a structure for Student Frank.
```

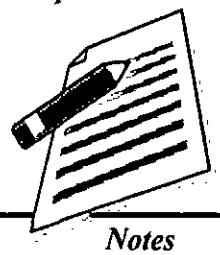
For example, the structure objects balu and frank can also be declared as the structure data type as:

```
struct Student
{
longrollno;
int age;
float weight;
}balu, frank;
```

Referencing Structure Elements

Once the two objects of student structure type are declared (balu and frank),their members can be accessed directly. The syntax for that is using a dot (.) between the object name and the member name. For example, the elements of the structure Student can be accessed as follows:

```
balu.rollno
balu.age
balu.weight
frank.rollno
```



frank.age

frank.weight

If the members are a pointer types then '→' is used to access the members. Let name is a character pointer ins student like char * name It can be accessed student → name

(Anonymous Structure Vs Named Structure)

A structure without a name/tag is called anonymous structure.

```
struct
{
long rollno;
    int age;
    float weight;
} student;
```

The student can be referred as reference name to the above structure and the elements can be accessed like student.rollno, student.age and student.weight .

Initializing structure elements

Values can be assigned to structure elements similar to assigning values to variables.

Example

```
balu.rollno= "702016";
balu.age= 18;
balu.weight= 48.5;
```

Also, values can be assigned directly as similar to assigning values to Arrays.

```
balu={702016, 18, 48.5};
```

Structure Assignments

Structures can be assigned directly instead of assigning the values of elements individually.

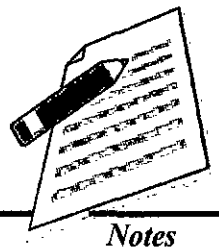
Example

If Mahesh and Praveen are same age and same height and weight then the values of Mahesh can be copied to Praveen

Structure assignment is possible only if both structure variables/ objects are same type.

```
struct Student
{
    int age;
    float height, weight;
}mahesh;
```

The age of Mahesh is 17 and the height and weights are 164.5 and 52.5 respectively. The following statement will perform the assignment.



mahesh = {17, 164.5, 52.5};
praveen =mahesh;
will assign the same age, height and weight to Praveen.

Examples:

The following C++ program reads student information through keyboard and displays the same

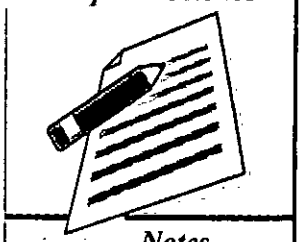
```
#include <iostream>
using namespace std;
struct Student
{
    int age;
    float height, weight;
} mahesh; void main( )
{
    cout<< " Enter the age:"<<endl;
    cin>>mahesh.age;
    cout<< "Enter the height:"<<endl;
    cin>>mahesh.height;
    cout<< "Enter the weight:"<<endl;
    cin>>mahesh.weight;
    cout<< "The values entered for Age, height and weight are"<<endl;
    cout<<mahesh.age<< "\t"<<mahesh.height<< "\t"<<Mahesh.. weight;
}
```

Output:

```
Enter the age:
18
Enter the height:
160.5
Enter the weight:
46.5
The values entered for Age, height and weight are
18  160.5   46.5
```

The following C++ Program assigns data to members of a structure variable and displays the contents

```
#include<iostream>
using namespace std;
struct Employee
```



Notes

```
{
char name[50];
int age;
    float salary;
};
int main()
{
    Employee e1;
    cout<< «Enter Full name: «;
    cin>>e1.name;
    cout<<endl<<»Enter age: «;
    cin>>e1.age;
    cout<<endl<< «Enter salary: «;
    cin>>e1.salary;
    cout<< «\nDisplaying Information.» <<endl;
    cout<< «Name: « <<e1.name <<endl;
    cout<<»Age: « <<e1.age <<endl;
    cout<< «Salary: « <<e1.salary;
return 0;
}
```

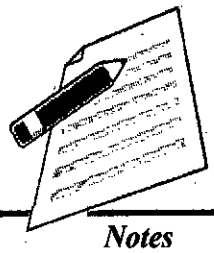
Output:

Enter Full name:
Ezhil
Enter age:
27
Enter salary:
40000.00
Displaying Information.
Name: Ezhil
Age: 27
Salary: 40000.00

Nested Structures

The structure declared within another structure is called a nested structure. A structure 'Student'was used to hold the student's information in the earlier examples. Date of birth can be included in the student's information. There are three components in the date of birth namely, date, month and year like 25-NOV-2017. Hence, another structure is used to keep the date of birth of a student. The following code creates a structure for the date of birth.

```
struct dob
```



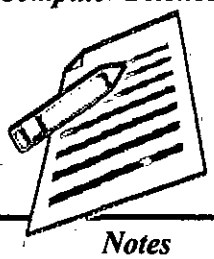

```
{
int date;
char month[3];
int year;
};
```

Values can be assigned to this structure as follows.

dob= {25,"NOV",2017}

The date of birth can be assigned as one of the elements in the student structure **nested structures act as members of another structure and the members of the child structure can be accessed as parent structure name. Child structure name. Member name.**

```
struct Student
{
    int age;
    float height, weight;
    struct dob
    {
    int date;
    char month[4];
    int year;
    };
}mahesh;
void main( )
{
    cout<< " Enter the age:"<<endl;
    cin>>mahesh.age;
    cout<< "Enter the height:"<<endl;
    cin>>mahesh.height;
    cout<< "Enter the weight:"<<endl;
    cin>>mahesh.weight;
    cout<< "The Date of birth:"<<endl;
    cout<< " Enter the day:"<<endl; cin>>mahesh.dob.date;
    cout<< "Enter the month:"<<endl;
    cin>>mahesh.dob.month;
    cout<< "Enter the year:"<<endl;
    cin>>mahesh.dob.year;
    cout<< "The values entered for Age, height and weightare"<<endl;
    cout<<mahesh.age<< "\t"<<mahesh.height<< "\t"<<mahesh.weight<<endl;
    cout<< "His date of Birth is:"<<endl<<mahesh.dob.date<< "-"<<mahesh.
    dob.month<< "-" <<mahesh.dob.year;
}
```



Output:

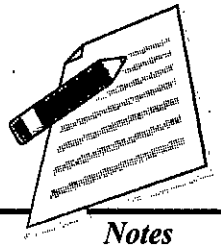
Enter the age:
18
Enter the height:
160.5
Enter the weight:
46.5
The Date of birth
Enter the day:
25
Enter the month:
NOV
Enter the year:
2017
The values entered for Age, height and weight are
18 160.5 46.5
His date of Birth is:
25-NOV-2017

Array of Structures

A class may contain many students. So, the definition of structure for one student can also be extended to all the students. If the class has 20 students, then 20 individual structures are required. For this purpose, an array of structures can be used. An array of structures is declared in the same way as declaring an array with built-in data types like int or char.

The following program reads the details of 20 students and prints the same.

```
#include <iostream>
using namespace std;
struct Student
{
    int age;
    float height, weight;
    char name[30];
};
void main( )
{
Student std[20];
int i;
```



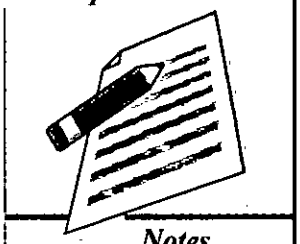

```
    cout<< " Enter the details for 20 students"<<endl;
    for(i=0;i<20;i++)
{
    cout<< " Enter the details of student"<<i+1<<endl;
    cout<< " Enter the age:"<<endl; cin>>std[i].age;
    cout<< "Enter the height:"<<endl;
    cin>>std[i].height;
    cout<< "Enter the weight:"<<endl;
    cin>>std[i].weight;
}
    cout<< "The values entered for Age, height and weight are"<<endl;
    for(i=0;i<20;i++)
   cout<<"Student "<<i+1<< "\t"<<std[i].age<< "\t"<<std[i].height<< "\t"<<std[i].
weight;
}
```

Output:

```
Enter the details of 20 students
Enter the details for student1
Enter the age:
18
Enter the height:
160.5
Enter the weight:
46.5
Enter the details for student2
Enter the age:
18
Enter the height:
164.5
Enter the weight:
61.5
The values entered for Age, height and weight are
Student 1   18    160.5 46.5
Student 2   18    164.5 61.5
```

The above program reads age , height and weight of 20 students and prints the same details. The output is shown for only two students due to space constraints.

Let an organization has three employees. If we want to read and print all their details then an array of structures is desirable for employees of this organization. This can be done through declaring an array of employee structures.

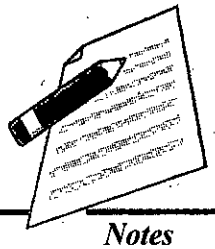


Notes

```
include<iostream>
using namespace std;
struct Employee
{
char name[50];
int age;
float salary;
};
int main()
{
Employee e1[3];
int i;
cout<< "Enter the details of 3 employees"<<endl;
for(i=0;i<3;i++)
{
    cout<< "Enter the details of Employee"<< i+1<<endl;
    cout<< «Enter name: «;
    cin>>e1[i].name;
    cout<<endl<<»Enter age: «;
    cin>>e1[i].age;
    cout<<endl<< «Enter salary: «;
    cin>>e[i]1.salary;
}
    cout<< «Displaying Information» <<endl;
    for(i=0;i<3;i++)
{
    cout<< " The details of Employee" <<i+1<<endl;
    cout<< «Name: « <<e1[i].name <<endl;
    cout<<»Age: « <<e1[i].age <<endl;
    cout<< «Salary: « <<e1[i].salary;
return 0;
}
```

Output:

```
Enter the details of 3 employees
Enter the details of Employee 1
Enter name:
Lilly
Enter age:
42
Enter salary:
40000.00
```




