COMPUTER CONCEPTS

CONTENTS:
INTRODUCTION TO DIGITAL COMPUTER

Basic functional units of a digital computer:

Central Processing Unit
Arithmetic and Logic Unit
Control Unit
Memory Unit
Input and Output Units
Stored program concept

Hardware:

Input Devices:
Keyboard
Mouse and Scanner
Output Devices:
The meaning of Hard copy and Soft Copy
Printers:

Dot Matrix printer
Laser printer and Ink jet printer
Plotters
Storage Devices:

Primary Storage
Random Access Memory
Read Only Memory
Secondary Storage:
Floppy Disk
Hard Disk
CD ROM and its operation
Components of a Personal Computer

Software:

Computer Languages:
Machine language
Assembly language
Higher-level languages and Compiler
Interpreter
Editor
System Software
Application Software

Operating System and its functions
Specific features of DOS and UNIX Operating System
Some preliminary commands of DOS and UNIX operating systems

Computing Environments:

Networking of computers and its advantages
LAN
WAN
Internet
E-mail

Chapter 1

INTRODUCTION TO DIGITAL COMPUTER

1.0 Introduction

We know information processing, plays very important role in taking decision every moment. In this context, computers play a significant role in bulk of information processing. Here, we study what is a computer and organization of a computer. The computer operates on a program or set of instructions. We discuss the important contribution made by the John Von Neumann. The objective is to understand the definition of computer, working concepts of the computer, stored program concept and Microprocessor.

1.1 BASIC FUNCTIONAL UNITS OF A DIGITAL COMPUTER

A computer is an electronic device which accepts information and processes the information according to the program and produces the output. Computer programs may be written in High level languages like Pascal, Fortran, Cobol and so on. Some programmer also writes assembly language to carry out the desired task.

A computer system consists of hardware and software. A hardware refers to any physical, electrical, electromechnaical components of the computer. For example keyboard, mouse, cabinet of computer is considered as hardware. A software refers to a program or set of instructions that is written to achieve a specified task.

A computer system has five basic functional units which are listed below

a) Input Unit
b) Output Unit
c) Control Unit
d) Memory Unit
e) Arithmetic Logic Unit

} central processing unit

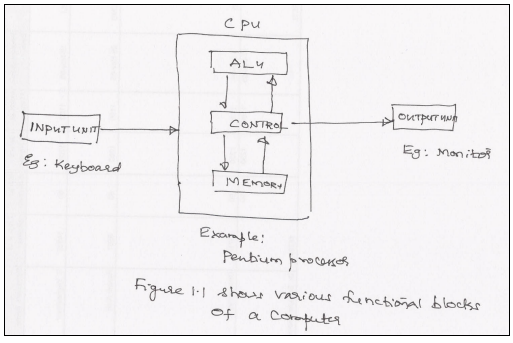


Figure 1.1 shows various functional blocks of a Computer

The figure 1.1 shows the computer organization depicting the basic units of a computer.

Input unit:

The input device is used to enter data and information into a computer. The devices like keyboard, mouse and scanner are commonly used as input devices. A keyboard is used to enter alphanumeric characters and symbols. The mouse is used to pick or select a command from the monitor screen. A scanner is used to scan an image or read a barcode and so on.

Central Processing Unit:

The processing unit comprises a processor which interprets the program instructions in memory, controls the flow of data and performs arithmetic and logical operations. The program instructions are processed one at a time along with the necessary data. The results are sent to memory and the next instruction is processed. This method is repeated until the program is executed.

Arithmetic and Logic unit:

The arithmetic-logic unit (ALU) is the unit of the computer that performs arithmetic and logical operations on the data. This section of the machine can be relatively small consisting of circuits and registers which perform arithmetic (+, -, *, /) and logic (>,<,<=,>=,etc) operations. Arithmetic-logic units which can add and subtract and perform logical operations form the backbone for the arithmetic and control operations in computers. To perform scientific calculations the floating-point number system is used.

Control unit:

The control unit controls the overall activities of the components of the computer. It is mainly used to coordinate the activities among other units. It will send commands signals and controls the sequence of instructions to be executed. The control unit may be defined as “the parts that effect the retrieval of instructions in proper sequence and application of the proper signals to the arithmetic unit and the other parts”.

The function of the control circuitry in a general purpose computer is to interpret the instruction words and then sequence the necessary signals to those sections of the computer that will cause it to perform the instructions.

Memory Unit:

The memory unit is the unit where all the input data and results stored. The CPU memory is also called as memory register. The memory of a computer is also available in the form of Random Access Memory (RAM). RAM is a semiconductor chip. RAM is considered as a volatile memory, it means as long power is supporting information stored in it remain. Once the power is lost, the information stored in the RAM also get erased. Microcomputers contains read Only Memory (ROM). ROM contains instructions for the microcomputers. Microcomputers use ROM, programmable read only memory (PROM), and erasable programmable read-only memory (EPROM) to store selected application programs. The contents of ROM are determined when the chips are manufactured. The ROM memory is considered as non volatile, means the information is not get erased even when power is failed. The most important ROM chip(s) we should know about is the Basic Input/output system or BIOS. The BIOS is a collection of small computer programs built into a ROM chip.

On personal computer there are three types of memory. They are

1) Conventional memory: The memory into which we load our software and work files. Conventional memory also known as base or low memory is any memory below 1M (1024) although only 640k of it is directly available for our work.

2) Extended memory (XMS): Memory above 1M. This type of memory is usually not directly available to our software.

3) Expanded memory (EMS): To expand the memory by reserving a special peephole of 64kb of memory to be used when the computer requests certain data not immediately available from RAM. Usually a software utility called an Expanded Memory Manager (EMM) manages this expanded memory.

Output Unit:

The output device is used to display or print result from a computer. Monitor, printer and plotter are commonly used output devices. A monitor is used to display the result in the form of text and graphics. The printer is used to print the result. A plotter is used to plot or print graphical result from a computer. Note that a result displayed in a monitor is temporary and it disappears when the next result is displayed, whereas the output printed using a printer or a plotter is permanent and these printouts can be used for any business correspondence or documentation. Normally soft copy is referred to information that is stored on the storage device. A hard copy refers to a print out showing the information.

1.2 Stored program concept.

Most computers use the stored-program concept designed by Hungarian mathematician John Von Neumann. In John Von Neumann architecture, a computing machine that uses a single storage structure to hold both the set of instructions on how to perform the computation and the data required or generated by the computation. Such machines are also known as stored program computer. The separation of storage from the processing unit is implicit in this model. The storage of instructions in computer memory to enable it to perform a variety of tasks in sequence.

Stored program concept has the following features

a) Random access memory which stores information and is accessible independently of its content.
b) A central processing unit that accesses the RAM using a fetch-decode-execute cycle.
c) Input/output devices.

The time taken to access the memory is constant over all addresses; each address stores the same amount of information.

1.3 Microprocessor

A microprocessor is a semiconductor chip, which is manufactured using the Large Scale integration (LSI) or Very Large Scale Integration (VLSI), which comprises Arithmetic Logic Unit, Control unit and Central Processing Unit (CPU) fabricated on a single chip. Intel 8088, Motorola 68000 are few microprocessors.

1.4 Summary:

- A computer is an electronic device which takes information and process information according to the program and produces the output.
- A computer system has five basic functional units.
- The Central Processing Unit is the brain of the computer.
- The arithmetic-logic unit (ALU) is the unit of the computer that performs arithmetic and logical operations on the data.
- The control unit controls the overall activities of the components of the computer.
- The memory unit is the unit where all the input data and results stored.
- Stored program concept uses the memory unit to store both instruction or operation code and data or operands.

1.5 QUIZ QUESTIONS

1) A Computer consists of --- units.
   a) 3 b) 4 c) 5 d) 6
2) Keyboard is an example of ------ unit.
   a) Memory b) Input c) Output d) ALU
3) ALU stands for -------
   a) Arithmetic Logic Unit b) Arithmetic Lower Unit c) Add Logical Unit
   d) None of the above
4) RAM is considered as a ----------
   a) Volatile Memory b) Non volatile Memory c) Permanent
   d) None of the above
5) ------ contains the a program during the manufacturing itself.
   a) RAM b) ROM c) Both a and b d) None of the above

6) ------- unit is used to store information.
a) Input b) Output c) Control d) Memory

7) In Stored program concept ---- and --- are stored in the same memory.
a) Data and Instruction b) Data and Operands c) Instruction and operation code d) None of the above

8) Microprocessor is the heart of ------ computer.

a) Digital b) Analog c) Both a and b d) None of the above
Answers: 1 c

2 b
3 a
4 a
5 b
6 d
7 a
8 a

1.6 Exercises

1. Mention the basic functional units of a computer?
2. With a neat diagram explain the working organization of a computer?
3. What is Stored program concept or John Von Neumann concept?
4. What is microprocessor?
5. What are the differences between RAM and ROM?

Chapter 2

Hardware

2.0 Introduction

We know hardware refers to physical, electrical, mechanical and electromechanical components of a computer. The Input unit is one, through which computer receives the information and send the processed output onto the output device. In this chapter we discuss the different types on input devices and output devices and their working procedure.

2.1 Input devices

2.1.1 Keyboard:

A keyboard is an input device used to enter data into a computer. The keyboard contains function keys, numeric keys and toggle keys (caps lock, num lock, scroll lock) and so on. Some keyboard supports 100 and some support 104 keys.
A keyboard is used to enter data into a computer. The latest keyboard (Windows keyboard) is available with 104 keys. The keyboard contains function keys, numeric keys and toggle key (Caps lock, Num lock, Scroll lock) and so on.

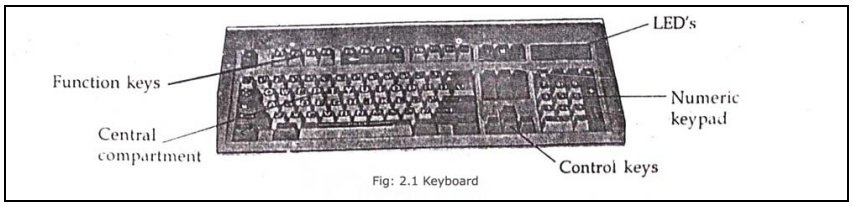


Figure 2.1 shows the diagram of a keyboard with various parts.

It is the most widely used input device. it has keys similar to a typewriter to enter characters and other symbols. The function keys are used to activate a particular feature of software like invoking the help system, selecting a menu and so on.

There is a separate numeric keypad to enter numeric keypad to enter numeric data. When a key is pressed the electric circuitry under the key will change which will be detected by the microprocessor and the binary code for the character is sent to the CPU. Some keyboards have a temporary memory or buffer to store the information typed a little ahead of the need of the computer.

2.1.2 Mouse:

A mouse is an input device used to select a command by moving it in any direction on a flat surface. It has two or three buttons to confirm the selection.

Figure 2.2 shows the picture of a mouse.

The software developed with graphical interface requires the mouse. The cursor is moved to the required icon or menu on the monitor and a button is pressed. The control is sent to the CPU to select the command corresponding to the icon or menu item. The latest is the wireless mouse or remote mouse which works on transmission of infrared or radio waves are also available.

The mouse can also be used to open menus, select text for editing, move objects on the screen and draw images or diagrams. The mechanical mouse uses a rubber-coated ball on the underside. The movement of this ball sends electrical signals to the system unit which cause the cursor or pointer to move in a corresponding fashion. An optical mouse uses diodes to emit light onto a metal pad performing the same work but with great accuracy.

2.1.3 Scanners:

A peripheral input device used to assist in the entry of data into a computer system. In desktop publishing, a scanner may be used to digitize artwork or photographs so that they can be merged with text. Scanners are used to scan a printed page or an illustration. These data are then converted into bit patterns for processing storage or output. When an image is scanned, it is converted into light and dark picture elements or pixels. The scanned images are used for word processing and printing multiple copies. Scanners are also useful to scan fingerprints. The scanned fingerprints can be compared with another fingerprint to find probable match in investigative services. Photoelectric scanners are commonly used in supermarkets to read barcodes.

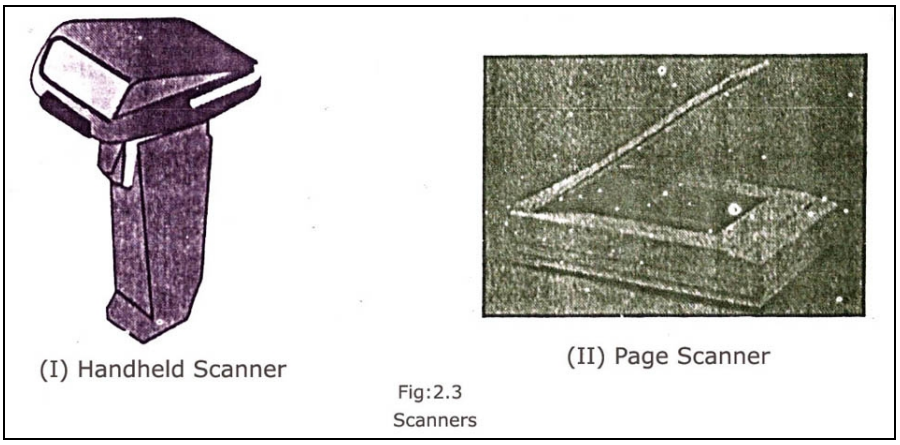


Fig:2.3
Scanners

Figure 2.3 shows the diagram of a scanner.

Scanners are available in different sizes. A handheld scanner is used to scan a few lines of text or a small photograph. A page scanner is used to scan a drawing or page.

The scanner is connected to the computer using a cable and controlled by software.

2.2 Output devices:

Hard copy:

The data consisting of text or graphics that is obtained as printouts or microfilm using printers or plotters is known as hardcopy.

For example, the hardcopy of an engineering drawing is obtained using plotters. Some hardcopy devices include dot matrix printer, laser printer, inkjet printer , flatbed pen plotter and drum type inkjet plotter.

A combination of printing, scanning, copying and/or faxing can also obtain a hardcopy. A hardcopy can be used for business correspondence and documentation. A copier machine also comes under hardcopy devices.

Soft copy:

The data that is stored in a storage device such as floppy disk, hard disk, CD-ROM magnetic tape and so on is called softcopy.

The data in a softcopy may be modified using the relevant software. A few softcopy devices are monitor and floppy disk.

2.2.1 Printers:

A printer is an output device used to print text or graphics on paper or on any other hardcopy medium which includes even microfilm. A permanent copy from the computer is produced using the printer. Printers are of two basic types impact and non-impact.

Impact and Non-impact printers:

Printers are categorized based on the physical contact of the print head with the paper to produce a text or an image. An impact printer is one where the print head will be in physical contact with the paper. In a non–impact printer, on the other hand the print head will have no physical contact with the paper. The Dot matrix printer is considered as a Impact printer and Laser printer is considered as Non-impact printer.

The basic operations performed by a printer are:

1) Moving the paper to a given line.
2) Moving the print head along the line.
3) Generating the character or image.
4) Producing the character or image on the paper.

2.2.2 Dot Matrix Printer:

The most popular kind of printer for small computers is the dot matrix printer, which forms characters as arrays of dots. Dot matrix printers are compact, reliable and relatively fast. This type of printer is an impact printer. The print head is the important hardware which produces the character using pins arranged in a matrix form. Normally a print head has 9 pins or 24 pins arranged in a matrix form. Combinations of pins strike an ink bed ribbon during the printing process. The print head moves in a line and the pattern of dots required for each character is printed on the paper. After printing a line, the paper rolls to print the next line.

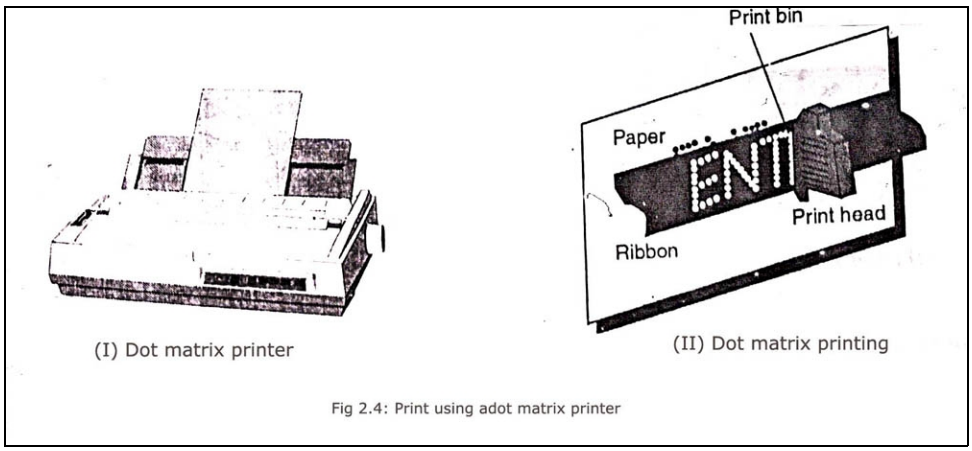


Figure 2.4 shows the diagram of a dot matrix printer.

Dot matrix printers produce average quality prints and as generally used in business applications. They are used printing train number, seat number etc on a railway reservation ticket. The speed of printing in dot matrix printer in dot matrix is measured in characters per second (cps).

The advantages in this type of printer is carbon copies can be obtained as printing takes place by physical impact with the paper. It is less of cost and easy to maintain.

The disadvantage is average printing quality and printer ribbon printer ribbon needs to be changed frequently.

2.2.3 Laser Printer:

Laser printers are fastest type of non impact electrostatic printers. They produce high quality prints at high speeds. It operates like a copier machine. In these printers, the controlled beam of intense laser forms images on an electrically charged rotating drum. The drum is rotated near the fine black powder called the toner. These charged images which sticks to the paper due to pressure and heat. The toner consists of oppositely charged ink particles which stick to the drum in the places where the laser has charged.

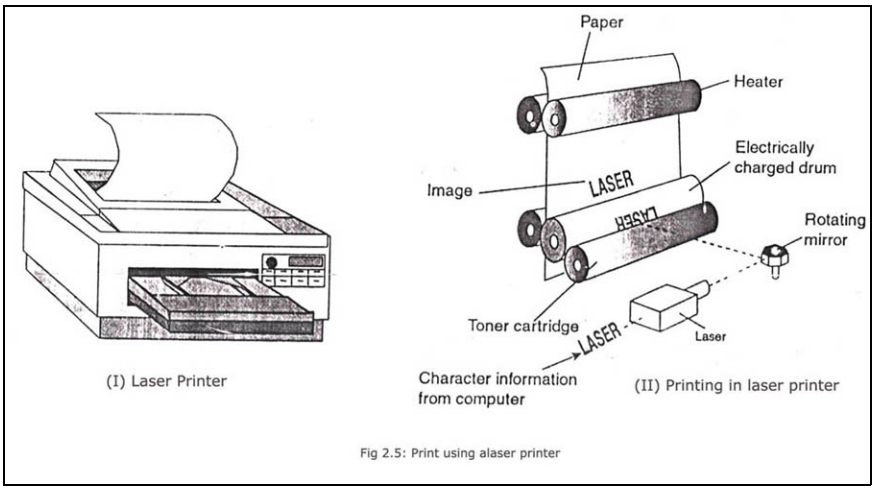


Figure 2.5 shows the diagram of a laser printer.

The light beam strikes a multi-sided rotating mirror. As the mirror rotates, the side currently in the path of the light beam sweeps the beam across the surface of the drum. As the beam sweeps across the drum, the light is modulated and a single line is drawn after a line has been drawn, the next side of the mirror is in place and a new line is drawn below the previous line.

The quality of the printout is measured by the number of dots per inch (dpi). Since the dots are printed closely, the text or graphics appears very smooth and elegant. The speed is measured in number of pages printed per minute (PPM) which varies between 5 and 25.

The advantages are good quality printouts can be obtained for documentation and business applications. The printing is faster and easy to handle and maintain. The disadvantages are the price is high and higher print cost.

2.2.4 Inkjet printer:

An inkjet is a non impact printer. It sprays tiny drops of ink to form character and graphic images on paper. The text and graphics printed in an inkjet printer are technically similar to that of a dot matrix printer. These type of printers can also be used color printing. The black inkjet printer uses black cartridge filled with black ink whereas the color printer uses four color cartridges namely cyan (blue), magenta (red), yellow and black. These four colors are used in combination to generate any color in the visible spectrum.

Figure 2.8 shows the diagram of an Inkjet printer (pl.insert any of the inkjet printer pic)

The print heads move across the page by the control of software and spray the dots of ink with the required combination of colors. The printer sends electrical pulses to thin resistors at the base of firing chambers behind the nozzle. A thin layer of ink is heated by the resistor which in turn forms a vapour bubble and the expansion forces ink through the nozzle and onto the paper at a rate of about 6000 dots per second. The quality of the printout is equivalent to that of laser printouts. The speed of printing is slower than that of laser printers.

The advantages of this type printer are the cost is low; quality of printing is equivalent to that of laser printing, color printouts are cheaper easy to handle and maintain.

The disadvantages are the ink cartridges may get spoiled if unused for a long time. Some inkjet printers are very expensive.

2.2.5 Plotters:

A plotter is an output device used to print engineering drawing or graphics on large size sheets. These are also used to draw the patterns from which microprocessors, memory chips, and other integrated circuits are manufactured. Plotters are used when highest quality and greatest accuracy are required.

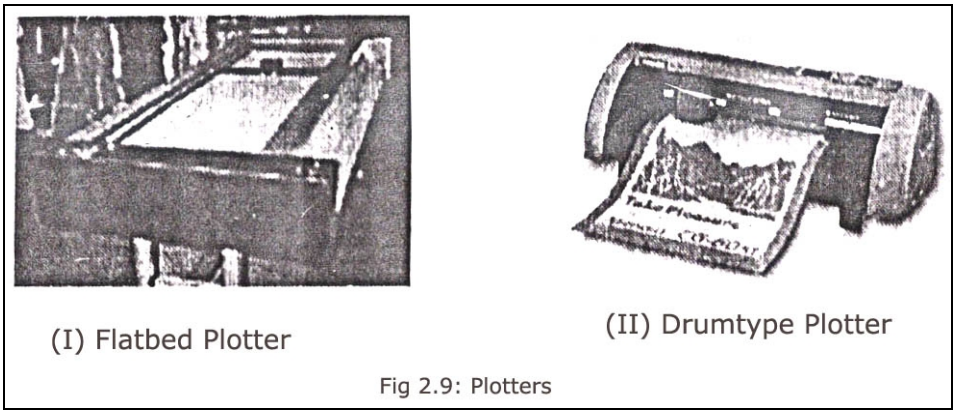


Fig 2.9: Plotters

Figure 2.9 shows the diagram of a Plotter.

There are two basic types of plotters: flat bed plotters and drum type plotters. Pen plotter is an example of a flatbed plotter. Laser plotter and inkjet plotter are commonly used drum type plotters.

A pen plotter has a surface where the paper or drawing sheet is properly fixed. It has a pen holder in a movable arm. Under the control of the computer the arm with the pen moves across the paper to draw the picture. A few pens are also placed in a row and the arm will pick the required color pen as per the instruction of the computer.

A drum type plotter uses a drum where the paper will be rolled. It has a print head/pen that moves like the print head in a printer. Drum type plotters are capable of producing longer, continous drawings. Drum type plotters that can produce color plots are available.

The disadvantages are these are expensive than printers. The cost of printing is high. Cost of maintenance is high. High skill of operation is required. Process time of printing is longer.

2.3 Summary

- A keyboard is an input device used to enter data into a computer.
- The keyboard contains function keys, numeric keys and toggle keys (caps lock, num lock, scroll lock) and so on.

- A mouse is an input device used to select a command by moving it in any direction on a flat surface
- The software developed with graphical interface requires the mouse.
- A peripheral input device used to assist in the entry of data into a computer system.
- A printer is an output device used to print text or graphics on paper or on any other hardcopy medium such as microfilm.
- Printers are of two basic types impact and non-impact.
- The most popular kind of printer for small computers is the dot matrix printer, which forms characters as arrays of dots.
- Laser printers are fastest type of non impact electrostatic printers. They produce high quality prints at high speeds.
- An inkjet is a non impact printer. It sprays tiny drops of ink to form character and graphic images on paper.
- A plotter is an output device used to print engineering drawing or graphics on large size sheets.

2.4 QUIZ QUESTIONS

1. ---- and ---- are examples of input device.

2. Printout of a program is considered as ------.

3. Payroll program stored on CD is considered as ----.

4. Dot matrix printer is an example of --------.

5. Laser printer is an example of -----.

6. The speed of Dot matrix printer is expressed as ----

7. The speed of Laser printer is expressed as ----

Answers:

1. Keyboard, Mouse
2. Hardcopy
3. Softcopy
4. Impact printer
5. Non-impact
6. Characters per second (CPS)
7. Pages per Minute (PPM)

2.5 EXERCISE

1. What is the difference between impact and non impact printer?
2. What is the difference between Hard and soft copy?
3. List and explain any two input and output devices.
4. With a neat diagram explain the working of a mouse.
5. What is a plotter? How plotter works?

Chapter 3

Storage devices

3.0 Introduction

We discussed in chapter 1, memory is used for storage purpose. Microprocessor fetches instruction or the operation code from the memory. Once the operation code is decoded, it fetches operand that is followed in the memory. There are two types of memory, one is called volatile memory and other is non-volatile memory. In volatile memory, information is retained as long as power is supplied to the chips. In nonvolatile memory, information is retained, even though power is not supplied. Random access memory belongs to volatile memory and hard disk belongs to the nonvolatile memory. In this chapter we discuss, various storage devices in detail.

3.1 Primary Storage: RAM (Random Access Memory)

Random Access Memory is a temporary storage medium in a computer. All data to be processed by the computer are transferred from a storage device or keyboard to RAM during data processing. Results obtained from executing any program are also stored in RAM. RAM is a volatile memory. Latest computers use RAM with a memory of more than 128MB. There are provisions also available to increase the RAM memory in any computer.

RAM consists of many storage cells each of size 1 byte and is identified by using a number called as address or memory location. The memory address is assigned by the computer which also varies from computer to computer and time to time. The data stored in memory are identified using the memory address.

The internal processing speed of a computer is very fast compared to the reading and writing from/to disk. During the time of reading from disk the CPU is idle. To reduce this waiting time and increase the processing speed, a cache memory is used in the computer Cache memory is a part of RAM that holds the data, which is needed next by the CPU. The size of cache memory is 512 KB. Normally cache memory holds the recent information that is accessed. The data retrieval time for the processor from cache is more than the thousand times faster from disk., so the processing speed of a computer is improved.

ROM (Read Only Memory)

Read Only Memory is a permanent storage medium which stores start up programs. These programs which are loaded when computer is switched on. ROM stores essentially the BIOS (Basic Input Output System) programs which are recorded by the manufacturer of the computer system. ROM is non-volatile memory.

ROM is also known as firmware. In ROM programs are burnt during manufacturing. Normally system programs and language translators are stored in ROM chips.

Both ROM and RAM are semiconductor chips. Normally size of the ROM holds 8k and more depending on the requirement.

3.2 Secondary storage

Floppy disk:

A floppy disk is used to store data permanently. It has a flexible disk coated with magnetic material and is enclosed in a plastic cover. Floppy disks of 3 ½ inch diameter have a storage capacity of 1.44MB. The FDD (floppy disk drive) has a read/write head which reads/writes data on to the disk. The disk rotates at 360rpm while reading or writing on to it. Figure 3.1 shows the floppy disk.

Data are stored in a floppy disk in concentric circles known as tracks. Tracks are divided into many storage locations called sectors. Tracks and sectors on a disk are identified by the disk drive through formatting. Formatting is a process by which the operating system program controls the disk drives by removing the old data and sets up each track and sector. The root directory will be created in the disk during formatting and the users create other directories. Information is stored in the form of files. A file allocation table (FAT) is used by the operating system to identify the files stored in the disk.

Figure 3.1 shows floppy disk

Hard disk:

Hard disk is a reliable and permanent storage disk. It has a set of metal disks coated with magnetic material and are mounted on a central spindle which rotates at 7200 rpm. The HDD has a set of read/write heads which are mounted on an arm. Latest hard disks are available with a storage capacity of more than 40GB. Figure 3.2 shows a hard disk.

A hard disk has a collection of several (say 6 or 7) hard disk platters staked one above another to have a high storage capacity.

A collection of tracks across all the disks is called cylinder. In order to read a specified file, the access mechanism with the head moves to the specified cylinder.

The advantages of a hard disk are high storage capacity, high speed of operation and reliable media mainly in personal computers. External disturbances to the drive may lead to the damage of disk surface or read/write head. This is a major disadvantage in a hard disk.

Figure 3.2 shows a hard disk

3.3 CD ROM:

Compact Disk (CD) is an optical disk used to store data permanently. It is the most reliable storage media available today. Data stored on a compact disk cannot be erased. The CD drives commonly available are read only. Read/Write CD drives are also available but are expensive. Storage capacity of CD is 700MB. Figure 3.3 shows a CD-ROM.

In the optical disk, a high power laser beam is used to record data by burning tiny pits in the surface of the hard plastic disk. To read the data, a low power laser beam is used to scan the disk surface. When the laser beam reflects from the smooth surface of the disk it is interpreted as a 1 bit and when the beam enters the tiny pits it is interpreted as a 0 bit.

The major advantages of the optical disk are high storage capacity and high quality recording of graphical images and sound. These are used commonly nowadays. CAD and structural drawings used by architects and engineers are generally stored in compact disks. It is also used in high quality music recording, multimedia and educational applications.

Figure 3.3 shows CD-ROM

3.4 Components of a personal computer:

A personal computer commonly available today comprises a processor (like Pentium IV), a keyboard, a mouse, a floppy disk drive(FDD), a hard disk drive(HDD), a compact disk drive (CDD) a color monitor, RAM(Random Access Memory) and ROM(Read Only Memory). The microprocessor/CPU, RAM, ROM and other supporting circuitry are interconnected on a single board called mother board.

3.5 Summary:

- Random Access Memory is a temporary storage medium in a computer. All data to be processed by the computer are transferred from a storage device or keyboard to RAM
- Read Only Memory is a permanent storage medium which stores start up programs
- A floppy disk is used to store data permanently.
- Hard disk is a reliable and permanent storage disk. It has a set of metal disks coated with magnetic material and are mounted on a central spindle which rotates at 7200rpm
- Compact Disk(CD) is an optical disk used to store data permanently.

3.6 Quiz questions

1. RAM is a --------memory.
2. Floppy disk storage capacity is -----.
3. Concentric circles in a floppy disk is known as -----
4. ROM is also known as -----.
5. ROM is a ----- memory.
6. Hard disk is ---- than primary memory.
7. CD-ROM storage capacity is ---- .
8. Information stored in a cd is accessed through ----

Answers
1. Volatile
2. 1.44 MB
3. tracks
4. firmware
5. Nonvolatile
6. Bigger
7. 700 MB
8. light

3.7 Exercises

1. Mention the storage devices used in a personal computer?

2. Explain primary memory its properties and its types?

3. What is the need for secondary storage? Briefly describe secondary storage devices like (i) Floppy disk (ii) hard disk (iii) CD ROM

4. Mention the components of a personal computer?

5. What is the difference between volatile memory and non-volatile memory?

Chapter 4

Software

4.0 Introduction

We know software is a set of instructions that are used to carry out a task. Software can be grouped into two categories namely application software and system software. The application software is one, which is application oriented, like our inventory program, payroll program are few. Similarly system software is used for system oriented tasks. Examples are compilers, assemblers, loaders. In this chapter, we discuss the computer language fundamentals, application software and system software.

The objective of this chapter is to understand

Concept of machine, assembly and high level language.
Role of compilers, assembler and interpreters.
Difference between editor and word processor.
Distinguish between application software and system software.
Operating system functions.
Features of DOS and UNIX.

4.1 Computer languages:

4.1.1 Machine language:

At the lowest level computer understands only 0 and 1. Programs expressed in terms of binary language are called machine language. A computer's programming language consists of strings of binary numbers (0's and 1's) and is the only one language computer can understand. This language is the lowest level of computer language recognized and used by the CPU. An instruction prepared in any machine language consists of 2 parts. The first part is the command or opcode or operation code. The second part of the instruction is the operand/s or data and it tells the processor where to find or store the data or other instructions that are manipulated. A short sample of machine language to perform addition in the storage location 0166 will look like this

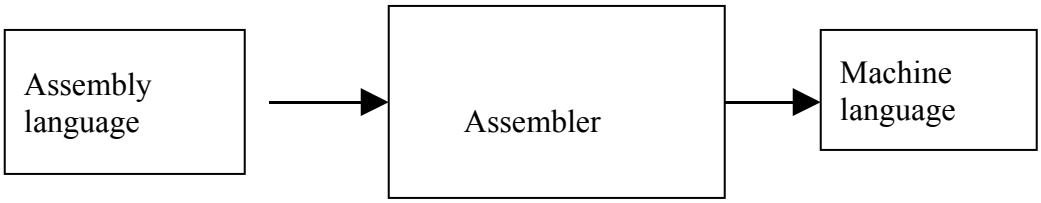
00010000 00000001 01100010

A machine language programmer has to know the binary code for each operation to be carried out. Machine language programmers must also be familiar with the internal organization of the computer. A machine language programmer must also keep track of all the addresser of main memory locations that are referred to in the program. The machine language format is slow and tedious. We the human beings work on natural language and not on binary language. Hence writing machine language program is difficult for the humans.

4.1.2 Assembly language and Assembler:

A low level first generation computer language, popular during early 1960s, which uses abbreviations or mnemonic codes (mnemonic means mind full) for operation codes and symbolic addresses. This symbolic instruction language is called Assembly language. One of the first step in improving the program preparation was to substitute mnemonics for operation codes. The mnemonics are different among makes and models of computer. Second step was symbolic addressing to express an address in terms of symbols convenient to the programmer. Another improvement was the programmer turned the work of assigning and keeping track of instruction addresses over to the computer. The programmer merely told the machine the storage address number of the first program instruction and the assembly language software the automatically stored all others in the sequence from that point.

The mnemonics are converted into binaries with the help of a translator known as Assembler.



The program written using mnemonics is called Source program or assembly language program , the binary form of the source program equivalent is called Object Program.

Let us consider an assembly language program

```
LDA 9000
MOV B,A
MOV C,A
HLT
```

Assembler is used to convert assembly language into the machine language.
For example object program or machine language equivalent for the above assembly language is

```
01110 0100001100
01110011101
 0111110000
 1011100000
```

Assembly language programs are commonly used to write programs for electronic controls using microprocessors e.g., compilers , operating systems, animation in computer graphics and so on.

Assembly language is relatively easy for the human beings compared to machine language. Programs writing are faster compared to machine language.

Assembly language programmer should know details of the architecture of the machine. Assembly language programs are not portable.

4.1.3 Higher level languages and compiler:

Instructions which are written using English language with symbols and digits are called high level languages. The high level language is closer to our natural language. The commonly used high level languages are FORTRAN, BASIC, COBOL, PASCAL, PROLOG, C, C++ etc. The complete instruction set written in one of these languages is called a high level language program or computer program or source program.