```
Enter the details of Employee 2
Enter name:
Aster
Enter age:
38
Enter salary:
60000.00
Enter the details of Employee 3
Enter name:
Jasmine
Enter age:
45
Enter salary:
80000.00
Displaying Information.
The details of Employee 1
Name:Lilly
Age: 42
Salary: 40000.00
The details of Employee 2
Name:Aster
Age:38
Salary:60000.00
The details of Employee 3
Name:Jasmine
Age:45
Salary:80000.00
```

ENUMERATED DATA TYPE

Enumerated data type works if you know in advance a finite list of values that a data type can take. It has the following features:

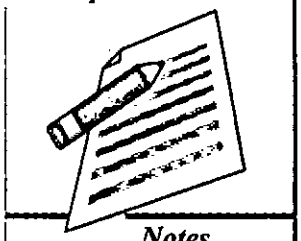
It is a user defined datatype.

It works if you know in advance a finite list of values that a data type can take.

The list cannot be input by the user or output on the screen. For example:

```
enum months { jan, feb, mar, apr, may};
enum days { sun, mon, tue, wed, thu };
enum toys { cycle, bicycle, scooter };
```

The enum specifier defines the set of all names which are permissible values of the type called members which are stored internally as integer constant. The first name



was given the integer value 0, the second value 1 and so on.

Example 3

jan = 0, feb = 1, mar = 2, apr = 3, may = 4

The ordering can be altered by using an equal sign and value.

enum months { jan = 1, feb, mar, apr, may };

Here jan = 1, feb = 2, mar = 3, apr = 4, may = 5

The value of the next element in the list is previous value plus one.

Enum color { red, green = 5, blue };

The value of blue will be 6

For example:

```
# include < iostream.h >
enum months { jan, feb, mar, apr, may };
void main ( )
{
months m1, m2;
m1 = jan;
m2 = apr;
int diff = m2 - m1;
cout << "Months between" << diff << "\n";
if (m1 > m2) cout << "m2 comes before m1"; }
```

The members of the enumerated series is stored as integer constant in the memory. This is the reason our program displays value 3 (m2 = apr i.e., m2 = 4, m1 = jan i.e., m1 = 1. so diff = m2 – m1 = 4 – 1 = 3).

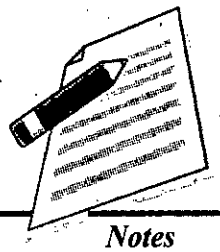
SUMMARY

Structure is a collection of simple variables which can be of same or different types. Typedef is used to define new data type for an existing data type. Typedef provides alternative name for standard data type. Enumerated data type is a user defined data type, it works if you know in advance a finite set of values that a data type can take. Enum specifier defines the set of names which are stored internally as integer constant.

EXERCISE

MCQ

1. The data elements in the structure are also known as
   (a) objects **(b) members**
   (c) data (d) records



2. Structure definition is terminated by

   (a) : (b) }

   (c) ; (d) ::

3. What will happen when the structure is declared?

   (a) it will not allocate any memory

   (b) it will allocate the memory

   (c) it will be declared and initialized

   (d) it will be only declared

4. Which of the following is a properly defined structure?

   (a) struct {int num;}

   (b) struct sum {int num;}

   (c) struct sum int sum;

   (d) struct sum {int num;};

5. Which of the following cannot be a structure member?

   (a) Another structure **(b) Function**

   (c) Array (d) variable of double datatype

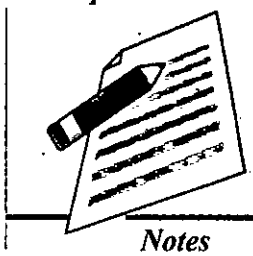
6. When accessing a structure member ,the identifier to the left of the dot operator is the name of

   (a) structure variable (b) structure tag

   (c) structure member (d) structure function

Review Questions

1. What is a structure? Write a structure specification in C++ for the record given below: Code: A string of 5 characters (including null) Cost: Integer Margin: Integer Call this structure item.
2. What will be the output of the following program? # include < iostream.h > enum days { sun, mon, tue, wed, thu, fri, sat }; void main ( ) { days today; for (today = sun; today < = sat; today ++) cout << "\n today is " << today; }.
3. State the reason why enum boolean {false, true}; is better than enum boolean {true, false}; 4. What is the advantage of typedef statement?
5. A phone number, such as 786-6370, can be thought of having two parts; the exchange code and the number. Write a program in C++ that uses a structure phone to store these two parts. Call the structure phone. Create two structure variable of type phone. Initialize one and have the user input a number for the other one. Then display both the numbers.



13 CLASSES & OBJECTS WITH CONSTRUCTORS / DESTRUCTORS

- Understand the concept of classes.
- Discuss the concept of objects.
- Describe the types of classes.
- Discuss the concept of constructors.
- Discuss the concept of destructors.

Objective of the chapter:

The basic objective of this chapter is to through some light on the initial concepts of classes and objects so that the types and applications of these can be learned.

Introduction

The most important feature of C++ is the "Class". It is significance is highlighted by the fact that Bjarne Stroustrup initially gave the name 'C with classes'. C++ offers classes, which provide the four features commonly present in OOP languages: **Abstraction, Encapsulation, Inheritance, and Polymorphism.**

Need for Class

Class is a way to bind the data and its associated functions together. Classes are needed to represent real world entities that not only have data type properties but also have associated operations. It is used to create user defined data type

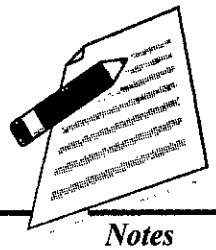
Declaration of a class

A class is defined in C++ using the keyword **class** followed by the name of the class. The body of the class is defined inside the curly brackets and terminated either by a semicolon or a list of declarations at the end.

The only difference between structure and class is the members of structure are by default **public** where as it is **private in class.**

The General Form of a class definition

```
class class-name
{
private:
    variable declaration;
    function declaration;
```



protected:

variable declaration;
function declaration;

public:

variable declaration;
function declaration;

};

- The class body contains the declaration of its members (Data member and Member functions).
- The class body has three access specifiers (visibility labels) viz., private , public and protected.

Class Access Specifiers

Data hiding is one of the important features of Object Oriented Programming which allows preventing the functions of a program to access directly the internal representation of a class type. The access restriction to the class members is specified by public, private, and protected sections within the class body. The keywords public, private, and protected are called access specifiers. The default access specifier for members is private.

The Public Members in child classes which are called derived classes A public member is accessible from anywhere outside the class but within a program. You can set and get the value of public data members even without using any member function.

The Private Members

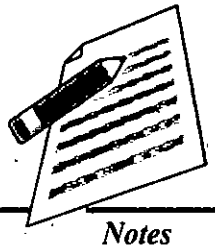
A private member cannot be accessed from outside the class. Only the class member functions can access private members. By default all the members of a class would be private.

The Protected Members

A protected member is very similar to a private member but it Provides one additional benefit that they can be accessed (inherited classes).

Example

```
                Keyword class intimates the compiler that it is a class definition
                Class name or tag name acts as a user defined data type. Using this,
                object of the same class type will be created.
class student
{
Private:                                    These are private access specifier members
      char name [10];                       That means these members cannot be accessed
      int rollno, mark1, mark2, total;      from outside
Protected:
      void accept();                        These are protected access specifier members.
      void compute();                       These members also cannot be accessed from
public:                                     outside
      void display(); ------------------>   Members under this specifier can be accessed
};                                          from outside
```



Notes

Definition of class members

Class comprises of members. Members are classified as Data Members and Member functions. Data members are the data variables that represent the features or properties of a class. Member functions are the functions that perform specific tasks in a class. Member functions are called as methods, and data members are also called as attributes.

Example

```
Class result                          Data members
{
Private;
        char name [10];
        int rollno,mark1, mark2, total;

Public:
        void accept();
        void display();
};
                                      Member functions
```

Classes also contain some special member functions called as constructors and destructors.

Defining methods of a class

Without defining the methods (functions), class definition will become incomplete. The member functions of a class can be defined in two ways.

(1) Inside the class definition

(2) Outside the class definition

(1) Inside the class definition

When a member function is defined inside a class, it behaves like inline functions. These are called Inline member functions.

If a function is inline, the compiler places a copy of the code of that function at each point where the function is called at compile time.

(2) Outside the class definition

When Member function defined outside the class just like normal function definition (Function definitions you are familiar with) then it is be called as **outline member function or non-inline member function. Scope resolution operator (::) is used for this purpose.** The syntax for defining the outline member function is

Syntax

return_type class_name :: function_name (parameter list)

```
{
    function definition
}
```

For example:

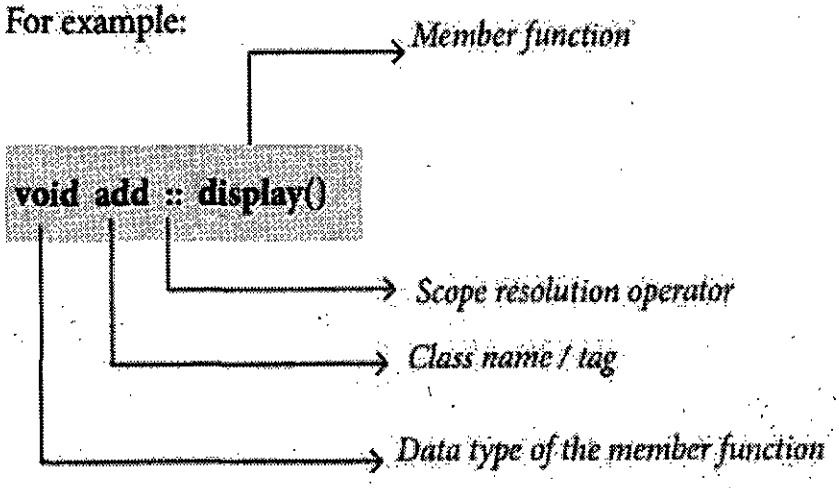
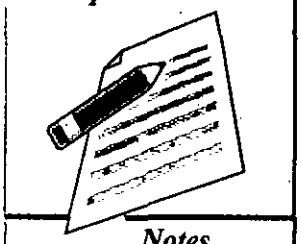


Illustration 14.1 Inline and Outline member function

Absence of access specifier means that members are private by default..