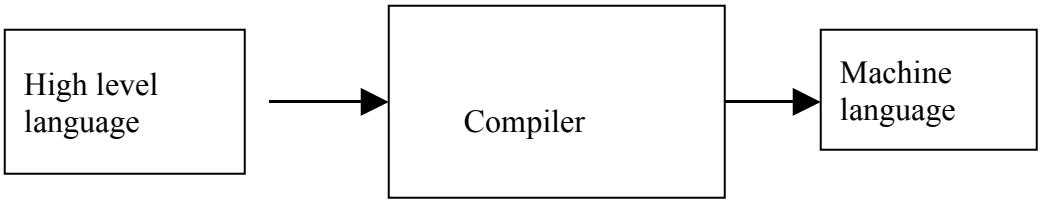
In order to execute the instructions, the source program is translated into binary form by a compiler or interpreter. A compiler is also used to translate source program into an object program. An interpreter is a program which takes the source program line by line and converts into machine code line by line. .

C language uses a compiler as its translator to translate or compile the complete C program. It is also necessary to create an executable program to execute the instructions given in a source program by linking the input and output devices with your program. A linker (another program) is used to link library routing and generate an executable program from an object program. Compiler converts source program into object program in terms of stages called passes. Normally, most of the compilers uses two passes to convert source program into the machine language program.

Gwbasic is an interpreter used to convert basic program into object program.

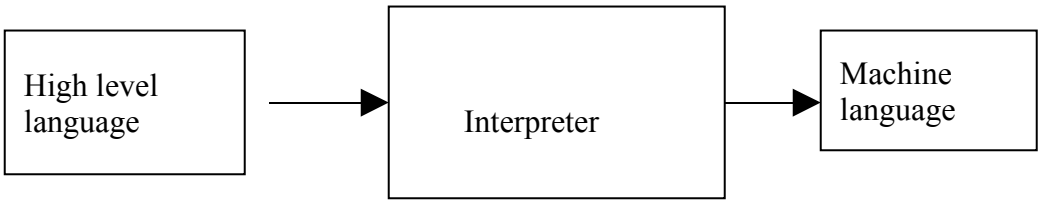
4.2 Compiler:

Compilers convert the program instructions from human understandable form to the machine understandable form and the translated program instruction is called object code. Compiler is nothing but a language translator used to translate the entire program of the high level language into machine language. Every programming language requires its own compiler to translate the program. For example, the programming language PASCAL requires PASCAL compiler and C uses C compiler.



4.3 Interpreter:

Interpreters also convert the source program to machine language instruction but executes each line as it is entered. The translation of the source program takes place for every run and is slower than the compiled code. An interpreter must accompany the object code to run a program. Programming languages BASIC and LISP use interpreters.



4.4 Editor:

An editor is used to type the source program and store program in disk. C language uses one popular Boroland's a IDE (Integrated Development Environment) editor in MS-DOS system and in Vi editor in UNIX system. In dos, we use popular Edit editor also.

Editors are commonly used to type and edit documents and store them. Thus, they are also called text editors. In word processors we can perform the operation like setting up margins, spell check and so on. MS-WORD is one of the popular word processor.

4.5 System software:

System software is designed for a specific type of hardware. For example, the disk operating system (DOS) is used to co-ordinate the peripherals of a computer. The system software controls the activities of a computer, application programs, flow of data in and out of memory and disk storage. Our operating system, compilers, assemblers, linker and loaders are the example of system software.

System software also handles data in communication applications and within the computer systems in a computer network. The communication software transfers data from one computer to another. These programs also provide data security and error checking along with the transfer of data between the computer systems.

4.6 Application software:

Application software are developed for application of the computer to common problems and tasks. They are available for business applications, science and engineering applications and so on. Personal productivity programs are categorized based on the nature of their use in word processing, generating spreadsheet, presenting graphics and maintaining databases. Application software is also available as packages and usually with a user manual.

Some of the application software are:

a) **Word processors**:

A word processor is used to prepare a report, a personal or business letter, in desktop publishing and so on. These offer formatting features such as using different character styles, line spacing, and page numbering and so on. Documents prepared using a word processor can be easily printed in any type of printer.

b) Electronic spread sheets:

An electronic spreadsheet software is used to prepare documents containing information or data in the form of numbers or characters. The information is arranged in rows and columns for further processing and analysis, preparing reports and generating charts. It is also capable of performing arithmetic operations and using functions.

c) Database software :

Databases are records related to a person or an organization. Database software have capability to edit and update data in a file. The data are processed to prepare and print salary details of employees, annual sales details and so on. One of the major applications of a computer is database management.

4.7 Operating systems and its functions:

We know operating system is a collection of programs and it is the interface between user and the computer. An operating system is a program which connects the user and the electronic hardware in a computer. It is a set of programs which supervise the activities of a computer and activate the operations of the hardware components such as CPU, main memory, disk drives, keyboard, monitor and printer and so on. Some of the startup programs initially loaded to RAM are stored in ROM, mainly the BIOS programs which are recorded by the manufacturers of the computer system. Service programs available in operating system for operating system for operations like copying a file, deleting a file, formatting a disk, printing a file and so on are usually stored in the disk. Error messages are displayed on the screen if there is any malfunctioning of hardware.

There are many operating system used in computers. Commonly used operating systems are MS-DOS (Microsoft Disk operating System), Windows 95/98/2000, Windows NT, UNIX and so on. Nowadays Windows 2000 operating system is widely used

in personal computers, and UNIX is used in Mainframes, Servers, Graphic Workstations and also personal computers. Linux is one of the most popular free operating system.

- Operating system will display instruction on the monitor screen and the user can interact with the computer.
- It loads the application programs such as MS Word ,AutoCAD and so on from disk to the computer memory.
- It manages the information stored on disk and retrieves the same whenever required.
- It supervises and coordinates the activities of the hardware and peripherals such as CPU, keyboard, mouse, monitor, printer, RAM, disk drives and so on.
- It utilizes the power of the CPU for multitasking and timesharing.

In general operating systems performs many task which include
Memory management
Process management
I/o management
Device management

4.8 Multitasking:

It is the ability of the computer to handle several application programs concurrently. Printing a document, executing a program and any other operation can be done simultaneously to reduce the idle time of the processor. The multi task capability of the operating system will utilize the processor efficiently, the reducing the user time. Another simple example is hearing audio songs and typing programs same time.

4.9 Timesharing:

It is the ability of the CPU to serve many users connected to it through a network. The operating system will assign each user a slice of processor time or time quantum in a round-robin fashion. Since the CPU has high processing speed, it can process information of many users.

4.10 Specific features of DOS and UNIX:

MS-DOS is a single user operating system developed by Microsoft Corporation. An operating system has a collection of program. When the computer is switched on, the file COMMAND.COM is loaded to the RAM and after the successful start of the computer, the DOS prompt or command prompt will be displayed. The DOS prompt displays the letter associated with the disk drive followed by a > symbol. For floppy disk drive , A> or A:> is displayed and for hard disk drive C> or C:\> is displayed. It indicates the operating system is ready to take commands from the user. MS-DOS is one of the popular operating system for desktop computers. DOS operating system consists of three parts in it, namely resident part, initialization and the transient part. Most of the command programs are located in the resident part. While booting, the number of files and buffers to opened are contained in the initialization part and

transient part is flexible part of the operating system. The commands are not case sensitive.

File:

A file is a collection of related information. For example , like the contents of a file folder in a desk drawer. Files on the disk can contain letters, memos and executable programs.

Program:

Programs are special types of files. These are series of instructions written in computer languages. These programs instructs the computer to perform the task.

Directory:

DOS uses a filing system to store its files. The filing system uses storage areas called directories. A directory is nothing more than an expandable file folder that can hold other expandable file folders. These file folders hold the data files. A directory is a table of contents for a disk. It contains the names of files, their sizes, and the dates they were last modified. All of the different directories are stored under one master directory. This directory is called the root directory.

In addition to directories, it uses an area on a disk called the File Allocation Table (FAT). The FAT is similar to our contents page in our book. It holds the information where the file is stored in the disk.

Multilevel directories:

When there are two or more users who share a computer, when you are working on several different projects, the number of files in the directory can become a large and unwieldy. Using directories is one way that we can divide our files into convenient groups. Any one directory can contain many files. This directory may also contain other directories or sub directories. This organized file structure is called a hierarchical directory system.

Specific features of UNIX:

The commands in UNIX are considered to case sensitive. It means, lower case a and uppercase A are considered differently.

Multitasking:

It refers to performing a number of tasks simultaneously. For example when a document is printed, you may run another program to sort large data and at the same time you may edit a document in the foreground screen. UNIX switches between the tasks and executes them one by one at small interval of time. This process of sharing the CPU to perform various tasks simultaneously is called time-sharing. The more number of the tasks are submitted then we end up with slower response from the computer.

Multi-user capability:

UNIX allows the computer to be used by several users through several terminal connected to a powerful computer. A terminal will have a keyboard and a monitor. The computer to which terminals are connected is called as the host computer or server. Any user on the terminal can run various programs, read file information or print a document at the same time. Multi user computer, are economical and efficient compared to stand-alone computers.

Portability:

One of the outstanding features of UNIX is its ability to port itself to another installation. For example, an application program developed in UNIX environment can be used in a different platform.

Security

Unix provides a good security for users. The users are required to authenticate before they use the system. The password is encrypted.

File system

Unix identifies three types of users , owner, group and others. For each group it provides permission on the files like to read, write and execute operation.

4.11 Preliminary commands of DOS:

DIR command:

DIR command is an internal command which is used to display the contents in disk directory. To locate data files and programs on a specific disk, DOS uses the directory along with a file allocation table(FAT).

C:\>DIR

This command will display the disk directory in the default drive.

C:\>DIR/W

This command will display the disk directory in the default drive in a wide format.

C:\>DIR/P

This command will display the disk directory in the default drive but page wise. This command is useful when the disk contains numerous files.

C:\>DIR A:

This command will display the disk contents in A drive.

C:\>DIR *.C

This command will display the disk contents in the default drive with only the files having the extension .C. Here * is known as wild card character. It means all matching characters are represented by *.

CLS command:

CLS command is used to clear the screen. When this command is entered, all the previously displayed text or messages are removed from the screen. The syntax is:

C:\>CLS

REN or RENAME command

REN or RENAME command is used to rename an existing file. Consider the following example to rename a file in the current directory.

C:\>REN A.BAK A.C

Or

C:\>RENAME A.BAK A.C

When this command is entered, the file A.BAK is renamed as A.C.

DEL command

DEL command is used to delete files in a directory. Consider the following example to delete files in the current directory.

C:\>DEL A.BAK

This command will delete all the files in the directory . The message "Bad command or file name" is displayed when the file is not as available in the directory.

C:\>DEL *.BAK

This command will delete all the files in the directory with the extension .BAK The message "Bad command or file name" is displayed when the file is not as available in the directory.

C:\>DEL *.*

This command will delete all the files in the directory. When this command used,the message "Are you sure to delete all files(y/n)?" is displayed. Press y to confirm deletion.

ERASE command

ERASE command is used to erase or remove files in the directory. Consider the following examples to erase files in the directory.

C:\> ERASE A.BAK

This command will erase the file A.BAK in the directory. The message "Bad command or file name" is displayed when the file is not as available in the directory.

C:>\ ERASE *.*

This command will erase all the files in the directory. when this command is entered, the message “Are you sure to delete all files(y/n)?”, is displayed. Press y to confirm deletion.

DATE command

DATE command is used to display the current system date. The computer also maintains a calendar. This command will display the current system date in mm-dd-yy(month-date-year) format and the user may enter the new date. Consider the following example to display the current date.

C:\>DATE
Current date is Sun 09-25-2005
Enter new date (mm-dd-yy):

TIME command

TIME command is used to display the current system time. The computer also maintains a clock. This command will display the current system time in (hours:minutes: seconds) format and the user enter the new time. Consider the following example to display the current time.

C:\> TIME
Current time is 11:37:25.34p
Enter new time:

CD command

CD(change Directory) command is used to change the directory to another specified directory/location in the disk. A message “Invalid directory” is displayed if the directory mentioned is not available. Consider the following example

(i) C:\>CD ABC

This command will change the current directory to the specified directory ABC in the disk. Now the prompt is displayed as follows.

(ii) C:\>CD FC
C:\TC>CD ABCP

These commands will change the current directory to the specified directory TC and then to the directory ABCP in the disk. Now the prompt is displayed as follows.

C:\TC\ABCP>
C:\>CD\TC\ABCP

This command will also change the directory to the specified directory TC and then to the directory ABCP in the disk. Now the prompt is displayed as follows.

C:\TC\ABCP>

CD command can be used in the following ways to quit from a directory or transfer the control to a root directory

C:\TC\ABCP>CD\
C:\TC>CD..
C:\>

MD command

MD(Make directory) command is used to create a new directory in the storage device to store programs. Consider the following example to create a new directory:

C:\>MD ABCP

This command will create a new directory in the current directory. To transfer the control to the new directory a CD command is used.

C:\>CD ABCP
CD:\ABCP>

RD Command

RD(Remove Directory) command is used to remove a directory permanently from the disk. Note that all the files in that directory should be removed before the RD command is used. Also know that you should quit from the directory being removed. Consider the following example:

C:\TC RD ABCP
C:\TC\ABCP>DEL *.*
All files in directory will be deleted!
Are you sure(y/n)? Press y to confirm the delete option.

C:\TC\ABCP>CD..
C:\TC>RD ABCP

One of these commands can be used to remove the directory ABCP from the disk.

COPY command

COPY command is used to copy a file to a new location or directory in the disk. A file cannot be copied to itself in the same directory in the same name.

Consider the following example:

C:\>COPY AB.TXT AB.BAK

This command will copy the file AB.TXT in the same directory as AB.BAK. The first file name in the command AB.TXT is the source file and the file AB.BAK is the target file which is a copy of AB.TXT.

C:\>COPY AB.TXT A:

This command will copy the file AB.TXT from the C drive to A drive and is copied in the same name.

C:\>COPY AB.TXT A: AB.BAK

This command will copy the file AB.TXT from the C: drive to A: drive but the target file is named as AB.BAK.

C:\>COPY *.* A:

This command will copy all the files in the current directory of C: drive to A: drive in the respective file names

C:\>COPY *.* A:

This command will copy all the C program files in the current directory of C: drive to A: drive in the respective file names.

The COPY command can also be used to create a file in the console using the keyword CON along with COPY command.

C:\>COPY CON sample.txt

TYPE command

This command is used to display the contents of a file on the monitor screen.

Consider the following example

C:\>TYPE AB.TXT

This command will display the contents of the file AB.TXT on the screen.

TYPE command can also be used to send the contents of the file to the printer in the console using the keyword PRN.

Consider the following example

C:\>TYPE AB.TXT >PRN

This command will print the contents of the file AB.TXT

FORMAT command:

FORMAT command is used to format a new disk. It is an utility program that is available on hard disk. Only the formatted disks can be used by the operating system to store files or programs. Consider the following example to format a new floppy disk. Insert the new floppy disk in A drive and give the following command.

C:\>FORMAT A:

Nowadays new formatted floppies are available in packs which need not be formatted again. If any old floppy is formatted then the contents will be erased automatically.

DISKCOPY Command:

DISKCOPY command is used to copy all the contents of various directories of a disk in another disk. It is faster and useful to take backup copies.

Consider the following example:

C:\>DISKCOPY A: C:

Now the contents of floppy disk in A drive is copied to the hard disk drive C.

CHKDSK command:

CHKDSK command is used to get the report about a disk such as disk directories,files, storage space available etc. Consider the following example to check the floppy disk in drive A

C:\>CHKDSK A:

Now the details of floppy disk in A drive is displayed. It is also possible to display details using the SCANDISK command.

4.12UNIX commands:

Who command :

Who command is used to list users who are currently logged to the system. The username together with the terminal, date and time the user last logged will be displayed.

```
$who
abhi  tty1  sep 29 13:01
reva   tty2  sep 29  14:15
user1  tty3  sep 29  15:45
$
```

```
$who –u
```

This option –u(means unused) will list the user with the unused time shown below

```
$who
abhi  tty1  sep 29 13:01  00:05
reva   tty2  sep 29  14:15  01:20
user1  tty3  sep 29  15:45
$
```

Note that the users abhi and reva were idle for 5 minutes and 1 hour 20 minutes respectively.

To display user name, terminal line, date and time of login, the who command is given as follows.

```
$who am i
user1 tty3 sep 29 15:45
```

to confirm login name, type the following command

$ logname
user1
$

pwd Command

pwd(print working directory) command will display the current working directory.
$pwd
/usr/user1
$
Note that /(slash) represents the root directory. In MS DOS, the root directory is represented by \(back slash)

echo command

echo command will display the text typed from the keyword. Consider the following example
$ echo Learning UNIX is fun
Learning UNIX is fun
$

cat command

cat command is the simplest way to create a file. It is equivalent to the copy con command in MS DOS. It takes names of zero or more files as argument. Consider the following example. With no arguments ,cat will take input from the keyboard.

$ cat
welcome
welcome
-have a nice day
have a nice day
ctrl –d
$
note that the cat command echoes each line as soon as it has been typed in.

sort command:
sort command is used to sort the contents of a file and name it students_list
$cat > students_list
Abhi 20101
Revathi 20125
Preethi 20104
Ravi 20121

ctrl –d
$
using sort command, the above list may be sorted alphabetically as shown below
$sort students_list
Abhi 20101
Preethi 20104
ravi 20121
Revathi 20125

wc command
wc command is used to count the number of lines, words and characters in a file. Consider the following example.
$wc students_list
5 10 53 students_list

note that there are 5 lines,10 words and 53 characters in the file. Also note that every line is terminated by pressing Enter key which is represented by an invisible new line character. These characters are also accounted to get the number of characters as 53 instead of the actual number of characters as 48.