```
# include <iostream>
using namespace std;
class Box
{
    double width; // no access specifier mentioned
public:
    double length;
void printWidth( ) //inline member function definition
    {
        cout<<"\n The width of the box is..."<<width;
    }
void setWidth( double w ); //prototype of the function
};
void Box :: setWidth(double w) // outline member function definition
{
width=w;
}
int main( )
{
```




```
Box b;              // object for class Box
b.setWidth(10.0);   // Use member function to set the width.
b.printWidth( );    //Use member function to print the width.
return 0;
}
```

Output:

The width of the box is... 10

Declaring a member function having many statements, looping construct, switch or goto statement as inline is not advisable.

Creating Objects

A class specification just defines the properties of a class. To make use of a class specified, the variables of that class type have to be declared. The class variables are called object. Objects are also called as instance of class.

For example

```
student s;
```

In the above statement **s** is an instance of the class **student.**

Objects can be created in two methods,

(1) Global object

(2) Local object

(1) Global Object

If an object is declared outside all the function bodies or by placing their names immediately after the closing brace of the class declaration then it is called as Global object. These objects can be used by any function in the program

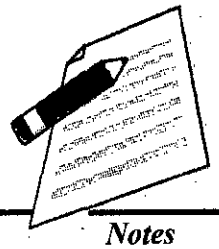
(2) Local Object

If an object is declared with in a function then it is called local object.
It cannot be accessed from outside the function.

Illustration 14.2 The use of local object

A global object can be declared only for global class. If a class definition is specified outside the body of all functions in a program then it is called global class

```
#include <iostream>
#include <conio>
```



Notes

```
using namespace
std class add        //Global class
{
int a,b;
public:
int sum;
void getdata()
{
a=5;
b=10;
sum = a+b;
}
} a1;
add a2;
int main()
{
add a3;      //Local object for a global class
a1.getdata();
a2.getdata();
a3.getdata();  //public data member accessed from outside the class
cout<<a1.sum;
cout<<a2.sum;
cout<<a3.sum;
return 0;
}
```

Output:

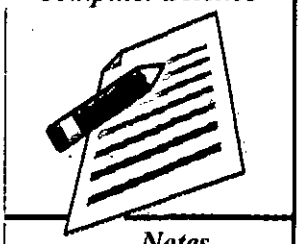
151515

Memory allocation of objects

The member functions are created and placed in the memory space only when they are defined as a part of the class specification. Since all the objects belonging to that class use the same member function, **no separate space is allocated for member functions when the objects are created. Memory space required for the member variables are only allocated separately for each object** because the member variables will hold different data values for different objects

Illustration 14.3 Memory allocation for objects

```
include <iostream>
using namespace std;
```



Notes

```
//The members will be allocated with memory space only after the creation of the
class type object
class product
{
    int code, quantity;
    float price;
    public:
    void assignData();
    void Print();
};
int main()
{
    product p1, p2;
    cout<<"\n Memory allocation for object p1 " <<sizeof(p1);
    cout<<"\n Memory allocation for object p2 " <<sizeof(p2);
    return 0;
}
```

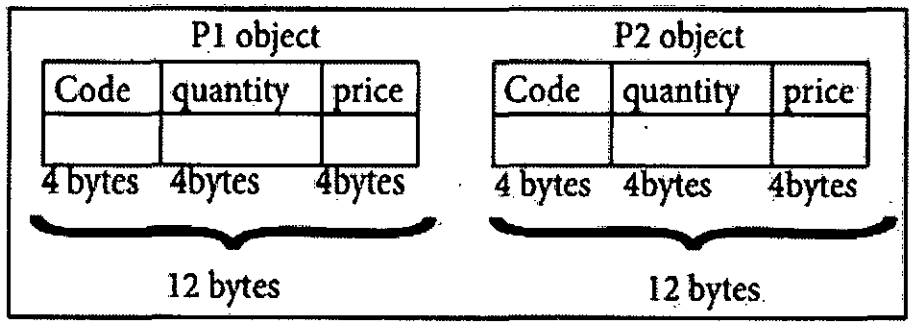
Output:

Memory allocation for object p1 12

Memory allocation for object p2 12

Member functions assign Data( ) and Print( ) belong to the common pool in the sense both the objects p1 and p2 will have access to the code area of the common pool.

Memory for Objects for p1 and p2 is illustrated:

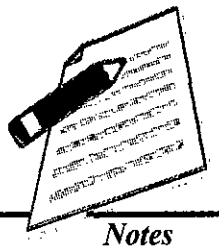


Introduction to Constructors

The definition of a class only creates a new user defined data type. The instances of the class type should be instantiated (created and initialized) . Instantiating object is done using constructor.

Need for Constructors

An array or a structure in c++ can be initialized during the time of their declaration.



For example

```
struct sum
{
    int n1,n2;
};
class add
{
    int num1,num2;
};
int main()
{
    int arr[]={1,2,3}; //declaration and initialization of array
    sum s1={1,1}; //declaration and initialization of structure object
   add a1={0,0}; // class object declaration and initialization throws compilation error
}
```

Member function of a class can access all the members irrespective of their associated access specifier.

The initialization of class type object at the time of declaration similar to a structure or an array is not possible because the class members have their associated access specifiers (private or protected or public). Therefore Classes include special member functions called as constructors. The constructor function initializes the class object.

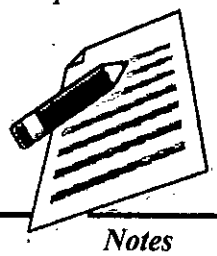
Declaration and Definition

When an instance of a class comes into scope, a special function called the constructor gets executed. The constructor function name has the same name as the class name. The constructors return nothing. They are not associated with any data type. It can be defined either inside class definition or outside the class definition.

Example 1:

Illustration 14.14 A constructor defined inside the class specification.

```
#include<iostream>
using namespace std;
class Sample
{
int i,j;
    public :
int k;
    Sample()
    {
    i=j=k=0;//constructor defined inside the class
    }
};
```



Notes

Illustration 14.15 A constructor defined inside the class specification.

```
#include<iostream>
using namespace std;
class Sample
{
int i,j;
    public :
int k;
    Sample()
    {
    i=j=k=0;//constructor defined inside the class
    }
};
int main()
{
Sample s1;
return 0;
}
```

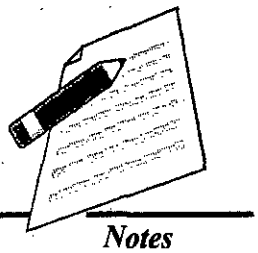
Example 2:

Illustration 14.16 A constructor defined outside the class specification.

```
#include<iostream>
using namespace std;
class Data
    {
int p,q;
    public :
int r;
    Data();  //only prototype to be specified here to intimate its access specifier
    };
Data ::Data()
{
    p=q=r=0;                // constructor defined outside the class
}
int main()
{
    Data d1;
return 0;
}
```

Illustration 14.17 illustrate a constructor defined inside the private visibility.

```
#include<iostream>
using namespace std;
```




```
class X
{
    int num;
    X()
{
    num=k=0;
}
public:
int k;
};
int main()
{
X x;   // The constructor of X cannot accessed by main() because main() is a
    //non member function
    //and the compiler throws error message          [Error] ‹X::X()› is private
return 0;
}
```

Functions of constructor

As we know now that the constructor is a special initialization member function of a class that is called automatically whenever an instance of a class is declared or created. The main function of the constructor is

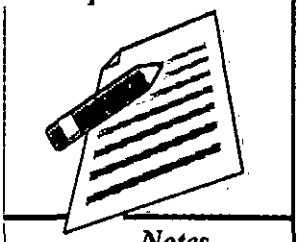
1) To allocate memory space to the object and
2) To initialize the data member of the class object

There is an alternate way to initialize the class objects but in that case we have to explicitly call the member function.

Illustration 14.18 illustrate a member function initializes the data member.

After creating the object the get value () should be explicitly called to initialize the object.

```
#include<iostream>
using namespace std;
class Sample
{
    int i, j;
    public :
    int k;
    void get value()
    {
    i=j=k=0; //member function
    }
int main()
{
Sample s1;
    s1.getvalue();              //member function initializes the class object
return 0;
}
```



Notes

Default Constructors

A constructor that accepts no parameter is called default constructor. For example in the class data program Data ::Data() is the default constructor . Using this constructor Objects are created similar to the way the variables of other data types are created.

Example

```
int num;     //ordinary variable declaration
Data d1;     // object declaration
```

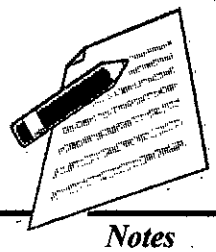
If a class does not contain an explicit constructor (user defined constructor) the compiler automatically generate a default constructor implicitly as an inline public member.

Illustration 14.19 illustrate the compiler generated constructor

```
#include<iostream>
using namespace std;
class Sample
{
    int i, j;
    public:
    int k; //no user defined constructor in this program
    void get value()//member function
    {
       i=j=k=0;
    }
};
int main()
{
    Sample s1; //uses the default constructor generated by the compiler
    s1.getvalue();
    return 0;
}
```

Illustration 14.20 to illustrate the constructor and other member function in a class

```
#include<iostream>
using namespace std;
class simple
{
private:
    int a,b;
public:
simple()
{
    a= 0 ;
    b= 0;
    cout<< «\n Constructor of class-simple «;
}
```




```
    void getdata()
    {
    cout<<»\n Enter values for a and b (sample data 6 and 7)... «;
    cin>>a>>b;
    }
    void put data()
    {
    cout<<»\nThe two integers are... «<<a<<›\t›<< b<<endl;
    cout<<»\n The sum of the variables «<<a<<» + «<<b<<» = «<<a+b;
    }
};
int main()
{
    simple s;
    s.getdata();
    s.putdata();
    return 0;
}
```

Output:

Constructor of class-simple

Enter values for a and b (sample data 6 and 7)... 6 7

The two integers are... 6 7

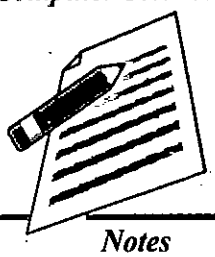
The sum of the variables 6 + 7 = 13

Parameterized Constructors

A constructor which can take arguments is called parameterized constructor .This type of constructor helps to create objects with different initial values. This is achieved by passing parameters to the function.

Illustration 14.21 to illustrate the Parameterized constructor used for creating objects

```
#include<iostream>
using namespace std;
class simple
{
private:
    int a,b;
public:
simple(int m, int n)
{
a= m ;
b= n;
cout<< "\n Parameterized Constructor of class-simple "<<endl;
}
```



Notes

```
void put data()
{
cout<<"\nThe two integers are... "<<a<<'\t'<< b<<endl;
cout<<"\n The sum of the variables "<<a<<" + "<<b<<" = "<< a+b; }
};
int main()
{
simple s1(10,20),s2(30,45); //Created two objects with different values created
cout<<"\n\t\tObject 1\n";
s1.putdata();
cout<<"\n\t\tObject 2\n";
s2.putdata();
return 0;
}
```

Output:

Parameterized Constructor of class-simple

Object 1

The two integers are .. 10 20

The sum of the variables 10 + 20 = 30

Object 2

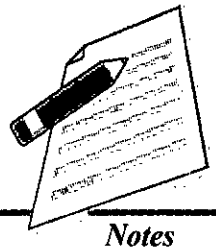
The two integers are... 30 45

The sum of the variables 30 + 45 = 75

Illustration 14.22 to illustrate the creation of object with no argument after defining parameterized constructor throws error

Note:- Just like normal function parameterized constructor can also have default arguments.

```
#include<iostream>
using namespace std;
class simple
{
private:
    int a,b;
public:
simple(int m, int n)
{
a= m ;
b= n;
cout<< "\n Parameterized Constructor of class-simple "<<endl;
}
void putdata()
{
```



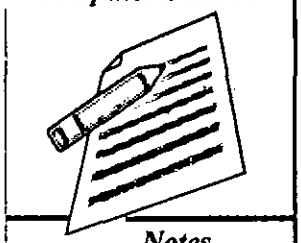

```
cout<<"\nThe two integers are .. "<<a<<'\t'<< b<<endl;
cout<<"\n The sum of the variables "<<a<<" + "<<b<<" = "<< a+b; }
};
int main()
{
simple s,s1(10,20)     // [Error] no matching function for call to 'simple::simple()'
s1.putdata();
s2.putdata();
return 0;
}
```

Illustration 14.23 to illustrate the default argument in parameterized constructor

```
#include<iostream>
using namespace std;
class simple
{
private:
int a,b;
public:
simple(int m, int n=100)   //default argument
{
a= m ;
b= n;
cout<< "\n Parameterized Constructor with default argument"<<endl;
}
void putdata()
{
cout<<"\nThe two integers are... "<<a<<'\t'<< b<<endl;
cout<<"\n The sum of the variables "<<a<<" + "<<b<<" = "<< a+b; }
};
int main()
{
simple s1(10,20),s2(50);
cout<<"\n\t\tObject 1 with both values \n";
s1.putdata();
cout<<"\n\t\tObject 2 with one value and one deafult value\n";
s2.putdata();
return 0;
}
```

Output:

```
Parameterized Constructor with default argument
    Object 1 with both values
The two integers are... 10      20
```



Notes

The sum of the variables 10 + 20 = 30

Object 2 with one value and one deafult value

The two integers are... 50 100

The sum of the variables 50 + 100 = 150

Destructors

When a class object goes out of scope, a special function called the destructor gets executed. The destructor has the same name as the class tag but prefixed with a **~(tilde)**! Destructor function also return nothing and it does not associated with any data type.

Need of Destructors

The purpose of the destructor is to free the resources that the object may have acquired during its lifetime. A destructor function removes the memory of an object which was allocated by the constructor at the time of creating a object.

Declaration and Definition

A destructor is a special member function that is called when the lifetime of an object ends and destroys the object constructed by the constructor. Normally declared under public.

Illustration14.29 to illustrate destructor function in a class

```
#include<iostream>
using namespace std;
class simple
{
private:
int a, b;
public:
simple()
{
a= 0 ;
b= 0;
cout<< "\n Constructor of class-simple ";
}
void getdata()
{
cout<<"\n Enter values for a and b (sample data 6 and 7)... ";
cin>>a>>b;
}
void putdata()
{
cout<<"\nThe two integers are .. "<<a<<'\t'<< b<<endl;
cout<<"\n The sum of the variables "<<a<<" + "<<b<<" = "<<a+b;
}
```


```
~simple()
{ cout<<"\n Destructor is executed to destroy the object";} };
int main()
{
simple s;
s.getdata();
s.putdata();
return 0;
}
```

Output:

Constructor of class-simple

Enter values for a and b (sample data 6 and 7)... 6 7

The two integers are .. 6 7

The sum of the variables 6 + 7 = 13

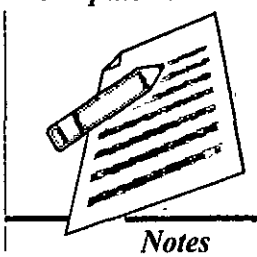
Destructor is executed to destroy the object

Characteristics of Destructors

- The destructor has the same name as that of the class prefixed by the tilde character '~'.
- The destructor cannot have arguments
- It has no return type
- Destructors cannot be overloaded i.e., there can be only one destructor in a class
- In the absence of user defined destructor, it is generated by the compiler
- The destructor is executed automatically when the control reaches the end of class scope to destroy the object
- They cannot be inherited

SUMMARY

Class is a way to bind the data and its associated functions together. Object is an instance of a class. Member function can be defined inside the class and outside the class. Member function can be called by using its name inside another member function of the same class. This is known as nesting of member functions. Constructor is a special member function that initializes objects of its class. It is special because its name is the same as the class name. Constructor that accepts no parameter is called default constructor. Constructors that take arguments are called parameterized constructors. Destructor is used to destroy the objects that have been created by a constructor.



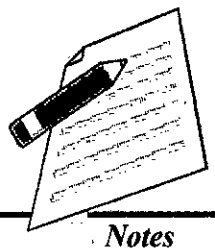
EXERCISE

MCQ

1. The variables declared inside the class are known as data members and the functions are known as
   (A) data functions (B) inline functions
   (C) member functions (D) attributes
2. Which of the following statements about member functions are True or False?
   i) A member function can call another member function directly with using the dot operator.
   ii) Member function can access the private data of the class.
   (A) i-True, ii-True **(B) i-False, ii-True**
   (C) i-True, ii-False (D) i-False, ii-False
3. A member function can call another member function directly, without using the dot operator called as
   (A) *sub function* (B) *sub member*
   (C) nesting of member function (D) sibling of member function
4. The *member function defined within the class behave like*
   (A) inline functions
   (B) Non inline function
   (C) Outline function
   (D) Data function
5. Which of the following access specifier protects data from inadvertent modifications?
   (A) Private (B) Protected
   (C) Public (D) Global

Review Questions

1. What do you understand by visibility modes in class derivations ? What are these modes?
2. What are the special properties of a constructor function?
3. What is parameterized constructor?
4. What is copy constructor ?
5. What is the importance of destructors ?



14 INHERITANCE EXTENDING CLASSES

- Understand the concept of inheritance.
- Discuss the concept of base class.
- Describe the types of inheritance.
- Discuss the concept of super class.
- Discuss the concept of sub class.

Objective of the chapter:

The basic objective of this chapter is to through some light on the initial concepts of inheritance and classes so that the types and applications of these concepts can be learned.

Introduction

Inheritance is one of the most important features of Object Oriented Programming is Inheritance. In object-oriented programming, inheritance enables new class and its objects to take on the properties of the existing classes. A class that is used as the basis for inheritance is called a super class or base class. A class that inherits from a super class is called a subclass or derived class.

Need for Inheritance

Inheritance is an important feature of object oriented programming used for code reusability. It is a process of creating new classes called derived classes, from the existing or base classes. Inheritance allows us to inherit all the code (except declared as private) of one class to another class. The class to be inherited is called base class or parent class and the class which inherits the other class is called derived class or child class. The derived class is a power packed class, as it can add additional attributes and methods and thus enhance its functionality.

Types of Inheritance

There are different types of inheritance viz., Single Inheritance, Multiple inheritance, Multilevel inheritance, hybrid inheritance and hierarchical inheritance.

1. Single Inheritance

When a derived class inherits only from one base class, it is known as single inheritance

2. Multiple Inheritance

When a derived class inherits from multiple base classes it is known as multiple inheritance

3. Hierarchical inheritance

When more than one derived classes are created from a single base class , it is known as Hierarchical inheritance.

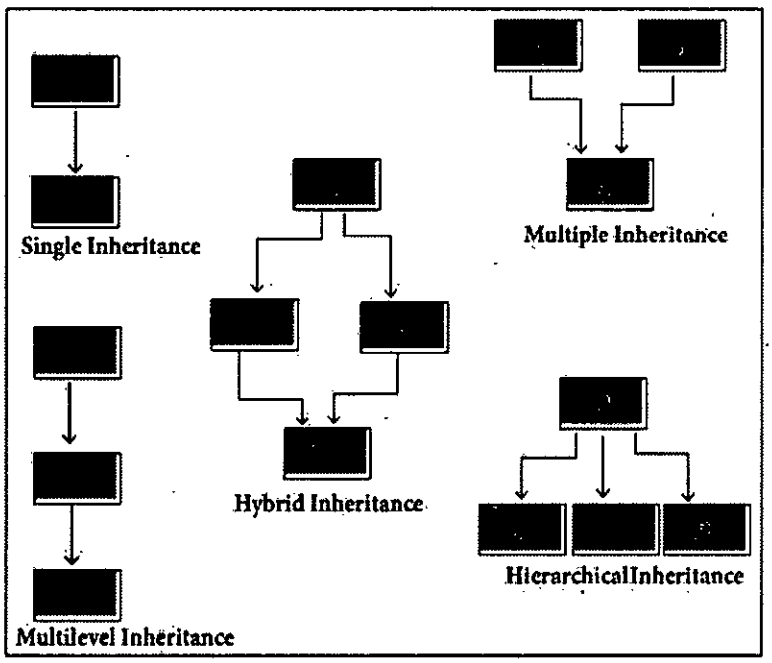
4. Multilevel Inheritance

The transitive nature of inheritance is itself reflected by this form of inheritance. When a class is derived from a class which is a derived class – then it is referred to as multilevel inheritance.

5. Hybrid inheritance

When there is a combination of more than one type of inheritance, it is known as hybrid inheritance. Hence, it may be a combination of Multilevel and Multiple inheritance or Hierarchical and Multilevel inheritance or Hierarchical, Multilevel and Multiple inheritance.

The following diagram represents the different types of inheritance



Derived Class and Base class

While defining a derived class, the derived class should identify the class from which it is derived. The following points should be observed for defining the derived class.

i. The keyword class has to be used

ii. The name of the derived class is to be given after the keyword class

iii. A single colon

iv. The type of derivation (the visibility mode), namely private, public or protected. If no visibility mode is specified , then by default the visibility mode is considered as private.

v. The names of all base classes (parent classes) separated by comma.

class derived_class_name :visibility_mode base_class_name

{

// members of derivedclass

};

The following are some of the examples for different forms of inheritance

Single Inheritance

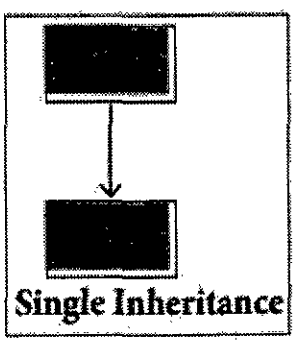
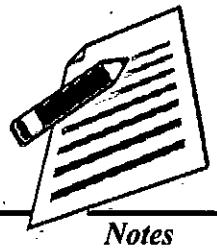


Illustration 16.1 single inheritance

Though the derived class inherits all the members of base class ,it has access privilege only to non-private members of the base class .