The wc command has the following options

(i) $ wc –l will display the number of lines in a file.

Example:

```
$wc -l  students_list
 5 students_list
```

(ii) $ wc –w will display the number of words in a file.

```
 $wc –w students_list
10 students_list
```

(iii) $ wc –c will display the number of characters in a file.

```
$wc –c students_list
  53  students_list
```

grep command:
grep command is used to search and display a line for a given word or pattern in a given file name. consider the following example to display the register number of a student.

$grep Abhi students_list
Abhi 20101

Filters:
These refer to any command that can take input from standard input, perform some operations and write the results to standard output.
Consider the following example:
$ who
cseabhi tty1 sep 29 13:01

```
itreva   tty2  sep 29  14:15
ituser1  tty3  sep 29  15:45
$
```

The above list can be short listed to specific users and display the list in alphabetical order as shown below.

```
$ who | grep it | sort
itreva   tty2  sep 29  14:15
ituser1  tty3  sep 29  15:45
$
```

The output of who is fed into the input of grep which will filter and display those users containing the pattern it.

ls command:

ls command is used to list the files stored in the directory
consider the following example

```
$ls
jp.c    test1    sample1     sample2
sam.txt   ex1.c   letter.doc
```

the following options are available with ls command.
The ls –l option is used for long listing of files in the current directory.

```
$ls -l
total 24
dwxr-xr-x2    user1  group   480   Sep 05  02:15   first.dir
-rw-r—r--  1  abhi     group     80  Sep 05 02:13  myfile.c
-rw-r—r--  1  reva   group     80   Sep 05  02:34    jp.c
```

$ls –r will display files in reverse alphabetically order.
$ls* will display all the files and directories in the current directory.

$ls* will display all the files and directories in the current directory.

mkdir command:

mkdir command is used to create a new directory . Consider the following example

```
$mkdir jpdir
$
```

will create a directory named jpdir

rmdir command:

rmdir command is used to remove a directory. Consider the following example.

```
$rmdir jpdir
$
```

note that all fields in the directory should be deleted before removing that directory.

cd command:

cd command is used to switch from the current directory to another directory. Consider the following example.

$ cd tjp
$

note that the current directory will now be tjp. The following options are available with cd command.

$cd or $cd .. is used to switch to the home directory.

$cd/usr/user1 is used to switch to the directory user1.

cp command:

cp command is used to copy a file. Consider the following example.

$ cp jp.prg jp.copy
$

note that jp.prg is an existing file which will be copied as jp.copy. if the file jp.copy already exists in the same directory,it will be over written without any warning.

The following options are available with cp command.

$cp jp.prg/usr/usr1 is used to copy the file to the specified directory.

$cp jp.prg/usr/user1/jp.bak is used to copy and change the name of the file.

rm command:

rm command is used to remove or delete a file. Consider the following example

$rm jp.bak
$

the file jp.bak will be permanently removed without any warning.

The following options are available with rm command

$rm –i jp.bak will remove a file after confirmation from the user.

Rm: remove 'jp.bak' ? y press y then enter to delete the file.

$

mv command

mv command is used to rename a file. Consider the following example.

$mv jp jp.c
$

note that the file jp is renamed as jp.c

man command

man command is used to display the help manual for UNIX commands.

4.13 Summary:

- A computer's programming language consists of strings of binary numbers(0's and 1's) and is the only one language computer can understand
- This symbolic instruction language is called Assembly language.

- Compilers convert the program instructions from human understandable form to the machine understandable form.
- Interpreters also convert the source program to machine language instruction but execute each line as it is entered. The translation of the source program takes place for every run and is slower than the compiled code.
- An editor is used to type the source program and store program in disk. C language uses a turbo editor in MS-DOS system and in Vi editor in UNIX system.
- System software is designed for a specific type of hardware.
- The system software controls the activities of a computer, application programs, flow of data in and out of memory and disk storage.
- Application software are developed for application of the computer to common problems and tasks. They are available for business applications, science and engineering applications and so on.
- An operating system is a program which connects the user and the electronic hardware in a computer.
- It is a set of programs which supervise the activities of a computer and activate the operations of the hardware components.

4.14 Quiz questions

1. Machine language is expressed in terms of -----.
2. Assembly language is written with the help of ------
3. High level language is similar to -------
4. Compiler converts ---- language into ---- language.
5. Assembler converts ----- language into ---- language.
6. Interpreter converts high level language into machine language ---- by ----.
7. edit is an example of ------.
8. Ms-word is an example of ------
9. To display files in the MS-DOS , ---- command is used.
10. To display copy a file in Unix --- command is used.
11. In Unix , commands are ---- sensitive.

Answers

1. 0 and 1.
2. Mnemonics
3. English
4. High, machine
5. Assembly , machine
6. line , line
7. editor
8. word processor
9. DIR
10. cp

11. case

4.15 Exercises

1.What is a machine language?

2. Explain assembly language and an assembler?

3. Mention any three higher level languages?

4. What is a compiler?

5. Give the differences between compiler and interpreter?

6. What is a source program?

7 . List the functions of an editor?

8. Differentiate between system software and application software?

9. Explain operating system and its functions?

10. Write the specific features of DOS and UNIX operating systems?

11. Explain any five MS-DOS commands and UNIX commands ?

Chapter 5

Computing environments

5.0 Introduction

We know, in information technology era, sharing of resources and easy communication are acting as a backbone of any network. Popular example is our Internet. With the help of internet we are able to exchange information and share resources. Hence in this chapter we study different advantages of the network and different forms of the network and their features.

The learning objectives of this chapter are to know
What is a computer network ?
How network is advantageous ?
What are different forms of networks ?
What is a e-mail ?

5.1 Networking of computers and its advantages.

Computer network is defined as an interconnection of autonomous computer. Here autonomous means, there is no master and slave relationship. All computers are equal. Computer network enables to share the resources. Computer networking also refers to connecting computers to share data, application software and hardware devices. Networks allow sharing of information among various computers and permit users to share files. For example a students accesses compilers sitting at one place, where compiler may be stored on the other machine. The students takes printout with the help of one printer connected to the network. The printer can be shared among many students.

Network offers the opportunity to communicate more efficiently with others through electronic mail. Networks allow companies to share software and peripherals such as printers, plotters, scanners and so on. With networking all the computers in an office can be connected to a single printer and scanner. It also helps in using storage devices efficiently.

Computer network acts as a very powerful communication medium. It means people exchange their information. When compared to mainframe computers, network of computers saves money.

5.2 Types of networks

Depending the nature of the distances, protocols (the set of rules used for communication) the network can be classified into LAN (Local Area Network), MAN (Metropolitan Area Network) and WAN (Wide Area Network).

Local Area Network (LAN)

A **LAN** (local area network) is a group of computers and network devices connected together, usually within the same building. By definition, the connections must be high-speed and relatively inexpensive (e.g., token ring or Ethernet). They function to link computers together and provide shared access to printers, file servers, and other services.

Any individual computer connected to a network is called workstation. A workstation may not need a floppy disk or hard disk. A LAN or local area network connects computers and peripherals in a limited area. LAN requires cables to connect workstations. For example LAN is used in a hall or within a building. Figure 5.1 shows Local Area Network , where various departments are connected.

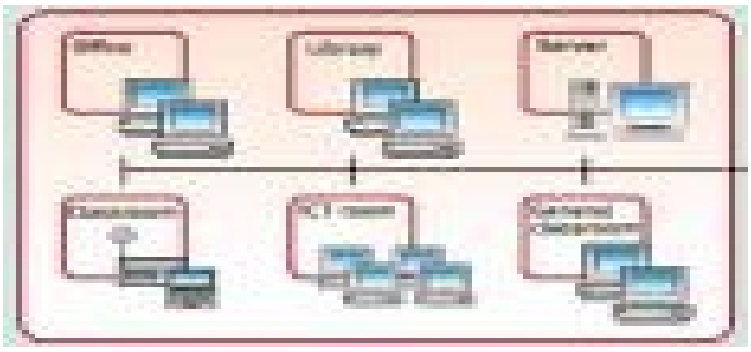


Fig 5.1 Local Area Network

Metropolitan Area Network (MAN)

A MAN (Metropolitan Area Network) is used to connect computers to cover the city or town. The range may be approximately 50 Kilometers. Normally cables and fiber optic cables are used to connect computers. The routing of the messages are fast. Normally central library in a city may be connected by a MAN, so users can access the information. Figure 5.2 shows a typical view of Metropolitan Area Network.

Figure 5.2 Metropolitan Area Network

Wide Area Network (WAN)

A WAN (Wide Area Network) covers large distance like state, country or continents. The WAN uses the fiber optics, cables and even satellites also. Here communication circuits are connected with the help of hardware device called routers. Routers forward small pieces of information called packets from one to another. Internet is the popular one comes under WAN. Some of the examples makes use of internet are reservation of airplane tickets, railway tickets and even cinema tickets. Another facility called e-commerce, where business is carried out through internet. Here people can buy books, articles and so on through registering their wants through the internet. Figure 5.3 shows a typical view of Wide Area Network.

Fig 5.3 Wide Area Network

5.3 Internet

The internet is a massive wide area network ,connecting thousands of computer networks around the world. The internet is a world wide "network of networks". It is a collection of thousands of smaller networks in different countries around the world. It links thousands of academic government, military and public computers, enabling millions of people to share information and other resources.

Internet pathways are used to exchange digitized computer data. The basic services that form the foundation of the internet are e-mail , telnet and FTP. With internet we can easily exchange electronic mail with friends and family anywhere in the world.

Telnet allows you to connect to a remote computer. We can access any of the public services or tools and library databases at the remote site. FTP(File Transfer Protocol) provides for transferring files from one computer to another across the internet.

Internet has many uses. For individuals, the most important uses of internet are e-mail and surfing the Web. One can read the topics of interest like sports, a hobby, a country or any place of interest.

E-mail:

The e-mail stands for electronic mail. One of the major features of computer networking is that messages can be sent electronically to various terminals on the network. The messages are sent very quickly and accurately. E-mail uses the concept of Storing and forwarding messages. It saves a lot of money for the users. Here user registers his/her account with one of the providers. The e-mail account normally contains username and the provider name. For example abc @ yahoo.com , represents abc is the name of the user , who is registered in yahoo provider.

5.4 Summary:

- Computer network is defined as a interconnection of autonomous computers.
- Computer networking enables us to connect computers to share data, application software and hardware devices.
- Networks allow sharing of information among various computers and permit users to share files.
- A LAN or local area network connects computers and peripherals in a limited area.
- A MAN covers the city or town.
- A Wide Area Network covers a large area.
- The internet is a massive wide area network, connecting thousands of computer networks around the world.
- E-mail is a cost effective communication

5.5 Quiz questions

1. Computer network defined as a interconnection of ---------- computers
2. Computer network allows user to ----- the resources
3. -------- network confined to hall or building.
4. ------- is one example of WAN.
5. ------ business is carried our through internet.
6. Small piece of information called ---- are forwarded by router in Internet.

Answers

1. Autonomous
2. share
3. Local area
4. Internet
5. E-commerce
6. packets

5.6 Exercise

1. What is a computer network ? What are the advantages of computer network ?
2. Briefly explain the Local area network, Metropolitan area network and wide area network.
3. What e-mail ? How e-mail works ?
4. What are the uses of Internet ?

C programming

TABLE OF CONTENTS:

b. Arithmetic expressions
c. Modes of expression
d. Arithmetic operators precedence(including parenthesis)
e. Increment and decrement Operators.
f. Relational operators.
g. Logical operators.
h. Relational and logical expressions.
i. Precedence of relational operators and logical operators.
j. The conditional operators
k. Bitwise operators.
l. The comma operator.
m. The precedence of operators among themselves and across all the set of operators.
n. The associativity of operators.
o. Evaluation of expressions involving all the above type of operators.
p. Mathematical functions.
q. Header files
r. Preprocessor directives.

Algorithms and flowcharts:

Algorithms :

The fundamental knowledge necessary to solve problems using a computer is the notion of an algorithm. An algorithm is a precise specification of a sequence of instructions to be carried out in order to solve a given problem. Each instruction tells us what task is to be performed.

The example below will make you understand the specifications:

Example: Recipe for Mutter paneer

Ingredients:

½ kg fresh peas, 400 gms paneer, 2 onoins grated, 2 tomatoes, peeled and chopped 1tsp chilli powder, ½ cup coriander leaves,1/2 turmeric powder,1tsp garlic/ginger paste,4 tsp oil,salt to taste.

Method:

- Step 1: Pressure cook peas in Pressure Pan with 1 ½ cups of water. Drain out excess water, keep peas aside.
- Step 2: Heat oil in the pressure pan and add the onions. Saute' till they turn brown then add tomato pulp.
- Step 3: Add spices, garlic and ginger paste, saute' till the gravy is well blended.
- Step 4: Add cooked peas,paneer and salt.

- Step 5: Simmer for 5 minutes.
- Step 6: Garnish with coriander leaves.

Result:

Mutter paneer ready to serve for 4 people.

The recipe given above is similar to an algorithm but it does not technically qualify as one as the instructions given above are not precise it depends on the individual whether the person has to wait for the gravy to be blended.

We will look at another example to examine another sequence of instructions:

Instructions to knit Sweater:

Input

Needles No.12=2

Wool 4 ply = 9 balls

Method

Step1 : Cast on 133 stitches

Step2: Repeat Steps 3 and 4, 11 times

Step3: Knit 2, *Purl 1, Knit 1, Repeat from * to last stitch, Knit 1.

Step 4: Knit, *Purl 1, Knit 1, Repeat from * to End

……

……

(Similar steps)

Result :

A Sweater

The above example shows:

1. The instructions are much more precise and unambiguous when compared to the recipe for mutter paneer.
2. The number of different types of actions to be carried out are very few.
3. By a proper permutation and combination of this elementary set of actions a virtually infinite number of patterns may be created.

Computers are built to carry out a small variety of elementary instructions. A computer may thus be thought of as a servant who would carry out instructions at very high speed obediently and uncritically. There is a need to give the computer extensive, detailed and correct instructions for solving problems. In order to do this there is need for algorithms which have to be precise, concise and unambiguous.

Flowcharts:

A flowchart depicts pictorially the sequence in which instructions are carried out in an algorithm. Flow charts are used not only as aids in developing algorithms but also to document algorithms.

For visual recognition a standard convention is used in drawing flowcharts. In this standard convention

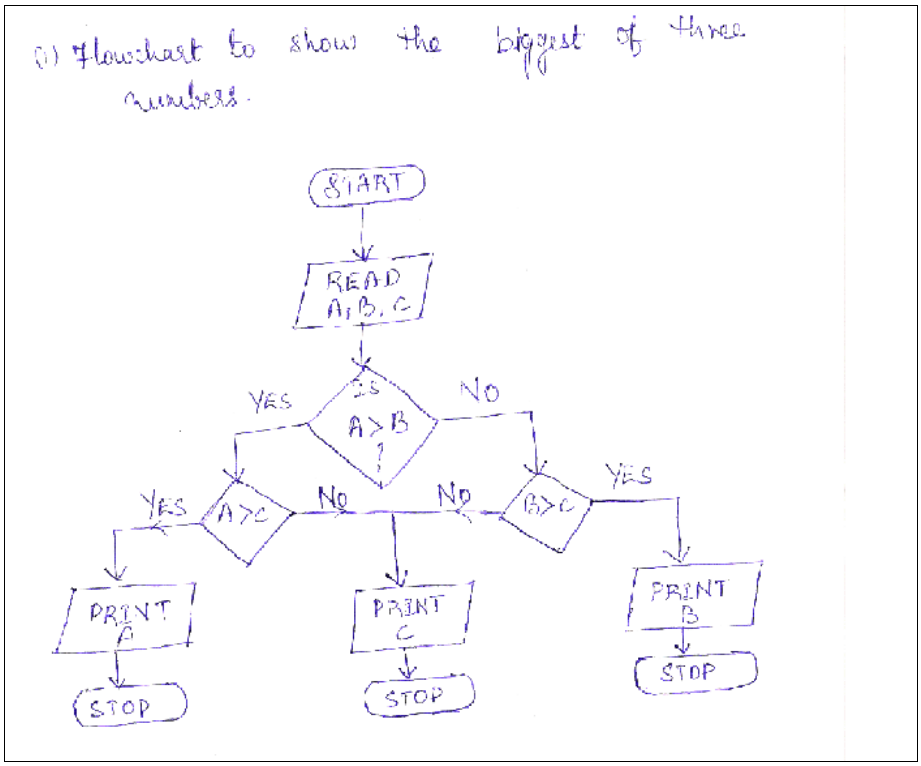
(i) Parallelograms are used to represent input and output operations.
(ii) Rectangles are used to indicate any processing operation such as storage and arithmetic
(iii) Diamond shaped boxes are used to indicate questions asked or conditions tested based on whose answers appropriate exits are taken by a procedure.
(iv) Rectangles with rounded ends are used to indicate the beginning or end points of a procedure.
(v) A circle is used to join different parts of a flowchart. This is called a connector.
(vi) Arrows indicate the direction to be followed in a flowchart.

Algorithm and flowchart for finding biggest of three numbers:

Finding the biggest of three numbers
Step1: Read three numbers A, B, C
Step2: Compare A with B
Step3: If A is larger compare it with C
Step4: If A is larger than C then A is the largest otherwise C is the largest.
Step5: If A is smaller than or equal to B in the first step then B is compared with C.
Step6: If B is larger than C then B is the largest number otherwise C is the largest number.
Step7: Stop

The above algorithm may be expressed much more clearly and concisely using a flowchart.