```
# include <iostream>
using namespace std;
class student        //base class
{
private :
char name[20];
int rno;
public:
void acceptname()
{
cout<<"\n Enter roll no and name .. ";
cin>>rno>>name;
}
void display name()
{
cout<<"\n Roll no :-"<<rno;
cout<<"\n Name :-"<<name<<endl;
}
```




```
};
class exam : public student      //derived class with single base class
{
public:
    int mark1, mark2 ,mark3,mark4,mark5,mark6,total;
void acceptmark()
{
cout<<"\n Enter lang,eng,phy,che,csc,mat marks.. ";
 cin>>mark1>>mark2>>mark3>>mark4>>mark5>>mark6;
}
void displaymark()
{
cout<<"\n\t\t Marks Obtained ";
cout<<"\n Language.. "<<mark1;
cout<<"\n English .. "<<mark2;
cout<<"\n Physics .. "<<mark3;
cout<<"\n Chemistry.. "<<mark4;
cout<<"\n Comp.sci.. "<<mark5;
cout<<"\n Maths ..  "<<mark6;
}
};
int main()
{
exam e1;
e1.acceptname();    //calling base class function using derived class object
e1.acceptmark();
e1.displayname();    //calling base class function using derived class object
e1.displaymark();
return 0;
}
```

Output

```
Enter roll no and name .. 1201 KANNAN
Enter lang,eng,phy,che,csc,mat marks.. 100 100 100 100 100 100
Roll no :-1201
Name :-KANNAN
Marks Obtained
Language.. 100
English .. 100
Physics .. 100
Chemistry.. 100
Comp.sci.. 100
Maths ..  100
```

In the above program the derived class "exam" inherits all the members of the base class "student". But it has access privilege only to the non-private members of the base class.

Multiple Inheritance

Program to illustrate Multiple inheritance

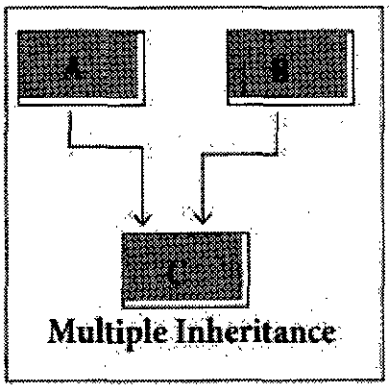
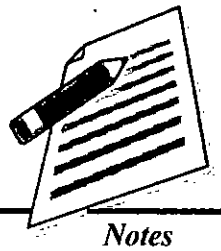


Illustration 16.2 Multiple inheritance

The order of inheritance by derived class to inherit the base class is left to right.

```
#include <iostream>
using namespace std;
class student //base class
{
private :
char name[20];
int rno;
public:
void acceptname()
{
cout<<"\n Enter roll no and name .. ";
cin>>rno>>name;
}
void displayname()
{
cout<<"\n Roll no :-"<<rno;
cout<<"\n Name :-"<<name<<endl;
}
};
class detail                    //Base class
{
    int dd,mm,yy;
    char cl[4];
public:
void acceptdob()
```



Notes

```
{
cout<<"\n Enter date,month,year in digits and class .. ";
cin>>dd>>mm>>yy>>cl;
}
void displaydob()
{
cout<<"\n class:-"<<cl;
cout<<"\t\t DOB        : "<<dd<<" - "<<mm<<" –" <<yy<<endl;
}
};
class exam : public student,public detail //derived class with multiple
base class
{
    public:
    int mark1, mark2 ,mark3,mark4,mark5,mark6,total;
    void acceptmark()
    {
    cout<<»\n Enter lang,eng,phy,che,csc,mat marks.. «;
    cin>>mark1>>mark2>>mark3>>mark4>>mark5>>mark6;
    }
void displaymark()
{
cout<<"\n\t\t Marks Obtained ";
cout<<"\n Language.. "<<mark1;
cout<<"\n English .. "<<mark2;
cout<<"\n Physics .. "<<mark3;
cout<<"\n Chemistry.. "<<mark4;
cout<<"\n Comp.sci.. "<<mark5;
cout<<"\n Maths ..  "<<mark6;
}
};
int main()
{
exam e1;
e1.acceptname();  //calling base class function using derived class object
e1.acceptdob();    //calling base class function using derived class object
e1.acceptmark();
e1.displayname(); //calling base class function using derived class object
e1.displaydob();   //calling base class function using derived class object
e1.displaymark();
return 0;
}
```

Output:

```
Enter roll no and name .. 1201 MEENA
Enter date,month,year in digits and class .. 7 12 2001 XII
Enter lang,eng,phy,che,csc,mat marks.. 96 98 100 100 100 100
Roll no :-1201
Name :- MEENA
class :-XII      DOB      : 7 - 12 -2001
Marks Obtained
Language.. 96
English .. 98
Physics .. 100
Chemistry.. 100
Comp.sci.. 100
Maths .. 100
```

In the above program the class "exam" is derived from class "student" and "detail" Hence it access all the members of both the classes.

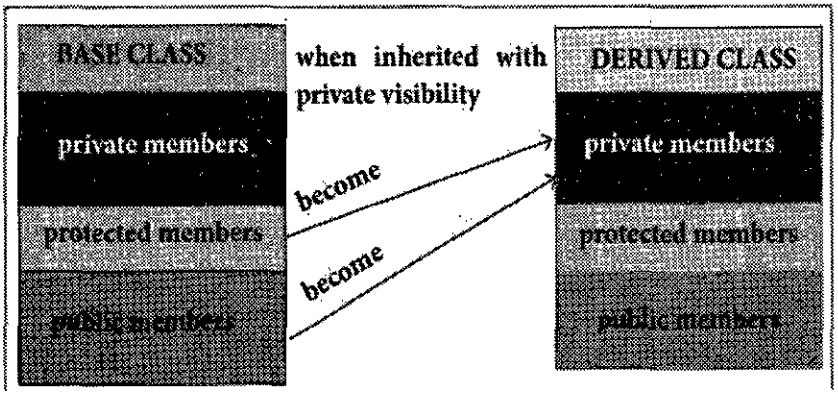
VISIBILITY MODES

An important feature of Inheritance is to know which member of the base class will be acquired by the derived class. This is done by using visibility modes.

The accessibility of base class by the derived class is controlled by visibility modes. The three visibility modes are private, protected and public. The default visibility mode is private. Though visibility modes and access specifiers look similar, the main difference between them is Access specifiers control the accessibility of the members with in the class whereas visibility modes control the access of inherited members with in the class.

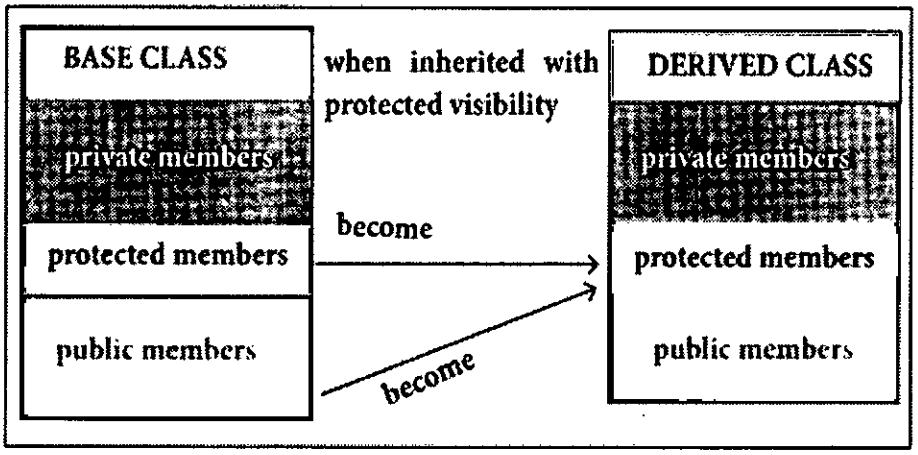
Private visibility mode

When a base class is inherited with private visibility mode the public and protected members of the base class become 'private' members of the derived class



Protected visibility mode

When a base class is inherited with protected visibility mode the protected and public members of the base class become 'protected members ' of the derived class



Public visibility mode

When a base class is inherited with public visibility mode , the protected members of the base class will be inherited as protected members of the derived class and the public members of the base class will be inherited as public members of the derived class.

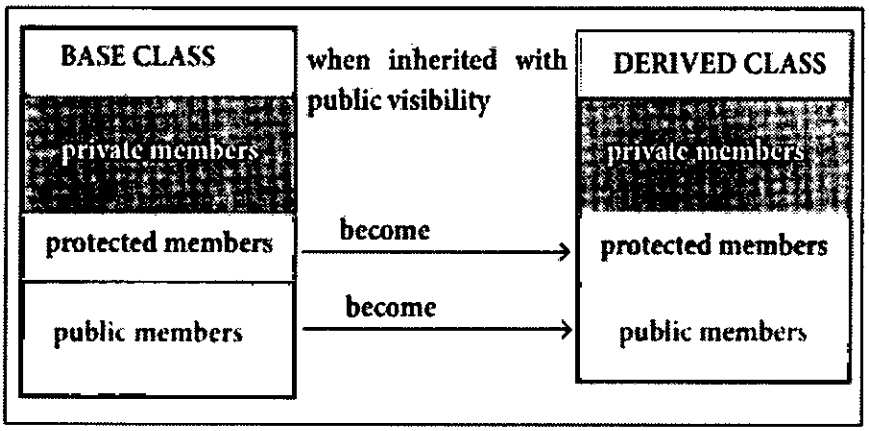
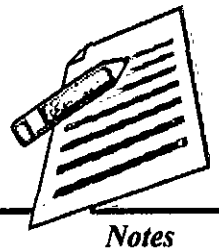


Illustration 16.6 explains the significance of different visibility modes.