Flowchart depicting the largest of three numbers. - A



Algorithm to solve the given quadratic equation:

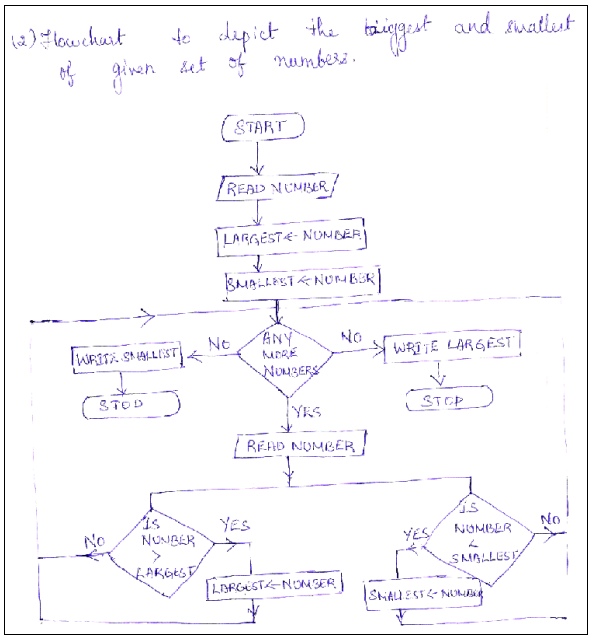
Step1: Read three numbers a, b, c
Step2: Multiply number b twice
Step3: Multiply number 4 with a and c
Step4: Subtract the result of step2 from the result of step3
Step5: Find the square root of the result obtained in Step4
Step6: Add the result of step5 with the negative of number b
Step7: Multiply the number 2 with a
Step 8: Divide the result of step 6 by the result of step 7
Step9: Similarly subtract the result of step5 with the negative of number b
Step10: The result is obtained for the quadratic equations for values x1 and x2.
Step11: Stop.

Flowchart depicting the solution for the quadratic equation….?

Algorithm to find the biggest and smallest of given set of numbers

Step1: Read Number
Step2: Assign the number to the value Largest
Assign the number to the value Smallest
Step3: Repeat steps 4 and 5 as long as numbers are there.
Step4: Read number
Step5: If the number is greater than largest then the number is largest.
If the number is lesser than smallest then the number is smallest.
Step6: Write Largest number
Write Smallest number
Step 7: Stop

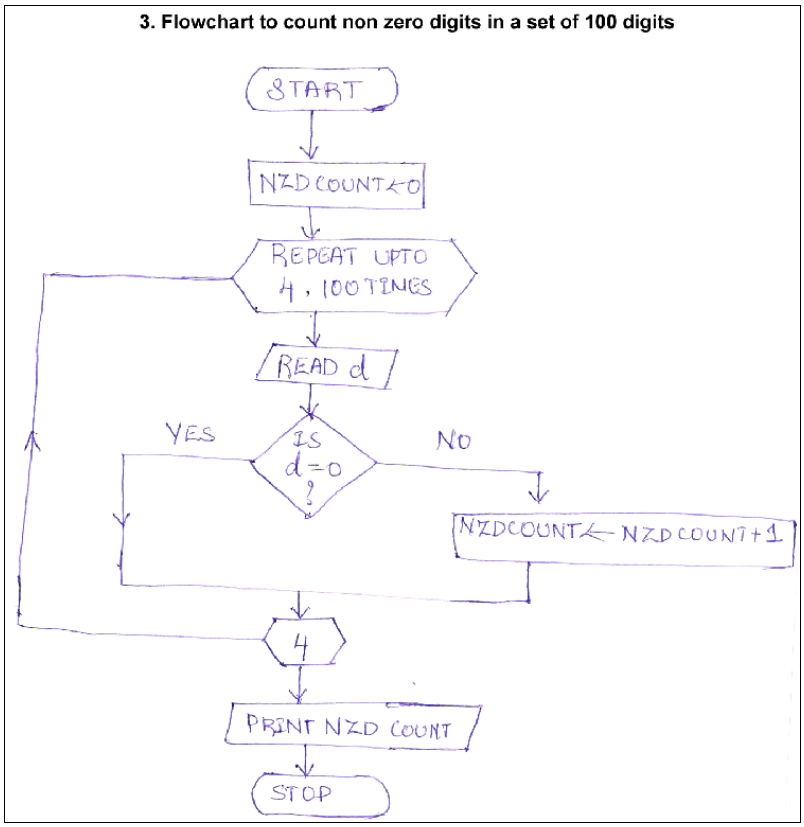
Flowchart to depict the biggest and smallest of given set of numbers. - B



Algorithm to count the number of non-zero data

Step1: Initialize the non-zero data counter nzdcount to 0
Step2: Repeat 100 times all steps up to Step4
Step3: Read d
Step4: If d is not equal to 0 then
nzdcount =nzdcount+1
Step5: Print nzdcount
Step6: Stop

Flowchart depicting the number of non-zero data - C



Summary

- Algorithms :
- The fundamental knowledge necessary to solve problems using a computer is the notion of an algorithm . An algorithm is precise specification of a sequence of instructions to be carried out to solve a given problem.

- Flowcharts:
- Pictorial representation of the task to be carried out. A flowchart depicts pictorially the sequence in which instructions are carried out in an algorithm. Flow charts are used not only as aids in developing algorithms but also to document algorithms.

C language Preliminaries

Introduction: C is a programming language developed at AT & T' s Bell laboratories of USA in 1972. It was designed and written by a system programmer Dennis Ritchie. The main intention was to develop a language for solve all possible applications. C language became popular because of the following reasons.

1. C is a robust language , which consists of number of built-in functions and operators can be used to write any complex program
2. Programs written in c are executed fast compared to other languages.
3. C language is highly portable
4. C language is well suited for structured programming.
5. C is a simple language and easy to learn.

Character Set

The characters that can be used to form words, numbers and expressions depend upon the computer on which the program is run. The characters in C are grouped into the following categories.

1. Letters
2. Digits
3. Special characters
4. White Spaces

Letters	Digits
Uppercase A…Z	All decimal digits 0…9
Lowercase a….z	

Special characters

, comma	& ampersand
. period	^ carat
; semicolon	*asterisk
: colon	-minus sign
? question mark	+ sign
‘ apostrophe	< opening angle bracket
! exclamation mark	(or less than sign)
| vertical bar	> closing angle bracket
/ slash	(or greater than sign)
\ backslash	(left parenthesis
~ tilde	) right parenthesis
_ underscore	[left bracket
$ dollar sign	] right bracket
% percent sign	{ left brace
# number sign	} right brace

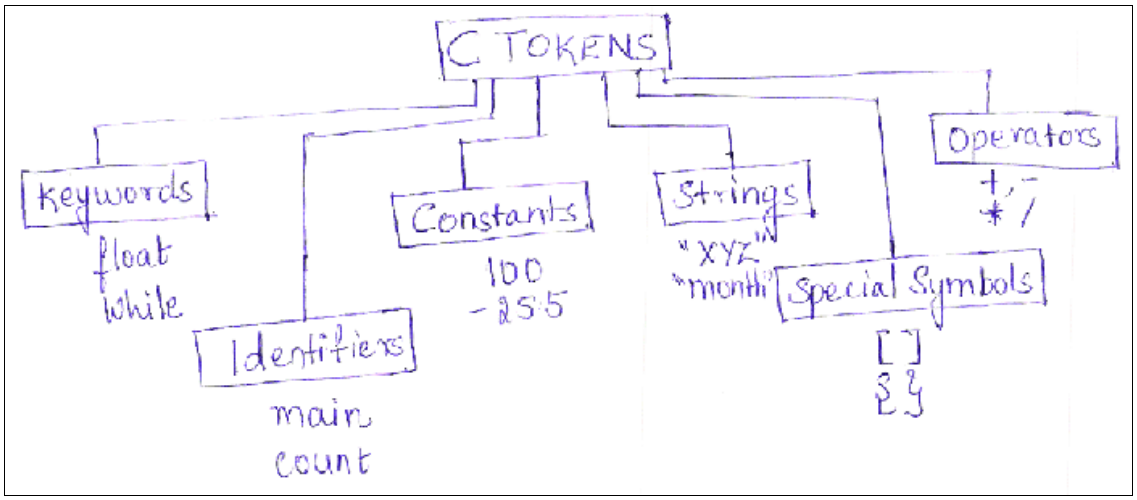
White Spaces

Blank Space
Horizontal tab
Carriage return
New line
Form feed

C tokens:

In a passage of text , individual words and punctuation marks are called tokens.

Similarly in a C program the smallest individual units are known as C tokens. C has 6 types of tokens. The below diagram shows the types of C tokens with examples. - D



Keywords and Identifiers:

In c language every word is classified into either keyword or identifier. All keywords have fixed meanings and these meanings cannot be changed. These serve as basic building blocks for program statements. All keywords must be written in lowercase. The list of all ANSI C keywords are listed below

ANSI C Keywords

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

Identifiers refer to the names of variables, functions and arrays. These are user defined names and consist of a sequence of letters and digits, with a letter as a first character. Both uppercase and lowercase letters are permitted, although lowercase letters are commonly used. The underscore character is also permitted in identifiers. It is usually used as a link between two word in long identifiers.

Structure of a C program:

The basic structure of a C program is shown below

```
Documentation Section
---------------------------------------------
Link Section
---------------------------------------------
Definition Section
---------------------------------------------
Global Declaration Section
_____________________________________________
main() Function Section
{

        Declaration Part
        ----------------------
         Executable Part
}
-------------------------------------------------------------------------------------------------
Subprogram section

  Function 1
_____________________________________________
  Function 2
  ------------
       .
       .
       .
   Function n
```

The documentation section consists of a set of comment lines giving the name of the program, the name author and other details which the programmer would like to use later.

The link section provides instructions to the compiler to link functions from the system library.

The definition section all symbolic constants.

There are some variables that are used in more than one function. Such variables are called global variables and are declared in the global declaration section.

Every C program must have one main() function section. This section contains two parts declaration part and executable part. The declaration part declares all the variables used in the executable part. There should be at least one statement in the executable part. These two parts must appear between the opening and the closing

braces. The program execution begins at the opening brace and ends at the closing brace. The closing brace of the main function section is logical end of the program. All statements in the declaration and executable parts end with a semicolon.

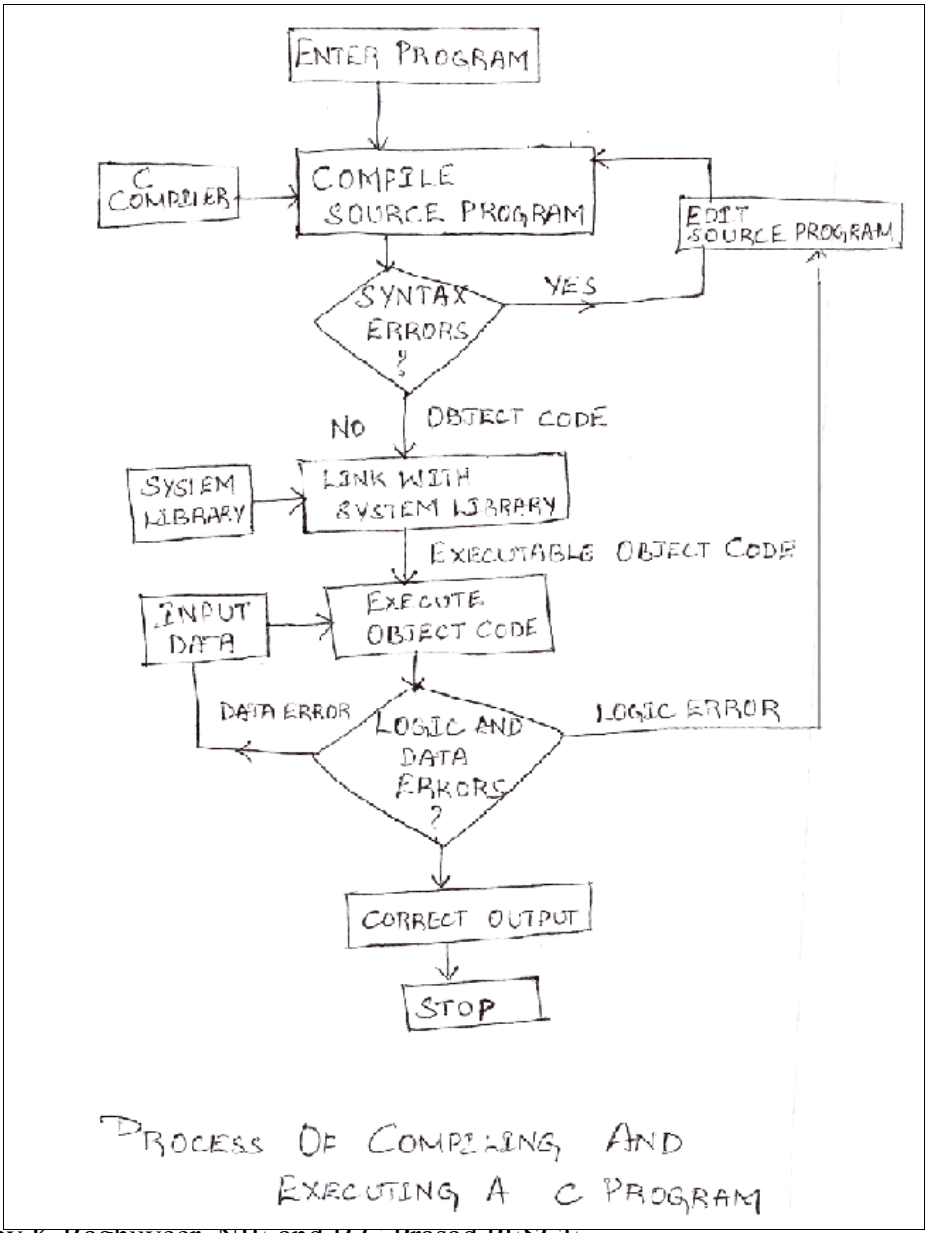
The subprogram section contains all the user-defined functions that are called in the main function. The main function is very important compared to other sections.

Executing a C program

Executing a program written in C involves a series of steps. These are

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

The figure below illustrates the process of creating, compiling and executing the program.



PROCESS OF COMPILING AND EXECUTING A C PROGRAM

Summary:

In c language every word is classified into either keyword or identifier. All keywords have fixed meanings and these meanings cannot be changed.

Numeric Constants and Variables:

Integer Constants:

An integer constant refers to a sequence of digits. There are three types of integers, namely decimal, octal and hexadecimal.

Decimal integers consist of a set of digits 0 through 9, preceded by an optional – or + sign. Some examples of decimal integer constants are

123
-431
0
34567
+678

Spaces, commas, and non-digit characters are not permitted between digits. For example

15 750
20,000
Rs 1000

are illegal numbers.

An octal integer constant consists of any combination of digits from the set 0 through 7 with a leading 0.

Some examples are:

037
0
0435
0567

A sequence of digits preceded by 0x is considered as hexadecimal integer. They may also include alphabets A through F or a through F. The letters A through F represent the numbers 10 through 15. The examples for hexadecimal integers are:

0x2
0x9F
0xbcd
0x

Floating-point constants:

Integer numbers are inadequate to represent quantities that vary continuously, such as distances ,heights ,temperatures ,prices and so on. These quantities are represented by numbers containing fractional parts like 23.78. Such numbers are called floating-point constants or real constants. Examples for floating-point constants are given below.

213.
.95

-.71
+.5

A real number may also be expressed in exponential(or specific notation). For example the value 213.45 may be written as 2.1345e2 in exponential notation. e2 means multiply by 102. The general form is

mantissa e exponent

Character Constants:

A character constant contains a single character enclosed within a pair of single quote marks. Examples of character constants are:

'5' 'X' ';' ' '

Note that the character constant '5' is not the same as the number 5. The last constant is a blank space.

Character constants have integer values known as ASCII values. For Example, the statement

printf ("%d", 'A');

would print the number 65,the ASCII value of the letter a. Similarly, the statement

printf("%c", 65)

would give the output as letter 'A'

It is also possible to perform arithmetic operations on character constants.

Backslash Character Constants:

C supports some special backslash character constants that are used in output functions. For example, the symbol '\n' stands for new line character. The below table gives you a list of backslash constants.

Backslash Character Constants

Constant	Meaning
'\a'	audible alert(bell)
'\b'	back space
'\f'	form feed
'\n'	new line character
'\r'	carriage return
'\t'	horizontal tab
'\v'	vertical tab
'\''	single quote
' \" '	double quote
'\?'	question mark
'\\'	backslash mark
'\0'	null character

Note that each one of them represents one character, although they consist of two characters. These character combinations are called escape sequences.

String constants:

A string constant is a sequence of characters enclosed in double quotes. The letters may be numbers, special characters and blank space.

Examples are given below

"THANK YOU"
"2345"
"?....."
"7+8-9"
"X"

Remember that a character constant 'X' is not equivalent to the single character string constant("X"). A single character string constant does not have an equivalent integer value as a single character constant. These type of constants are used in programs to build meaningful programs.

Meaning of variables:

A variable is a data name that may be used to store a data value. Unlike the constants that remain unchanged during the execution of a program, a variable may take different values at different times during execution.

A variable name can be chosen by the programmer in a meaningful way so as to reflect its function or nature in the program. Some examples are given below.

Average
Height
Total
Counter_1

Rules for defining variables:

Variable names may consist of letters, digits, and the underscore(_) character, subject to the rules given below:

1. The variables must always begin with a letter. Some systems permit underscore as the first character.
2. ANSI standard recognizes a length of 31 characters. However, the length should not be normally mare than eight characters. Since first eight characters are treated as significant by many compilers.
3. Uppercase and lowercase are significant. That is ,the variable Rate is not the same as rate or TOTAL.
4. The variable name should not be a keyword.
5. White space is not allowed.