```
//Implementation of Single Inheritance using public visibility mode
#include <iostream>
using namespace std;
class Shape
{
private:
    int count;
protected:
    int width;
    int height;
```




```
public:
    void setWidth(int w)
    {
    width = w;
    }
 void setHeight(int h)
{
height = h;
}
};
class Rectangle: publicShape
{
public:
int getArea()
{
return (width * height);
}
};
int main()
{
Rectangle Rect;
Rect.setWidth(5);
Rect.setHeight(7);
// Print the area of theobject.
cout<< "Total area: "<<Rect.getArea() <<endl;
return 0;
}
```

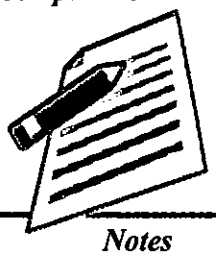
Output

Total area: 35

The following table contain the members defined inside each class before inheritance

MEMBERS of class	visibility modes		
	Private	Protected	Public
Shape(base class)	int count;	int width; int height;	void setWidth(int) void setHeight(int)
Rectangle (derived class only with its defined members)			intgetArea();

The following table contain the details of members defined after inheritance



MEMBERS of class	visibility modes –public for acquiring the properties of the base class		
	Private	Protected	Public
Shape(base class)	int count;	int width; int height;	void setWidth(int) void setHeight(int)
Rectangle (derived class acquired the properties of base class with public visibility)	Private members of base classes are not directly accessible by the derived class	int width; int height;	int getArea(); void setWidth(int) void setHeight(int)

Suppose the class **rectangle is derived with protected visibility** then the properties of class rectangle will change as follows

MEMBERS of class	visibility modes –protected for acquiring the properties of the base class		
	Private	Protected	Public
Shape(base class)	int count;	int width; int height;	void setWidth(int) void setHeight(int)
Rectangle (derived class acquired the properties of base class with protected visibility)	Private members of base classes are not directly accessible by the derived class	int width; int height; void setWidth(int) void setHeight(int)	intgetArea();

In case the class rectangle is derived with private visibility mode from its base class shape then the property of class rectangle will change as follows

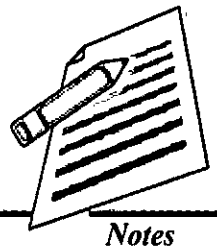
MEMBERS of class	visibility modes –private for acquiring the properties of the base class		
	Private	Protected	Public
Shape(base class)	int count;	int width; int height;	void setWidth(int) void setHeight(int)
Rectangle (derived class acquired the properties of base class with private visibility)	Private members of base classes are not directly accessible by the derived class	int width; int height; void setWidth(int) void setHeight(int)	int getArea();

When you derive the class from an existing base class, it may inherit the properties of the base class based on its visibility mode. So one must give appropriate visibility mode depends up on the need.

Private inheritance should be used when you want the features of the base class to be available to the derived class but not to the classes that are derived from the derived class.

Protected inheritance should be used when features of base class to be available only to the derived class members but not to the outside world.

Public inheritance can be used when features of base class to be available the derived class members and also to the outside world.



SUMMARY

The process of creating a new class i.e., derived class from an existing class is called as inheritance. There are five types of inheritance: single inheritance, multiple inheritance, multilevel inheritance, hybrid inheritance, hierarchical inheritance. A derived class with only one base class is called as single inheritance. A derived class with several base classes is called as multiple inheritance. Multilevel inheritance: The mechanism of deriving a class from another derived class is called multilevel inheritance. Hierarchical inheritance: One class may be inherited by more than one class. This process is known as hierarchical inheritance. Hybrid inheritance: It is a combination of two or more types of inheritance. The private data of base class cannot be inherited. An abstract class is one that is not used to create objects. An abstract class is designed only to act as a base class. Virtual base classes are used for preventing multiple "instances" of a given class appearing in an inheritance.

EXERCISE

MCQ

1. Which of the following is the process of creating new classes from an existing class
   (a) Polymorphism **(b) Inheritance**
   (c) Encapsulation (d) super class

2. Which of the following derives a class student from the base class school
   (a) school: student
   (b) class student : public school
   (c) student : public school
   (d) class school : public student

3. The type of inheritance that reflects the transitive nature is
   (A) Single Inheritance (B) Multiple Inheritance
   (C) Multilevel Inheritance (D) Hybrid Inheritance

4. Which visibility mode should be used when you want the features of the base class to be available to the derived class but not to the classes that are derived from the derived class?
   (A) Private (B) Public
   (C) Protected (D) All of these

5. Inheritance is process of creating new class from
   (A) Base class (B) abstract
   (C) derived class (D) Function

Notes

6. A class is derived from a class which is a derived class itself, then this is referred to as

   (A) multiple inheritance
   (B) multilevel inheritance
   (C) single inheritance
   (D) double inheritance

7. Which amongst the following is executed in the order of inheritance?

   (A) Destructor
   (B) Member function
   (C) Constructor
   (D) Object

8. Which of the following is true with respect to inheritance?

   (A) Private members of base class are inherited to the derived class with private

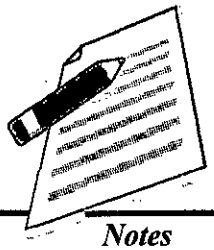
   (B) Private members of base class are not inherited to the derived class with private accessibility

   (C) Public members of base class are inherited but not visible to the derived class

   (D) Protected members of base class are inherited but not visible to the outside class

Review Questions

1. What are the needs of inheritance?
2. What are the three modes of inheritance? Explain.
3. What is a virtual base class? Explain it by taking an example.



Notes

15 POINTER

- Understand the concept of pointer.
- Describe the types of pointer.
- Discuss the need of pointer.
- Discuss the application of pointer.

Objective of the chapter:

The basic objective of this chapter is to through some light on the initial concepts of pointer so that the types and applications of pointer can be learned.

Introduction

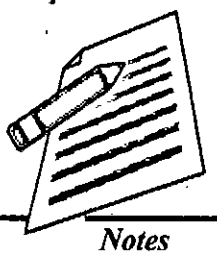
C Pointer is a variable that stores/points the address of the another variable. C Pointer is used to allocate memory dynamically i.e. at run time. The variable might be any of the data type such as int, float, char, double, short etc.

Syntax : data_type *var_name; Example : int *p; char *p;

Where, * is used to denote that "p" is pointer variable and not a normal variable.

Key points to remember about pointers in C:

- Normal variable stores the value whereas pointer variable stores the address of the variable.
- The content of the C pointer always be a whole number i.e. address.
- Always C pointer is initialized to null, i.e. int *p = null.
- The value of null pointer is 0.
- & symbol is used to get the address of the variable.
- * symbol is used to get the value of the variable that the pointer is pointing to.
- If pointer is assigned to NULL, it means it is pointing to nothing.
- The size of any pointer is 2 byte (for 16 bit compiler).
- No two pointer variables should have the same name.
- But a pointer variable and a non-pointer variable can have the same name.



1 Pointer –Initialization:

Assigning value to pointer:

It is not necessary to assign value to pointer. Only zero (0) and NULL can be assigned to a pointer no other number can be assigned to a pointer. Consider the following examples;

int *p=0;

int *p=NULL; The above two assignments are valid. int *p=1000; This statement is invalid.

Assigning variable to a pointer:

int x; *p; p = &x;

This is nothing but a pointer variable p is assigned the address of the variable x. The address of the variables will be different every time the program is executed.

Reading value through pointer:

int x=123; *p; p = &x;

Here the pointer variable p is assigned the address of variable x. printf("%d", *p); will display value of x 123. This is reading value through pointer printf("%d", p); will display the address of the variable x.

printf("%d", &p); will display the address of the pointer variable p. printf("%d",x); will display the value of x 123.

printf("%d", &x); will display the address of the variable x.

Note: It is always a good practice to assign pointer to a variable rather than 0 or NULL.

Pointer Assignments:

We can use a pointer on the right-hand side of an assignment to assign its value to another variable.

Example: int main()

{

int var=50; int *p1, *p2; p1=&var; p2=p1;

}

Chain of pointers/Pointer to Pointer:

A pointer can point to the address of another pointer. Consider the following example.

int x=456, *p1, **p2; //[pointer-to-pointer];

p1 = &x;

p2 = &p1;

printf("%d", *p1); will display value of x 456.

printf("%d", *p2); will also display value of x 456. This is because p2 point p1, and p1 points x. Therefore p2 reads the value of x through pointer p1. Since one pointer is points towards another pointer it is called chain pointer. Chain pointer must be declared with ** as in **p2.

Manipulation of Pointers

We can manipulate a pointer with the indirection operator „*□, which is known as dereference operator. With this operator, we can indirectly access the data variable content.

Syntax: *ptr_var;

Example:

```
#include<stdio.h> void main()
{
int a=10, *ptr; ptr=&a;
printf("\n The value of a is ",a);
*ptr=(*ptr)/2;
printf("The value of a is.",(*ptr));
 }
```

Output:

The value of a is: 10

The value of a is: 5

2 Pointer Expression & Pointer Arithmetic

C allows pointer to perform the following arithmetic operations:

A pointer can be incremented / decremented.

Any integer can be added to or subtracted from the pointer.

A pointer can be incremented / decremented.

In 16 bit machine, size of all types[data type] of pointer always 2 bytes. Eg: int a;
int *p; p++;

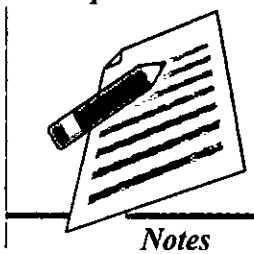
Each time that a pointer p is incremented, the pointer p will points to the memory location of the next element of its base type. Each time that a pointer p is decremented, the pointer p will points to the memory location of the previous element of its base type.

```
int a,*p1, *p2, *p3; p1=&a;
p2=p1++;
p3=++p1;
printf("Address of p where it points to %u", p1);        1000
printf("After incrementing Address of p where it points to %u", p1);        1002
printf("After assigning and incrementing p %u", p2);    1000
printf("After incrementing and assigning p %u", p3);    1002
```

In 32 bit machine, size of all types of pointer is always 4 bytes.

The pointer variable p refers to the base address of the variable a. We can increment the pointer variable,

p++ or ++p



This statement moves the pointer to the next memory address. let p be an integer pointer with a current value of 2,000 (that is, it contains the address 2,000). Assuming 32-bit integers, after the expression

p++;

the contents of p will be 2,004, not 2,001! Each time p is incremented, it will point to the next integer. The same is true of decrements. For example,

p--; will cause p to have the value 1,996, assuming that it previously was 2,000. Here is why: Each time that a pointer is incremented, it will point to the memory location of the next element of its base type. Each time it is decremented, it will point to the location of the previous element of its base type.

Any integer can be added to or subtracted from the pointer.

Like other variables pointer variables can be used in expressions. For example if p1 and p2 are properly declared and initialized pointers, then the following statements are valid.

y=*p1**p2;

sum=sum+*p1; z= 5* - *p2/p1; *p2= *p2 + 10;

C allows us to add integers to or subtract integers from pointers as well as to subtract one pointer from the other. We can also use short hand operators with the pointers p1+=; sum+=*p2; etc., we can also compare pointers by using relational operators the expressions such as p1 >p2 , p1==p2 and p1!=p2 are allowed.

/*Program to illustrate the pointer expression and pointer arithmetic*/

#include< stdio.h > #include<conio.h> void main()

{

int ptr1,ptr2; int a,b,x,y,z; a=30;b=6; ptr1=&a; ptr2=&b;

x=*ptr1+ *ptr2 –6; y=6*- *ptr1/ *ptr2 +30;

printf("\nAddress of a + %u",ptr1); printf("\nAddress of b %u", ptr2); printf("\na=%d, b=%d", a, b); printf("\nx=%d,y=%d", x, y); ptr1=ptr1 + 70;

ptr2= ptr2;

printf("\na=%d, b=%d", a, b);

}

/* Sum of two integers using pointers*/

#include <stdio.h> int main()

{

int first, second, *p, *q, sum; printf("Enter two integers to add\n"); scanf("%d%d", &first, &second);

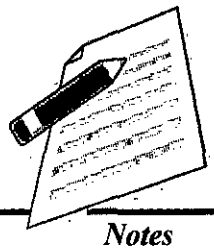
p = &first;

q = &second;

sum = *p + *q;

printf("Sum of entered numbers = %d\n",sum); return 0;

}



3 Pointers and Arrays

Array name is a constant pointer that points to the base address of the array[i.e the first element of the array]. Elements of the array are stored in contiguous memory locations. They can be efficiently accessed by using pointers.\Pointer variable can be assigned to an array. The address of each element is increased by one factor depending upon the type of data type. The factor depends on the type of pointer variable defined. If it is integer the factor is increased by 2.

Consider the following example:

int x[5]={11,22,33,44,55}, *p;

p = x; //p=&x; // p = &x[0];

Remember, earlier the pointer variable is assigned with address (&) operator. When working with array the pointer variable can be assigned as above or as shown below:

Therefore the address operator is required only when assigning the array with element. Assume the address on x[0] is 1000 then the address of other elements will be as follows

x[1] = 1002

x[2] = 1004

x[3] = 1006

x[4] = 1008

The address of each element increase by factor of 2. Since the size of the integer is 2 bytes the memory address is increased by 2 bytes, therefore if it is float it will be increase 4 bytes, and for double by 8 bytes. This uniform increase is called scale factor.

p = &x[0];

Now the value of pointer variable p is 1000 which is the address of array element x[0]. To find the address of the array element x[1] just write the following statement.

p = p + 1;

Now the value of the pointer variable p is 1002 not 1001 because since p is pointer variable the increment of will increase to the scale factor of the variable, since it is integer it increases by 2.

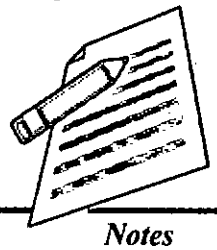
The p = p + 1; can be written using increment or decrement operator ++p; The values in the array element can be read using increment or decrement operator in the pointer variable using scale factor.

Consider the above example.

printf("%d", *(p+0)); will display value of array element x[0] which is 11. printf("%d", *(p+1)); will display value of array element x[1] which is 22. printf("%d", *(p+2)); will display value of array element x[2] which is 33. printf("%d", *(p+3)); will display value of array element x[3] which is 44. printf("%d", *(p+4)); will display value of array element x[4] which is 55.

/*Displaying the values and address of the elements in the array*/

```
#include<stdio.h>
void main()
{
int a[6]={10, 20, 30, 40, 50, 60};
```




```
int *p;
 int i;
p=a;
for(i=0;i<6;i++)
{
 printf("%d", *p); //value of elements of array printf("%u",p); //Address of array
}
getch();
}
```

/* Sum of elements in the Array*/

```
#include<stdio.h>
#include<conio.h> void main()
{
 int a[10]; int i,sum=0; int *ptr;
printf("Enter 10 elements:n");
for(i=0;i<10;i++)
scanf("%d",&a[i]);
ptr = a;     /* a=&a[0] */
 for(i=0;i<10;i++)
{
 sum = sum + *ptr; //*p=content pointed by ‹ptr› ptr++;
 }
 printf(«The sum of array elements is %d»,sum);
 }
```

***Sort the elements of array using pointers*/**

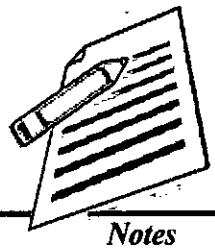
```
#include<stdio.h> int main(){
int  i,j, temp1,temp2;
 int arr[8]={5,3,0,2,12,1,33,2}; int *ptr;
 for(i=0;i<7;i++){ for(j=0;j<7-i;j++){
if(*(arr+j)>*(arr+j+1)){
 ptr=arr+j;
temp1=*ptr++;
temp2=*ptr; *ptr--=temp1; *ptr=temp2;
}}}
for(i=0;i<8;i++){
 printf(« %d»,arr[i]); } }
```

4 Pointers and Multi-dimensional Arrays

The array name itself points to the base address of the array.

Example:

```
int a[2][3];
int *p[2];
p=a;  //p points to a[0][0]
```




```
/*Displaying the values in the 2-d array*/
#include<stdio.h> void main()
{
int a[2][2]={{10, 20},{30, 40}}; int *p[2];
 int i,j; p=a;
 for(i=0;i<2;i++)
{
for(j=0;j<2;j++)
{
printf("%d", *(*(p+i)+j));  //value of elements of array
 }
}
getch();
}
```

5 Dynamic Memory Allocation

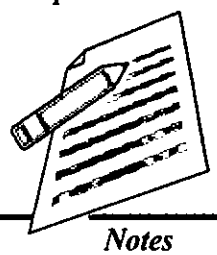
The process of allocating memory during program execution is called dynamic memory allocation.

Dynamic memory allocation functions

S. No.	*Function*	*Syntax*	*Use*
1	malloc()	ptr=(cast-type*)malloc(byte-size)	Allocates requested size of bytes and returns a pointer first byte of allocated space.
2	calloc()	ptr=(cast-type*)calloc(n,element-size);	Allocates space for an array elements, initializes to zero and then returns a pointer to memory.
3	free()	free(ptr);	dellocate the previously allocated space.
4	realloc()	ptr=realloc(ptr,newsize);	Change the size of previously allocated space.

SUMMARY

Pointer is a variable that represents the location (rather than the value) of a data item such as a variable or an array element. Object pointers are useful in creating objects at run time. Object pointers can be used to access the public members of an object. C++ uses a unique keyword called this to represent the object that invokes a member function.



Notes

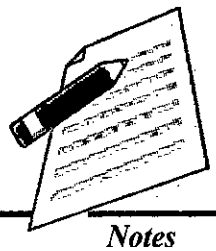
EXERCISE

MCQ

1. Which of the following is the correct way to declare a pointer ?
   A. int *ptr B. int ptr
   C. int &ptr D. All of the above
   Ans : A
2. Which of the following gives the [value] stored at the address pointed to by the pointer : ptr?
   A. Value(ptr) B. ptr
   C. &ptr D. *ptr
   Ans : D
3. A pointer can be initialized with
   A. Null B. Zero
   C. Address of an object of same type D.\ All of the above
   Ans : D
4. Choose the right option string* x, y;
   A. x is a pointer to a string, y is a string
   B. y is a pointer to a string, x is a string
   C. Both x and y are pointers to string types
   D. none of the above
   Ans : A
5. Generic pointers can be declared with_____ .
   A. auto B. void
   C. asm D. None of the above
   Ans : B

Review questions

1. If arr is an array of integers, why is the expression arr++ not legal?
2. What is the difference between arr [4] and *(arr+4)?
3. If a structure defined has pointer variable, how can it access the members of the structure ? Explain it by taking an example.
4. How can a data member and member function present in public mode in class be accessed through pointer object? Explain it by taking an example.
5. What is this pointer? Explain briefly.



Notes

16 FILES

- Understand the concept of file.
- Describe the types of files.
- Discuss how to create a file.
- Discuss how to open a file.

Objective of the chapter:

The basic objective of this chapter is to through some light on the initial concepts of files so that the types and applications of files can be learned.

Introduction

File is a named location on disk to store related information. It is used to permanently store data in a non-volatile memory (e.g. hard disk).

Since, random access memory (RAM) is volatile which loses its data when computer is turned off, we use files for future use of the data.

When we want to read from or write to a file we need to open it first. When we are done, it needs to be closed, so that resources that are tied with the file are freed. Hence, in Python, a file operation takes place in the following order.

1. Open a file
2. Read or write (perform operation)
3. Close the file

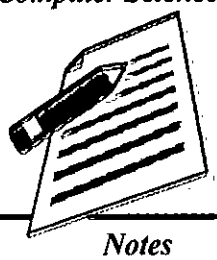
Opening a file

Python has a built-in function open() to open a file. This function returns a file object, also called a handle, as it is used to read or modify the file accordingly.