Some examples are given below:

Abhi	Value	I_rate
Mumbai	s1	ph_value

Rate sum1 distance

The examples given below are invalid:

345 (rate)

% 56 nd

Declaration of variables:

Identifiers which are used as variable names should be prefixed as integer or float by the following declaration should appear at the beginning of a program before the variable names are used.

type _name variable name……variable name;

The type_name is always a reserved word. The type_name available for variable names storing numbers are int for integers and float for floating point numbers.

Valid examples are given below:

int n, height ,count ,digit;

float rate, average , y_coordinate,p1;

When a variable name is declared then a memory location is identified and given this name.

The following declarations of variables are invalid:

Float a ,b ,c ; (comma after float is not valid)

Int :x; (: is not valid)

Real x, y;(real is not the correct type_name)

Assignment operators:

Assignment operators are used to assign the result of an expression to a variable. There are several different assignment operators in C. All of them are used to form assignment expressions which assign the value of an expression to an identifier. The commonly used assignment operator is '=' operator. The general syntax is:

Identifier=expression

Where identifier generally represents a variable, and expression represents a constant, a variable or a more complex expression.

For example:

```
x=y+1;
a=2*c;
rate= 45;
delta= 0.001
sum=a+b;
```

Note that the assignment operator = and the equality operator == are distinctly different. The assignment operator is used to assign a value to an identifier, whereas the equality operator is used to determine if two expressions have the same value. These operators cannot be used in place of one another.

Assignment expressions and assignment statements:

The general syntax for an assignment statement is:

v op=exp;

Where v is a variable , exp is an expression and op is a C binary arithmetic operator. The operator op=is known as the shorthand assignment operator.

The assignment statement

v op=exp;

is equivalent to

v=v op(exp);

with v evaluated only once. Consider the following example:

a+=b+1;

This is same as the statement

a=a+(b+1);

The shorthand operator += means 'add b+1 to a' or 'increment a by b+1' . For b=2, the above statement becomes a+=3;

and when this statement is executed, 3 is added to a, if the old value of a is 4 then the new value of a will be 7.

Consider the following expression, evaluate the equivalent expression and their final result.

```
int  i, j, k, m, n ;
i=j=2;
```

Expression	Equivalent expression	value
i+=++j+3	i=(i+((++j)+3))	8

When an assignment statement is executed, the expression on the right of the assignment operator is first evaluated and the number obtained is stored in the storage location named by the variable name appearing on the left of the assignment operator.

Arithmetic Conversion:

C allows mixing of float with integers in expressions. In such cases integers are converted to float before computation. C provides explicit type conversion functions using which a programmer can intentionally change the type of expressions in an arithmetic statement. This is done by a unary operator called a cast. If we write(type_name) expression the expression is converted to the type_name

For example

(float)(integer expression or variable name)

then the integer expression or variable name is converted to float. If we write:

(int)(float expression or variable name)

then the float expression or variable name is converted to integer.

```
X = (int) 2.3 ;
Y = (float) 234;
```

Conversion with float and double:

If an expression real variables declared as float and double appear together all variables are converted to double.

For example, in the statement:

```
float x y ,s;
double p ,q ,z;
x=q/(y + s);
```

the variable names y and s are assumed to be double. The result is double as z is double.

Fundamental data types:

C language has varieties of data types. Storage representations and machine instructions differ from machine to machine. The variety of data types available allow the programmer to select the appropriate to the needs of the application as well as the machine.

The fundamental or primary data types are integer(int), character(char), floating point(float), and double-precision floating point(double). Many of them also has extended data types such as long, double, short, unsigned and signed. The range of basic four types are given below:

Data types	Range of values
char	-128 to 127
int	-32,768 to 32,767
float	3.4e-38 to 3.4e+38
double	1.7e-308 to 1.7e+308

Char, int, float and double are all keywords and therefore their use is reserved. They may not be used as names of variables. Char stands for "character" and int stands for "integer". The keywords short int, long int and unsigned int may be and usually are, shortened to just short, long, and unsigned, respectively. Also, double and long float are equivalent, but double is the keyword usually used.

Integer Types:

Integers are whole numbers with a range of values supported by a particular machine. Generally integers occupy one word of storage. If we use a 16 bit word length, the size of the integer value is limited to the range -32768 to +32767. A signed integer uses one bit for sign and 15 bits for the magnitude of the number. Similarly, a 32 bit word length can store an integer ranging from -2,147,483,648 to 2,147,483,647

C has three classes of integer storage, namely short int, int, and long int in both unsigned and signed forms. For example, short int represents fairly small integer.
Values and requires half the amount of storage as a regular int number uses. Unlike signed integers, unsigned integers use all the bits for the magnitude of the number and are always positive. Therefore, for a 16 bit machine, the range of unsigned integer numbers will be from 0 to 65,535.

We declare long and unsigned integers to increase the range of values.

Floating point types:

Floating point(or real) numbers are stored in 32 bits(on all 16 bit and 32 bit machines), with 6 digits of precision. Floating point numbers are defined in C by the keyword float. The type double can be used when the accuracy provided by a float number is not sufficient. A double data type number uses 64 bits giving a precision of 14 digits. These are known as double precision numbers. To extend the precision further, we may use long double which 80 bits..

Character types:

A single character can be defined as a character(char) type data. Characters are usually stored in 8 bits(one byte) of internal storage. The qualifier signed or unsigned may be explicitly applied to char. While unsigned chars have values between 0 and 255, signed chars have values from -128 to 127.

Size and range of data types on a 16-bit machine

Type	Size(bits)	Range
Char or signed char	8	-128 to 127
Unsigned char	8	0 to 255
Int or signed int	16	-32,768 to 32,767
Unsigned int	16	0 to 65535
Short int or Signed short int	8	-128 to 127
Unsigned short int	8	0 to 255
Long int or signed Long int	32	-2,147,483,648 to 2,147,483,647
Unsigned long int	32	0 to 4,294,967,295
Float	32	3.4E-38 to 3.4+38
Double	64	1.7E-308 to 1.7E+308
Long double	80	3.4E-4932 to 1.1E+4932

Data Type	Description	Typical Memory Requirements
Int	integer quantity	2 bytes or 1 word
Short	Short integer quantity (may contain fewer digits than int)	2 bytes or 1 word (varies from one computer to another)
long	long integer quantity (may contain more digits than int)	1 or 2 words (varies from one computer to another)
unsigned	Unsigned(positive) integer quantity(maximum permissible quantity is approximately twice as large as int)	2 bytes or 1 word (varies from one computer to another)
char	Single character	1 byte
signed char	Single character, with numerical values ranging from 128 to +127	1 byte
unsigned char	Single character, with numerical values ranging from 0 to 255	1 byte
float	floating-point number(i.e., a number containing a decimal point and/or an exponent)	1 word
double	double-precision floating-point number(i.e., more significant figures, and an exponent that may be larger in magnitude)	2 words
long double	Double-precision floating-point number(may be higher precision than double)	2 or more words (varies from computer to another)
void	Special data type for functions that do not return any value	(not applicable)
enum	Enumeration constant(special type of int)	2 bytes or 1 word

(varies from
one computer
to another)

Short hand assignment operators:

In addition to the usual assignment operator '=' C has a set of assignment operators called shorthand operators.

Some of the shorthand operators are listed below:

=
+=
-=
*=
/=
%=
>>=
<<=
&=
^=
!=

Declaring variables as constant:

We may want the value of certain variables to remain constant during the execution of a program. We can achieve this by declaring the variable with the qualifier const at the time of initialization.

Example:

Const int class_size=45;

Const is a new data type qualifier by ANSI standard. This tells the compiler not to modify the value of the variable class_size in the program. However, it can be used on the right_hand side of an assignment statement like any other variable.

Declaring variables as Volatile:

There is another qualifier called volatile that could be used to tell explicitly the compiler that a variable's value may be changed at any time by some external sources(from outside the program).

For Example:

Volatile int date;

The value of date may be altered by some external factors even if it does not appear on the left hand side of an assignment statement. When we declare a variable as volatile, the compiler will examine the value of the variable each time it is encountered to see whether any external alteration has changed the value.

We can modify the value of a variable declared as volatile by its own program. If we wish that the value must not be modified by the program while it may

be altered by some other process, then we may declare the variable as both const and volatile as shown below:

Volatile const int location=100

Symbolic constants:

We often use certain unique constants in a program. These constants may appear repeatedly in a program. One such example of such a constant is 3.142, representing the value of the mathematical constant "pi". These constants may appear repeatedly in a number of places in the program. The cc compiler has built into it a preprocessor.
A constant is defined as follows:

#define symbolic-name value or constant

Valid examples are:

```
#define LIMIT  100
#define  PI  3.14159
#define          TOLERANCE   1.0e-5
```

If the above lines occur in a file that is being compiled, the preprocessor first changes all occurrences of the identifier LIMIT to 100 and all occurrences of the identifier PI to 3.14159. A # define line may occur anywhere in a program, but it must start in column 1. Normally #define statements are placed at the beginning of the file.
The use of symbolic constants in a program makes it more readable.
If a constant has been defined symbolically by means of the #define facility and used throughout a program, it is easy to change the constant later, if necessary.

Symbolic names are sometimes called constant identifiers. The following rules apply to a #define statements which define a symbolic constants.

1. # must be the first character in the line.
2. No blank between pound sign '#' and the word define is permitted.
3. A blank space is required between #define and symbolic name and between the symbolic name and the constants.
4. #define statement does not end with semicolon(;)
5. Symbolic names have the same form as variable names.
6. After definition ,the symbolic name should not be assigned any other value within the program by using an assignment statement . For example, PI=12.9 is illegal.
7. #define statements may appear anywhere in the program, but before it is referenced in the program, but the usual practice is to place them in the beginning of the program.
8. Symbolic names are NOT declared for data types. Its data type depends on the type of constant.
9. Only one symbolic constants can be declared in one line.

Some of the invalid #define statements.

```
#define A = 2.5                 /*'=' sign is not permitted  */
# define PI 3.142               /* No white space between # and define  */
# define A  10;                 /* No semicolon at the end*/
```

```
#define A=10, b =25          /*One statement can define only one name*/
#Define SUM 25                    /*define should be in lowercase*/
#define COUNT$  100               /*$ symbol is not permitted*/
```

Multiple assignment statements:

C also has the provision to do multiple assignments For example , we can write:

a=b=c=1

1 will be assigned to c. The value of c will be assigned to b and the value of b to a. Thus a=1,b=1,c=1.

In the statement

a=b= c + d-e

C language will first calculate c + d - e. This will be assigned to b. The value stored in b will be assigned to a. Thus an assignment b=(c + d - e) can be thought to have a value which is the value of b. The = operator is said to be right associative as we scan the statement from right to left . + and – operators in c + d-e are left associative as we can scan the expression from left to right.

We can also write:

a=x=b=c + d-e

which will be interpreted as:

a=(x=(b=c + d-e))

if we write: a+=b*=c+d-e it is equivalent to:

a=a + b=a+(b*(c +d-e))

Observe that when we use the operator += or *= the right hand side assumed to be enclosed in parenthesis.

Thus

X*=y+1

is x=x*(y+1) and not x=x*y+1

Input and Output Functions:

Scanf functions:-

The function scanf() is used to read data into variables from the standard input, namely a keyboard. The general format is:

Scanf(format-string, var1,var2,……varn)

Where format-string gives information to the computer on the type of data to be stored in the list of variables var1,var2……varn and in how many columns they will be found

For example, in the statement:

Scanf("%d %d", &p, &q);

The two variables in which numbers are used to be stored are p and q. The data to be stored are integers. The integers will be separated by a blank in the data typed on the keyboard.

A sample data line may thus be:

456 18578

Observe that the symbol &(called ampersand) should precede each variable name. Ampersand is used to indicate that the address of the variable name should be found to store a value in it. The manner in which data is read by a scanf statement may be explained by assuming an arrow to be positioned above the first data value. The arrow moves to the next data value after storing the first data value in the storage location corresponding to the first variable name in the list. A blank character should separate the data values.

The scanf statement causes data to be read from one or more lines till numbers are stored in all the specified variable names.
No that no blanks should be left between characters in the format-string. The symbol & is very essential in front of the variable name.
If some of the variables in the list of variables in the list of variables in scanf are of type integer and some are float, appropriate descriptions should be used in the format-string.

For example:

Scanf("%d %f %e", &a , &b, &c);

Specifies that an integer is to be stored in a, float is to be stored in b and a float written using the exponent format in c. The appropriate sample data line is:

485 498.762 6.845e-12

Printf function:

The general format of an output function is

Printf(format-string, var1,var2…..varn);

Where format-string gives information on how many variables to expect, what type of arguments they are , how many columns are to be reserved for displaying them and any character string to be printed. The printf() function may sometimes display only a message and not any variable value. In the following example:

printf("Answers are given below");

The format-string is:

Answers are given below

And there are no variables. This statement displays the format-string on the video display and there are no variables. After displaying, the cursor on the screen will remain at the end of the string. If we want it to move to the next line to display information on the next line, we should have the format-string:

printf("Answers are given below\n");

In this string the symbol \n commands that the cursor should advance to the beginning of the next line.

In the following example:

printf("Answer x= %d \n", x);

%d specifies how the value of x is to be displayed. It indicates the x is to be displayed as a decimal integer. The variable x is of type int. %d is called the conversion specification and d the conversion character . In the example:

printf("a= %d, b=%f\n", a, b);

the variable a is of type int and b of type float or double. % d specifies that a is to be displayed as an integer and %f specifies that, b is to be displayed as a decimal fraction. In this example %d and %f are conversion specifications and d, f are conversion characters.

Example to indicate how printf() displays answers.

```
/*Program illustrating printf()*/
# include<stdio.h>

main()
{
        int a= 45,  b= 67
        float x=45.78 , y=34.90
printf("Output:\n");
printf("1,2,3,4,5,6,7,,8,0\n");
printf("\n");
printf("%d, %d,,%f ,%f \n" , a,b,x,y);
printf("\n");
}
```

Output:

1234567890

45,67,45.78,34.90

Example for illustrating scanf and printf statements:

```
/* Program for illustrating use of scanf and printf statements */

#include<stdio.h>
main()
{
 int a,b,c,d;
float x,y,z,p;
scanf("%d %o %x %u", &a, &b ,&c ,&d);
printf("the first four data displayed\n");
printf(("%d %o %x %u \n", a,b,c,d);
scanf("%f %e %e %f", &x, &y, &z, &p);
printf("Display of the rest of the data read\n");
```

```
printf("%f %e %e %f\n", x,y,z,p);
printf("End of display");
}
```

Input:
-768 0362 abf6 3856 -26.68 2.8e-3 1.256e22 6.856

Output:

```
The first four data displayed
-768 362 abf6 3856
Display of the rest of the data read
-26.680000 2.800000 e-03  1.256000e+22  6.866000
End of display
```

Formatted input and output using format specifiers:

The function scanf is the input analog of printf, providing many of the same conversion facilities in the opposite direction.

int scanf (char *format, …..)

scanf reads characters from the standard input, interprets them according to the specification in format, and stores the results through the remaining arguments. The format argument is described below; the other arguments, each of which must be a pointer, indicate where the corresponding converted input to be stored.

scanf stops when it exhausts its format string, or when some input fails to match the control specification. It returns as its value the number of successfully matched and assigned input items. This can be used to decide how many items were found . On end of file EOF is returned; note that this is different from 0, which means that the next input character does not match the first specification in the format string. The next call to scanf resumes searching immediately after the last character already converted.

A conversion specification directs the conversion of the next input field. Normally the result is placed in the variable pointed to by the corresponding argument. If assignment suppression is indicated by the * character, however, the input field is skipped; no assignment is made. An input field is defined as a string of non-white space characters; it extends either to the next white space character or until the field width, if specified, is exhausted. This implies that scanf will read across line boundaries to find its input, since new lines are white space. (White space characters are blank, tab, new line, carriage return, vertical tab and form feed).

Conversion characters are shown in the table below:

CHARACTER	INPUT DATA; ARGUMENT TYPE
d	decimal integer; int *

i integer; int * The integer may be in octal (leading 0) or hexadecimal (leading 0x or 0x)

o octal integer(with or without leading zero); int *.

u unsigned decimal integer; unsigned int *

x hexadecimal integer (with or without leading ox or ox)

c characters; char *. The next input characters (default 1) are placed at the indicated spot. The normal skip over white space is suppressed; to read the next non-white space character, use of %1s.

s character string (not quoted); char *, pointing to an array of characters large enough for the string and a terminating '\0' that will be added.

e,f,g floating-point number with optional sign, optional decimal point and optional exponent; float *.

% literal %; no assignment is made.

The output function printf translates internal values to characters. The general syntax is

int printf (char *format, arg1,arg2……..)

printf converts, formats, and prints its arguments on the standard output under control of the format. It returns the number of characters printed.

The format string contains two types of objects: ordinary characters, which are copied to the output stream, and conversion specifications each of which causes conversion and printing of the next successive argument to printf. Each conversion specification begins with a % and ends with a conversion character. Between the % and the conversion character there may be in order:

- A minus sign, which specifies left adjustment of the converted argument.
- A number that specifies the minimum field width. The converted argument will be printed in a field at least this wide. If necessary it will be padded on the left or right, to make up the field width.
- A period, which separates the field width from the precision.
- A number, the precision, that specifies the maximum number of characters to printed from a string, or the number of digits after the decimal point of a floating point value, or the minimum number of digits for an integer.
- An h if the integer is to be printed as a short, or l if as a long.

CHARACTER ARGUMENT TYPE

d,i	int; decimal number
o	int; unsigned octal number(without a leading zero)
x, x	int; unsigned hexadecimal number(without a leading ox or OX), using abcdef or ABCDEF for 10….15
u	int; unsigned decimal number
c	int; single character.
S	char *; print characters from the string until a '\o' or the number characters given by the Precision.
F	double; [-] m.dddddd, where the number of d's Is given by the precision(default 6)
e,E	double; [-] m.dddddd e+/-xx or [-] m.dddddd E+/- xx, where the number of d's is given By the precision(default 6)
g,G	double; use %e or %E if the exponent is less than or equal to the precision; otherwise use %f. Trailing zeros and a trailing decimal point are not printed.
P	void *; pointer(implementation-dependent Representation).
%	no argument is converted ; print a %

A width or precision may be specified as * , in which case the value is computed by converting the next argument(which must be an int). For example, to print at most max characters from a string s.

Printf ("%.*s", max, s);

Most of the format conversions have been illustrated in earlier . One exception is precision as it relates to strings. The following example shows the effect of a variety of specifications in printing "hello world".

```
:%s:          :hello, world:
:%10s:        :hello, world:
:%.10s:       :hello, wor:
:%-10s:       :hello,  world:
:%.15s:       :hello,  world:
:%-15s:       :hello,  world:
:%15.10s:     :        hello, wor:
:%-15.10s:    :hello, wor     :
```

Address Operator:

The unary & operator takes the address of its operand. The operand must be an lvalue referring neither to a bit-field not to an object declared as register, or must be of function type. The result is a pointer to the object or function referred to by the lvalue. If the type of the operand is T, the type of the result is "pointer to T".

The getchar function

Single characters can be entered into the computer using the C library function getchar. The getchar function is a part of the standard C I/O library. It returns a single character from a standard input device typically a keyboard. The function does not require any arguments, though a pair of empty parentheses must follow the word getchar.

The general syntax

Character variable=getchar();

Where character variable refers to some previously declared character variable.

Example:

A C program contains the following statements

```
char c;
…….
C=getchar();
```

The first statement declares that c is a character-type variable. The second statement causes a single character to be entered from the standard input device and then assigned to c.

If an end-of-file condition is encountered when reading a character with getchar function, the value of the symbolic constant EOF will automatically be returned. (This value will be assigned within the stdio.h file. Typically, EOF will be assigned the value -1, though this may vary from one compiler to another).

The getchar function can also be used to read multi character strings, by reading one character at a time within a multi pass loop.

The putchar function:

Single characters can be displayed using the C library function putchar. This function is complementary to the character input function getchar. The putchar function, like getchar, is a part of the standard C I/O library. It transmits a single character to a standard output device. The character being transmitted will normally be represented as a character type variable. It must be expressed as an argument to the function, enclosed in parentheses, following the word putchar.

The general syntax is

putchar(character variable)

where character variable refers to some previously declared character variable.

A C program contains the following statements

```
Char c;
……
putchar(c);
```

C programs:

1) Program to demonstrate printf statement

```
#include<stdio.h>

main()
{
  printf("hello, world\y");
  printf("hello, world\7");
  printf("hello, world\?");
                                                                    }
```

2) Program to convert farenheit to Celsius

```
#include<stdio.h>
main()
{
   float fahr, Celsius;

printf(" enter the value for farenheit\n");
scanf(" %f", &fahr);
Celsius=(5.0/9.0)*fahr-32.0;
printf("%f  %f \n", fahr,Celsius);
}
```

3) Program to depict interactive computing using scanf function.

```
#include<stdio.h>

main()
{
        int number;
        printf("enter an integer number\n");
    scanf("%d",  &number);

     If (number<100)
          {
```

```
        printf("Your number is smaller than 100\n\n");
            else
            printf("Your number contains more than two digits\n");
}
```

Output

```
Enter an integer number
54
Your number is smaller than 100

Enter an integer number
108
Your number contains more than digits
```

4) Program to depict interactive investment program

```
#include<stdio.h>

main()
{
  int year,period;
 float amount,inrate,value;

printf("Input amount , interest rate and period \n\n");
scanf("%f %f %d", &amount, &inrate,&period);
printf("\n");
year=1;

while(year<=period)
{
          value amount + inrate*amount;
          printf("%2d Rs. %8.2f\n", year, value);
          amount=value;
          year=year+1;
}
}
```

5) Program to calculate the average of a set of N numbers

```
#define N 10

main()
{
   int count;
    float sum, average,number;
```

```
sum=0;
count=0;
while(count<N)
{
   scanf("%f", &number);
   sum=sum+number;
count=count+1;
}

average= sum/N;
printf("N=%d Sum= %f", N, sum);
printf("Average=%f", average);
}
```

6) Program to convert days to months and days

```
    #include<stdio.h>

  main()
 {
        int months,days;
        printf("enter days \n");
        scanf("%d", &days);
        months=days/30;
        days=days%30;
        printf("Months = %d  Days= %d", months,days);
   }
```

Operators In C

Arithmetic operators:

C provides all the basic arithmetic operators. The operators +,-,* and / all work the same way as they do in other languages. These can operate on any built-in data type allowed in C. The unary minus operator, in effect, multiplies its single operand by -1. Therefore, a number preceded by a minus sign changes its sign.

Operator	Meaning
+	Addition or unary plus
-	Subtraction or unary minus
*	Multiplication
/	Division
%	Modulo division

Integer division truncates any fractional part. The modulo division produces the remainder of an integer division.
Examples are:

```
        a - b          a + b
a * b          a / b
        a % b          -a * b
```

Here a and b are variables and are known as operands. The modulo divison operator % cannot be used on floating point data.

Arithmetic expressions:

An arithmetic expression is a combination of variables, constants and operators arranged as per the syntax of the language.

Expressions are evaluated using an assignment statement of the form

Variable=expression;

The table below shows the algebraic expression and C language expression

Algebraic expression	C expression
a x b-c	a*b-c
(m + n) (x + y)	(m + n) *(x + y)
(a b)/c	a*b/c
3x2+2x+1	3*x*x+2*x+1
x/y +c	x/y + c

Variable is any valid C variable name. When the statement is encountered, the expression is evaluated first and the result then replaces the precious value of the variable on the left-hand side. All variables used in the expression must be assigned values before evaluation is attempted.

```
x=a*b-c;
y=b/c*a;
z=a-b/c+d;
```

The blank space around an operator is optional and adds only to improve readability. When these statements are used in a program, the variables a ,b ,c and d must be defined before they are used in the expressions.

Modes of expression:

There are three different modes of expression.

1. Integer Arithmetic
2. Real Arithmetic

3. Mixed-mode Arithmetic

Integer Arithmetic

When both the operands in a single arithmetic expression such as a+b are integers, the expression is called an integer expression, and the operation is called integer arithmetic. This mode of expression always yields an integer value. The largest integer value depends on the machine, as pointed out earlier

Example:

If a and b are integers then for a=14 and b=4

We have the following results:

a - b=10
a + b = 18
a * b = 56
a / b = 3
a % b = 2

During integer division, if both the operands are of the same sign, the result is truncated towards zero. If one of them is negative, the direction of truncation is implementation dependent. That is,

6/7=0 and -6/-7=0

but -6/7 may be zero -1 (Machine dependent)

Similarly, during modulo division , the sign of the result is always the sign of the first operand(the dividend)

That is

-14 % 3 =-2
-14 % -3= -2
14 % -3=2

Real Arithmetic

An arithmetic operation involving only real operands is called real arithmetic. A real operand may assume values either in decimal or exponential notation. Since floating point values are rounded to the number of significant digits permissible, the final value is an approximation of the correct result. If x, y, and z are floats, then we will have:

x=6.0/7.0=0.857143
y= 1.0/3.0 =0.333333
z= -2.0/3.0= -0.666667

The operator % cannot be used with real operands.

Mixed- mode Arithmetic

When one of the operands is real and the other is integer, the expression is called a mixed-mode arithmetic expression. If either operand is of the real type, then only the real operation is performed and the result is always a real number.

Thus

15/10.0=1.5

where as

15/10=1

Arithmetic operators precedence:-

In a program the value of any expression is calculated by executing one arithmetic operation at a time. The order in which the arithmetic operations are executed in an expression is based on the rules of precedence of operators.

The precedence of operators is :

Unary (-) FIRST
Multiplication(*) SECOND
Division(/) and (%)
Addition(+) and Subtraction(-) LAST

For example, in the integer expression –a *b/c+d the unary- is done first, the result –a is multiplied by b, the product is divided by c(integer division) and d is added to it. The answer is thus:

-ab/c+d

All the expressions are evaluated from left to right. All the unary negations are done first. After completing this the expression is scanned from left to right; now all *, / and % operations are executed in the order of their appearance. Finally all the additions and subtractions are done starting from the left of the expression..

For example, in the expression:

Z=a + b* c

Initially b*c is evaluated and then the resultant is added with a. Suppose if want to add a with b first, then it should be enclosed with parenthesis , is shown below

Z = (a + b) * c

Use of parentheses:

Parentheses are used if the order of operations governed by the precedence rules are to overridden.

In the expression with a single pair of parentheses the expression inside the parentheses is evaluated FIRST. Within the parentheses the evaluation is governed by the precedence rules.

For example, in the expression:

a * b/(c+d * k/m+k)+a

the expression within the parentheses is evaluated first giving:

c+dk/m+k

After this the expression is evaluated from left to right using again the rules of precedence giving

ab/c+dk/m+k +a

If an expression has many pairs of parentheses then the expression in the innermost pair is evaluated first, the next innermost and so on till all parentheses are removed. After this the operator precedence rules are used in evaluating the rest of the expression.

((x * y)+z/(n*p+j)+x)/y+z

xy,np+j will be evaluated first.

In the next scan

Xy+z/np+j +x

Will be evaluated. In the final scan the expression evaluated would be:

(Xy+ z/np+j+x)/y +z

Increment and Decrement operators:-

The increment operator ++ and decrement operator – are unary operators with the same precedence as the unary -, and they all associate from right to left. Both ++ and – can be applied to variables, but no to constants or expressions. They can occur in either prefix or postfix position, with possibly different effects occurring. These are usually used with integer data type.

The general syntax is:

++variable|--variable| variable++| variable—

Some examples are

++count -kk index++ unit_one--

We use the increment and decrement statements in for and while extensively.

Consider the following example

```
m=5;
y=++m;
```

In this case, the value of y and m would be 6. Suppose, if we rewrite the above statements as

```
m=5;
y=m++;
```

then the value of y would be 5 and m would 6. A prefix operator first adds to 1 to the operand and then the result is assigned to the variable on left. On the other hand, a postfix operator first assigns the value to the variable on left and then increments the operand.

Similar is the case, when we use ++(or--) in the subscripted variables. That is, the statement a[i++]=10;

is equivalent to

```
a[i]=10;
i=i+1;
```

The increment and decrement operators can be used in complex statements. Example

```
m=n++ -j+10;
```

Old value of n is used in evaluating the expression. n is incremented after the evaluation.

Relational operators:

We often compare two quantities and depending on their relation, to take certain decisions. For example, we may compare the age of two persons, or the price of two items, and so on. These comparisons can be done with the help of relational operators.

C supports six relational operators in all. These operators and their meanings are shown below

Relational Operators

Operator	Meaning
<	is less than
➢	is greater than
<=	is less than or equal to
>=	is greater than or equal to
= =	is equal to
!=	is not equal to

A simple relational expression contains only one relational operator and has the following form:

ae- 1 relational operator ae-2

ae-1 and ae-2 are arithmetic expressions, which may be simple constants, variables or combination of them.

Given below are some examples of simple relational expressions and their values:

```
4.5<=    10    TRUE
4.5<     10    FALSE
 -35>=    0     FALSE
 10<      7+5   TRUE
```

a+b == c+d TRUE only if the sum of values of a and b is equal to the sum of values of c and d.

When arithmetic expressions are used on either side of a relational operator, the arithmetic expressions will be evaluated first and then the results compared. That is, arithmetic operators have a higher priority over relational operators.

Relational expressions are used in decision statements such as, if and while to decide the course of action of a running program.

Logical operators:

In addition to the relational operators . C has the following three logical operators.

&& logical AND
|| logical OR
! logical NOT

The logical operators && and || are used when we want to test more than one condition and make decisions.
Example:

a>b && x == 10

An expression of this kind which combines two or more relational expression is termed as a logical expression or a compound relational expression. Like the simple relational expressions , a logical expression also yields a value of one or zero, according to the truth table shown below. The logical expression given above is true only if a>b is true and x==10 is true. If either (or both) of them are false, the expression is false.

Truth Table

Op-1	op-2	Value of the expression	
		Op-1 && op-2	op-1 || op-2
Non-zero	Non-zero	1	1
Non-zero	0	0	1
0	Non-zero	0	1
0	0	0	0

Some examples of the usage of logical expressions are:

1. If(age>55 && salary <1000)
2. If (number<0 || number>100)

Relational and logical expressions:

We have seen that float or integer quantities may be connected by relational operators to yield an answer which is true of false.
For example the expression,

Marks>=60

Would have an answer true if marks is greater than or equal to 60 and false if marks is less than 60. The result of the comparison (marks>=60) is called a logical quantity. C provides a facility to combine such logical quantities by logical operators to logical expressions. These logical expressions are useful in translating intricate problem statements.

Example :

A university has the following rules for a student to qualify for a degree with Physics as the main subject and Mathematics as the subsidiary subject:

(i) He should get 50 percent or more in Physics and 40 percent or more in Mathematics.
(ii) If he gets less than 50 percent in Physics he should get 50 percent or more in Mathematics. He should get atleast 40 percent in Physics.
(iii) If he gets less than 40 percent in Mathematics and 60 percent or more in Physics he is allowed to reappear in an examination in Mathematics to qualify.
(iv) In all the other cases he is declared to have failed.

A Decision Table for Examination Results

Chemistry marks	>=50	>=35	>= 60	Else
Physics Marks	>=35	>=50	<35	
Pass	x	x	-	-
Repeat Physics	-	-	x	-
Fail	-	-	-	x

/*This program implements above rules*/

include<stdio.h>

```
main()
{
        unsigned int roll_no, physics_marks, chem_marks;
        while(scanf("%d %d %d", &roll_no,&physics_marks,
&chem_marks)!=EOF)
{
        If(((chem_marks>=50 &&(physics_marks>=35)) ||
((chem_marks>=40)) &&(physics_marks>=50)))

Printf("%d %d %d  Pass\n", roll_no, chem_marks, physics_marks);
```

```
Else

If (( chem_marks>=60)) && physics_marks<35))

Printf("%d %d %d Repeat Physics\n", roll_no,physics_marks,chem_marks);

Else
Printf("%d %d %d Failed \n", roll_no, physics  _marks, chem_marks);
}
}/*End while*/
}/*End main*/
```

Precedence of relational operators and logical operators:

Example:

(a>b *5) &&(x<y+6)

In the above example, the expressions within the parentheses are evaluated first. The arithmetic operations are carried out before the relational operations. Thus b*5 is calculated and after that a is compared with it. Similarly y+6 is evaluated first and then x is compared with it .

In general within parentheses:

1. The unary operations, namely, -,++,--,! (logical not) are performed first.
2. Arithmetic operations are performed next as per their precedence.
3. After that the relational operations in each sub expressions are performed, each sub expression will be a zero or non _ zero. If it is zero it is taken as false else it is taken as true.
4. These logical values are now operated on by the logical operators.
5. Next the logical operation && is performed next.
6. Lastly the evaluated expression is assigned to a variable name as per the assignment operator.

The conditional operators:

An operator called ternary operator pair "?:" is available in C to construct conditional expressions of the form.

exp1? exp2: exp3;

where exp1,exp2, and exp3 are expressions.

The operator ?; works as follows: exp1 is evaluated first. If it is nonzero(true), then the expression exp2 is evaluated and becomes the value of the expression. If exp1 is false, exp3 is evaluated and its value becomes the value of the expression. Note that only one of the expressions(either exp2 or exp3) is evaluated.

For example, consider the following statements

```
x=3;
y=15;
z=(x>y)?x:y;
```

In this example, z will be assigned the value of b. This can be achieved using the if..else statements as follows:

```
If (x>y)
  z=x;
else
  z=b;
```

Bitwise operators:

C has a distinction of supporting special operators known as bitwise operators for manipulation of data at bit level. These operators are used for testing the bits, or shifting them right or left. Bitwise operators may not be applied to float or double.
where the filename is the name containing the required definitions or functions. At this point, the preprocessor inserts the entire contents of the filename into the source code of the program. When the filename is included within the double quotation marks, the search for the file is made first in the directory and then in the standard directories.

Bitwise Operators

Operator	Meaning
&	bitwise AND
!	bitwise OR
^	bitwise exclusive OR
<<	shift left
>>	shift right
~	One's Complement

The Comma Operator

C has some special operators. The comma operator is one among them. This operator can be used to link the related expressions together. A comma-linked list of expressions are evaluated left to right and the value of right-most expression is the value of the combined expression.
For example, the statement

```
Value=(a=2, b=6 ,a+b);
```

First assigns the value 2 to a, then assigns 6 to b, and finally assigns 8(i.e 2+6) to value. The comma operator has the lowest precedence of all operators,hence the parentheses are necessary.

Some examples are given below:

In for loops:

For(a=1, b=10;a<=b; a++, b++)

In while loops:

While(c=getchar(), C!='10')

Exchanging values:

T=x, x=y, y=t;

The precedence of operators among themselves and across all the set of operators:

Each operator in C has a precedence associated with it. This precedence is used to determine how an expression involving more than one operator is evaluated. The operator at the higher level of precedence are evaluated first.