```
>>> f = open("test.txt") # open file in current directory
>>> f = open("C:/Python33/README.txt") # specifying full path
```

We can specify the mode while opening a file. In mode, we specify whether we want to read 'r', write 'w' or append 'a' to the file. We also specify if we want to open the file in text mode or binary mode.

The default is reading in text mode. In this mode, we get strings when reading from the file. On the other hand, binary mode returns bytes and this is the mode to be used when dealing with non-text files like image or exe files.



Notes

Python File Modes

Mode : Description

'r' : Open a file for reading. (default)

'w' : Open a file for writing. Creates a new file if it does not exist or truncates the file if it exists.

'x' : Open a file for exclusive creation. If the file already exists, the operation fails.

'a': Open for appending at the end of the file without truncating it. Creates a new file if it does not exist.

't' : Open in text mode. (default)

'b' : Open in binary mode.

'+' : Open a file for updating (reading and w

f = open("test.txt") # equivalent to 'r' or 'rt'

f = open("test.txt",'w') # write in text mode

f = open("img.bmp",'r+b') # read and write in binary mode

Hence, when working with files in text mode, it is highly recommended to specify the encoding type.

f = open("test.txt",mode = 'r',encoding = 'utf-8')

Closing a File

When we are done with operations to the file, we need to properly close it.

Closing a file will free up the resources that were tied with the file and is done using the close() method.

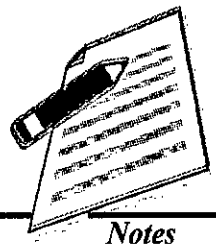
Python has a garbage collector to clean up unreferenced objects but, we must not rely on it to close the file.

```
f = open("test.txt",encoding = 'utf-8')
# perform file operations
f.close()
```

This method is not entirely safe. If an exception occurs when we are performing some operation with the file, the code exits without closing the file. A safer way is to use a try...finally block.

```
try:
f= open("test.txt",encoding = 'utf-8')
# perform file operations
finally:
f.close()
```

This way, we are guaranteed that the file is properly closed even if an exception is raised, causing program flow to stop.



The best way to do this is using the with statement. This ensures that the file is closed when the block inside with is exited.

We don't need to explicitly call the close() method. It is done internally.

with open("test.txt",encoding = 'utf-8') as f:

perform file operations

Reading and writing

A text file is a sequence of characters stored on a permanent medium like a hard drive, flash memory, or CD-ROM.

To write a file, you have to open it with mode 'w' as a second parameter:

```
>>>     fout = open('output.txt', 'w')
>>>     print fout
<open file 'output.txt', mode 'w' at 0xb7eb2410>
```

If the file already exists, opening it in write mode clears out the old data and starts fresh, so be careful! If the file doesn't exist, a new one is created.

The write method puts data into the file.

```
>>>     line1 = "This here's the wattle,\n"
>>>fout.write(line1)
```

Again, the file object keeps track of where it is, so if you call write again, it adds the new data to the end.

```
>>>     line2 = "the emblem of our land.\n"
>>>     fout.write(line2)
```

When you are done writing, you have to close the file.

```
>>> fout.close()
```

Format operator

The argument of write has to be a string, so if we want to put other values in a file, we have to convert them to strings. The easiest way to do that is with str:

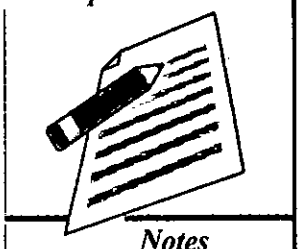
```
>>>     x = 52
>>>     fout.write(str(x))
```

An alternative is to use the **format operator**, %. When applied to integers, % is the modulus operator. But when the first operand is a string, % is the format operator.

The first operand is the **format string**, which contains one or more **format sequences**, which specify how the second operand is formatted. The result is a string.

For example, the format sequence '%d' means that the second operand should be formatted as an integer (d stands for "decimal"):

```
>>> camels = 42
>>> '%d' % camels
'42'
```



The result is the string '42', which is not to be confused with the integer value 42. A format sequence can appear anywhere in the string, so you can embed a value in a sentence:

```
>>>    camels = 42
>>>    'I have spotted %d camels.' % camels
'I have spotted 42 camels.'
```

If there is more than one format sequence in the string, the second argument has to be a tuple.

Each format sequence is matched with an element of the tuple, in order.

The following example uses '%d' to format an integer, '%g' to format a floating-point number and '%s' to format a string:

```
>>>    'In %d years I have spotted %g %s.' % (3, 0.1, 'camels')
'In 3 years I have spotted 0.1 camels.'
```

The number of elements in the tuple has to match the number of format sequences in the string. Also, the types of the elements have to match the format sequences:

```
>>> '%d %d %d' % (1, 2)
Type Error: not enough arguments for format string
>>> '%d' % 'dollars'
Type Error: illegal argument type for built-in operation
```

Filenames and paths

Files are organized into **directories** (also called "folders"). Every running program has a "current directory," which is the default directory for most operations. For example, when you open a file for reading, Python looks for it in the current directory.

The os module provides functions for working with files and directories ("os" stands for "operating system"). os.getcwd returns the name of the current directory:

```
>>>    import os
>>>    cwd = os.getcwd()
>>>    print cwd /home/dinsdale
```

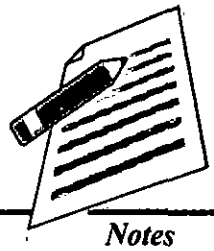
cwd stands for "current working directory." The result in this example is /home/dinsdale, which is the home directory of a user named dinsdale.

A string like cwd that identifies a file is called a **path**. A **relative path** starts from the current directory; an **absolute path** starts from the topmost directory in the file system.

The paths we have seen so far are simple filenames, so they are relative to the current directory. To find the absolute path to a file, you can use os.path.abspath:

```
>>>    os.path.abspath('memo.txt')
'/home/dinsdale/memo.txt'
```

os.path.exists checks whether a file or directory exists:




```
>>> os.path.exists('memo.txt')
True
```

If it exists, os.path.isdir checks whether it's a directory:

```
>>>     os.path.isdir('memo.txt')
False
>>>     os.path.isdir('music')
True
```

Similarly, os.path.isfile checks whether it's a file.

os.listdir returns a list of the files (and other directories) in the given directory:

```
>>>     os.listdir(cwd)
['music', 'photos', 'memo.txt']
```

To demonstrate these functions, the following example "walks" through a directory, prints the names of all the files, and calls itself recursively on all the directories.

```
def walk(dirname):
for name in os.listdir(dirname):
path = os.path.join(dirname, name)
if os.path.isfile(path):
print path
else:
walk(path)
```

os.path.join takes a directory and a file name and joins them into a complete path.

SUMMARY

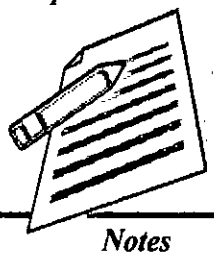
File is a collection of logically related records. You can write data on the file and read data on the file. A file can be opened in two ways: (i) using constructor function of a class, (ii) using the member function open() of the class. Open( ) requires two parameters: file name and access mode. File has two pointers called input pointer and output pointer. Tellg( ) function gives the position of get pointer in terms of number of bytes. Tellp( ) function gives the position of put pointer in terms of bytes.

EXERCISE

MCQ

1. Which one of the following is correct syntax for opening a file.
   a) FILE *fopen(const *filename, const char *mode)
   b) FILE *fopen(const *filename)
   c) FILE *open(const *filename, const char *mode)
   d) FILE open(const*filename)

   Answer: a



2. What is the function of the mode ' w+'?
   a) create text file for writing, discard previous contents if any
   b) create text file for update, discard previous contents if any
   c) create text file for writing, do not discard previous contents if any
   d) create text file for update, do not discard previous contents if any

   Answer: b
3. If the mode includes b after the initial letter, what does it indicates?
   a) text file
   b) big text file
   c) binary file
   d) blueprint text

   Answer: c
4. fflush(NULL) flushes all _____
   a) input streams
   b) output streams
   c) previous contents
   d) appended text

   Answer: b
5. _____removes the named file, so that a subsequent attempt to open it will fail.
   a) remove(const *filename)
   b) remove(filename)
   c) remove()
   d) fclose(filename)

   Answer: a

Review Questions

1. What is an input stream?
2. What is the difference between opening a file with constructor function and opening a file with open () function?
3. What is the file access mode? Describe the various file modes.
4. A file consists of 5 records, each takes 100 bytes of storage: fstream file: file.seekg (0, ios::end); N = file.tellg ( ); (i) What will be the datatype of N? (ii) What will be the value of N?
5. Consider the following statements: fstream file; file.open ("ABC", ios::in | ios::out); Write C++ statement(s) for the following: (i) To move the pointer at the beginning of file. (ii) To move the pointer at the end of file. (iii) To find the total number of bytes. (iv) To close the file.
6. Explain the functioning of the following: fstream file; (i) file.seekg (100, ios::cur); (ii) file.seekg (-100, ios::end); (iii) file.seekg (100, ios::beg);

BOARD OF OPEN SCHOOLING AND SKILL EDUCATION
Near Indira Bypass, NH-10, Gangtok, East Sikkim- 737102
Telephone : 03592-295335, 94066 46682 Email : bosse.org.in