Operator	Description
+	Unary plus
-	Unary minus
++	Increment
--	Decrement
!	Logical negation
~	One's Complement
&	Address
size of(type)	type cast conversion
*	Multiplication
/	Division
%	Modulus
+	Addition
-	Subtraction
<<	left shift
>>	Right shift
<	less than
<=	less than or equal to
>	Greater than
>=	Greater than or equal to
==	Equality
!=	Inequality
&	Bitwise AND
^	Bitwise XOR
|	Bitwise OR
&&	Logical AND
||	Logical OR
?:	Conditional expression
=	Assignment operators

*= /= %=
+= -= &=
^= |=
<<= >>=
Comma operator

The associativity of operators:

The operators of the same precedence are evaluated either from left to right or from right to left depending on the level. This is known as the associativity property of an operator.

The table below shows the associativity of the operators:

Operators	Associativity
() [ ] →	left to right
~ ! –(unary)	left to right
++ -- size of(type)	
&(address)	left to right
*(pointer)	
* / %	
<< >>	left to right
<<= >>=	left to right
== !=	left to right
&	left to right
^	left to right
|	left to right
&&	left to right
||	left to right
?:	right to left
=+ =- *= /= %= &= ^= |= <<= >>=	right to left
,(comma operator)	left to right

Evaluation of expressions involving all the above type of operators:

The following expression includes operators from six different precedence groups.

Consider variables x, y , z as integer variables.

Z+=(x0>0 && x<=10) ? ++x : x/y;

The statement begins by evaluating the complex expression

(x>0 && x<=10)

If this expression is true, the expression ++x is evaluated. Other wise, the a/b is evaluated. Finally, the assignment operation(+=) is carried out, causing the value of c to be increased by the value of the conditional expression.

If for example x, y, and z have the values 1,2,3 respectively, then the value of the conditional expression will be 2(because the expression ++a will be evaluated), and the value of z will increase to 5(z=3+2). On the other hand, if x,y and z have the values 50,10,20 respectively, then the value of the conditional expression will be 5(because the expression x/y will be evaluated) and the value of z will increase to 25(z=20+5).

Mathematical Functions:

In c there are no built-in mathematical functions. The table below shows the mathematical functions.

Function	Meaning
Trigonometric	
1. acos(x)	Arc cosine of x
2. asin(x)	Arc sine of x
3. atan(x)	Arc tan of x
4. atan2(x,y)	Arc tangent of x/y
5. cos(x)	cosine of x
6. sin(x)	sine of x
7. tan(x)	tangent(x)
Hyperbolic	
1. cosh(x)	Hyperbolic cosine of x
2.sinh(x)	Hyperbolic sine of x
3. tanh(x)	Hyperbolic tangent of x
Other functions	
Ceil(x)	x rounded up to the nearest integer
Exp(x)	e to the power x
Fabs(x)	Absolute value of x
Floor(x)	x rounded down to the nearest integer
Fmod(x,y)	Remainder of x/y
Log(x)	Natural log of x, x>0
Log 10 (x)	Base 10 log of x, x>0
Pow(x,y)	x to the power y
Sqrt(x)	Square root of x, x>=0

Note:

1. x and y should be declared as double

2. In trigonometric and hyperbolic functions, x and y are in radians
3. All the functions return a double.

Preprocessor directive:

There are different preprocessor directives. The table below shows the preprocessor directives.

Directive	Function
#define	defines a macro substitution
#undef	Undefines a macro
#include	Specifies the files to be included.
#ifdef	Tests for a macro definition
#endif	Specifies the end of #if
#ifndef	Tests whether a macro is not defined
#if	Tests a compile-time condition
#else	Specifies alternatives when #if tests fails

#**define statement:**

One of the uses of the #define statement is for assigning symbolic names to program constants.

The general syntax is :

```
#define TRUE 1
```

defines the name TRUE and makes it equivalent to the value 1. The name TRUE can subsequently be used anywhere in the program where the constant 1 could be used. Whenever this name appears, its defined value of 1 will be automatically substituted into the program by the preprocessor. For example, we might have the following C statement that uses the defined name TRUE

```
count=TRUE;
```

This statement would assign the value of TRUE to count.

The preprocessor statement

```
#define FALSE  0
```

would define the name FALSE and would make its subsequent use in the program equivalent to specifying the value 0. Therefore, the statement

```
count=FALSE;
```

would assign the value of FALSE to count and the statement

```
If (count==FALSE)
        ……
```

would compare the value of count against the defined value of FALSE.

```
#define TRUE  1
#define  FALSE 0
/*Function to determine if an integer is even*/
```

```
main()
{
   int  number,ans;
printf("enter the number:");
scanf("%d" , &number);
if (number%2==0)
{
    ans=TRUE;
else
    ans=FALSE;
}
```

Some more examples of definition constants:

```
#define COUNT 100
#define SUBJECTS 6
#define PI  3.1415
#define CAPITAL "BANGALORE"
```

#include statement:

C preprocessor enables you to collect all your definitions into a separate file and then include them in your program using the #include statement.

The general statement for this preprocessor directive is:

```
#include "filename"
```

The #include directive can take the form

```
#include<filename>
```

without double quotation marks. In this case, the file searched only in the standard directories.

```
#include  <stdio.h>
#include  "SYNTAX.C"
#include   "STAT.C"
 #include    "TEST.C"
```

Header files:

C language offers simpler way to simplify the use of library functions to the greatest extent possible. This is done by placing the required library function declarations in special source files, called header files. Most C compilers include several header files, each of which contains declarations that are functionally related.

stdio.h is a header file containing declarations for input/ouput routines; math.h contains declarations for certain mathematical functions and so on. The header files also contain other information related to the use of the library functions, such as symbolic constant definitions.

The required header files must be merged with the source program during the compilation process. This is accomplished by placing one or more #include statements at the beginning of the source program.

The other header files are:

<ctype.h>	character testing and conversion functions
<stdlib.h>	utility functions such as string conversion routines , memory allocation routines, random number generator, etc
<string.h>	String manipulations functions
<time.h>	time manipulation functions

Control Statements

Introduction :

So far we have studied the programs , where the instructions are executed sequentially. It means statements are executed in the same order in which they appear in the program. But there are many times, we want the instructions to be executed in one situation. The 'c' language was capable of performing different sets of actions depending on the circumstances. The C has three major decision making instructions, the **if** statement , the **if-else** statement and the **switch** statement.

General form of if statement :

The general form of if statement looks like

```
if ( condition)
  statement;
```

The keyword if tells the compiler if the condition is true, then statement should be executed. Suppose if the condition is false, then statement has to be skipped. The following table shows the relation operator used in condition.

==	equal
!=	Not equal to
>	Greater than
<	Less than
<=	Less than or equal to
>=	Greater than or equal to

The three logical operators used are

&&	and
||	or
!	not

&& operator return true when all conditions are evaluated to true. Otherwise it return false. For example

((5 > 4) && (2 < 3)) return true.

|| operator return true when one of the condition returns true, otherwise it returns false. For example

((5 > 4) || (2 > 4)) return true

! operator is used to negate the result. For example

(!(5>4)) return false.

Let us see some of the example.

Problem : Given two integer numbers, find out the larger of them.

```
#include <stdio.h>
main()
{
  float  x, y;
printf(" input two numbers\n");
scanf("%f %f ", &x, &y);
if ( x > y)
printf(" x is big\n");
if (y > x)
printf("y is big\n");
}
```

Problem : Find out a number is even or odd.

```
#include <stdio.h>
main()
{
 int x;
printf("input an integer\n");
scanf("%d",&x);
if ((x % 2) == 0)
printf("it is even number\n");
if ((x%2) == 1)
printf("it is odd number\n");
}
```

Problem : Imagine there are n cricket palyers. Find out how many cricket team can be formed and how many players left out.

```
#include <stdio.h>
main()
{
  int n,teams,leftout;
printf("input how many members\n");
scanf("%d",&n);
teams = n/11;
leftout=n %11;
printf("number of teams can be formed = %d\n", teams);
```

```
printf("number of members are left out = %d\n", leftout);
}
```

Multiple statements within if

It may so happen that in a program we want more than one statement to be executed if the condition following if is satisfied. If such multiple statements are to be executed then they must be placed within a pair of braces.

Let us consider an example of finding out the roots of the quadratic equation.

```
#include <stdio.h>
main()
{
  float a,b,c,d,x1,x2, xr1,xi1,xr2,xi2;

printf("input the coefficients of a b and c\n");
scanf("%f %f %f", &a, &b, &c);

if ( a == 0)
{
  printf(" it is linear equation\n");
    x1 = -b/c;
printf("root is %f\n", x1);
exit(0);
}

d = b*b – 4.0 * a * c;

if  ( d == 0)
{
  printf(" equal roots\n");
   x1 = -b/ (2.0*a);
   x2 = -b/(2.0*a);
printf(" %f %f\n", x1,x2);
exit(0);
}

if ( d >0)
{
  printf(" real and unequal roots\n");
  x1 = (-b + sqrt(d)) / (2.0*a);
  x2 = (-b – sqrt(d))/(2.0*a);
  printf(" %f %f\n", x1,x2);
  exit(0);
```

```
}

if ( d < 0)
{
  printf( " complex roots\n");
  d= -d;
   xr1 = -b/(2.0*a);
   xi1 =  sqrt(d)/(2.0*a);
   xr2= -b/(2.0*a);
   xi2 = sqrt(d)/(2.0*a)
printf(" first root is %f + %f\n", xr1, xi1);
printf("second root is %f - %f\n", xr2,xi2);
exit(0);
}
}
```

In the above program exit(0) is used to quit from the program. This is because once the condition is fulfilled, there is no necessity to continue the program.

The if-else statement

The if statement by itself will execute a single statement or a group of statements, when the condition following if is true. It does nothing when it is false. The if-else statement provides the option a statement/s can be executed when the condition is false.

The syntax is

```
if (condition)
 statement1;
else
statement2;
```

If the condition is true , the statement1 is executed. If the condition is false, then statement2 under else part is get executed.

Let us consider an example.

Problem : : Find out a number is even or odd

```
#include <stdio.h>
main()
{
 int x;
printf("input an integer\n");
scanf("%d",&x);
```

```
if ((x % 2) == 0)
printf("it is even number\n");
else
printf("it is odd number\n");
}
```

Switch statement

This is another form of the multi way decision. It is well structured, but can only be used in certain cases where;

- Only one variable is tested, all branches must depend on the value of that variable. The variable must be an integral type. (int, long, short or char).
- Each possible value of the variable can control a single branch. A final, catch all, default branch may optionally be used to trap all unspecified cases.

The C `switch` allows multiple choice of a selection of items at one level of a conditional where it is a far neater way of writing multiple `if` statements:

```
switch (expression) {
                        case item1:
                                    statement1;
                                    break;
                        case item2:
                                    statement2;
                                    break;
                                    ⋮
                    ⋮                       case itemn:
                                    statementn;
                                    break;
                        default:
                                    statement;
                                    break;
                    }
```

In each case the value of $item_i$ must be a constant, variables are <u>not</u> allowed.

The `break` is needed if you want to terminate the `switch` after execution of one choice. Otherwise the next case would get evaluated.

The `default` case is optional and catches any other cases.

For example:-

```
switch (letter)
                        {
                         case `A':
                         case `E':
                         case `I':
                         case `O':
                         case `U':
                                        numberofvowels++;
                                        break;

                         case ` ':
                                        numberofspaces++;
                                        break;

                         default:
                                        numberofconstants++;
                                        break;
                        }
```

In the above example if the value of `letter is `A', `E', `I', `O' or `U' then numberofvowels is incremented.`

`If the value of letter is ` ' then numberofspaces is incremented.`

`If none of these is true then the default condition is executed, that is numberofconstants is incremented.`

The ? operator

The `?` (***ternary condition***) operator is a more efficient form for expressing simple `if` statements. It has the following form:

expression$_1$ `?` *expression*$_2$`:` *expression*$_3$

It simply states:

`if` *expression*$_1$ `then` *expression*$_2$ `else` *expression*$_3$

For example to assign the maximum of `a` and `b` to `z`:

```
z = (a>b) ? a : b;
```

which is the same as:

```
if (a>b)
                        z = a;
        else
                        z=b;
```

problem : Find out larger of two numbers using ternary operator

```
#include <stdio.h>
main()
{
  int a,b,max;
printf("input two integer numbers\n");
scanf("%d %d", &a,&b);
max = (a > b) ? a: b;
printf("maximum number is equal to = %d\n", max);
}
```

Loops

Introduction :

When we write a list of instructions for someone to perform we usually expect them to follow the list from the top to the bottom, *one at a time*. This is the simple default flow of control through a program. The C programs we have written so far use this one-after-another default flow of control.

This is fine and simple, but it limits the running time of any program we can write. There is a limit to the number of instructions we can write and it does not take long for a computer to read though and obey our list. Whenever we want to execute a group of instruction many times, the concept called loop it comes into the picture.

Suppose we want to display "computer" six times on the screen. We may write a program that is given below

```
#include <stdio.h>
main()
 {
   printf("computer\n");
   printf("computer\n");
   printf(“computer\n”);
   printf("computer\n");
   printf("computer\n");
   printf("computer\n");
 }
```

But this becomes complicated when we want to print 100 times “computer” messages.

What you really need is some way of repeating the **`printf`** statements without having to write it out each time. The solution to this problem is to use loop statements.

Loop statement in c :

C language provides three types of loop, while, do while and for.

- The while loop keeps repeating an action until an associated test returns false. This is useful where the programmer does not know in advance how many times the loop will be traversed.

- The do while loops is similar, but the test occurs after the loop body is executed. This ensures that the loop body is run at least once.
- The for loop is frequently used, usually where the loop will be traversed a fixed number of times. It is very flexible, and novice programmers should take care not to abuse the power it offers.

While statement

We use a while statement to continually execute a block of statements while a condition remains `true`. The following is the general syntax of the `while` statement.

```
while (expression) {
    statement
}
```

First, the `while` statement evaluates *expression*, which must return a boolean value. If the expression returns `true`, the `while` statement executes the statement(s) in the `while` block. The `while` statement continues testing the expression and executing its block until the expression returns `false`.

Let us consider a simple example, to calculate the factorial of a number

```
#include <stdio.h>
main()
{
  int fact, i,n;
printf("input a number\n");
scanf("%d",&n);
fact=1;
i=1;
while ( i <= n)
{
  fact=fact*i;
  ++i;
}
printf(" factorial = %d\n", fact);
}
```

do –while statement

Syntax:

```
do {
```

```
    statement(s)
} while (expression);
```

Instead of evaluating the expression at the top of the loop, `do-while` evaluates the expression at the bottom. Thus, the statements within the block associated with a `do-while` are executed at least once.

Each line of a C program up to the semicolon is called a statement. The semicolon is the statement's terminator. The braces { and } which have appeared at the beginning and end of our program unit can also be used to group together related declarations and statements into a ***compound statement*** or a *block*.

In the case of the **while** loop before the ***compound statement*** is carried out the ***condition*** is checked, and if it is *true* the ***statement*** is obeyed one more time. If the ***condition*** turns out to be *false*, the looping isn't obeyed and the program moves on to the next statement. So we can see that the instruction really means *while something or other is true keep on doing the statement*.

In the case of the **do while** loop it will always execute the code within the loop at least once, since the ***condition*** controlling the loop is tested at the bottom of the loop. The **do while** loop repeats the instruction while the ***condition*** is *true*. If the ***condition*** turns out to be *false*, the looping isn't obeyed and the program moves on to the next statement.

Let us consider an example to calculate the factorial using do-while loop.

```
#include < stdio.h>
main()
{
  int n, fact, i;

printf("input an integer\n");
scanf("%d",&n);
fact=1;
i=1;
do
 {
   fact=fact*i;
   ++i;
  }
   while ( i <= n);

printf("factorial = %d\n", fact);
```

}

for statement:

The for statement provides a compact way to iterate over a range of values. The general form of the `for` statement can be expressed as follows.

```
for (initialization; termination; increment) {
    statement(s)
}
```

The *initialization* expression initializes the loop — it's executed once at the beginning of the loop. The *termination* expression determines when to terminate the loop. When the expression evaluates to `false`, the loop terminates. Finally, *increment* is an expression that gets invoked after each iteration through the loop. All these components are optional. In fact, to write an infinite loop, you omit all three expressions.

```
for ( ; ; ) {    //infinite loop
    ...
}
```

Let us consider an example to calculate the factorial of a number using the for loop.

```
#include <stdio.h>
main()
{
 int n,fact,i;
printf("enter an integer\n");
scanf("%d",&n);
fact=1;
for (i=1; i<=n; ++i)
fact=fact*i;
printf("factorial = %d\n", fact);
}
```

Arrays:

The meaning of an array:
A group of related data items that share a common name is called an array. For example, we can define an array name marks to represent a set of marks obtained by a group of students. A particular value is indicated by writing a number called index number or subscript in brackets after the array name.
Example,

Marks [7]

Represents the marks of the 7th student. The complete set of values is referred to as an array, the individual values are called elements. The arrays can be of any variable type.

One-dimensional array:

When a list of items can be given one variable name using only one subscript and such a variable is called a single-subscripted variable or one-dimensional array.

In C language, single-subscripted variable xi can be represented as

x[1],x[2],x[3]……x[n]

The subscripted variable xi refers to the ith element of x. The subscript can begin with number 0. For example, if we want to represent a set of five numbers say (57,20,56,17,23), by a array variable num, then we may declare num as follows

int num[5];

And the computer reserves five storage locations as shown below:

Num[0]
Num[1]
Num[2]
Num[3]
Num[4]

The values can be assigned as follows:

```
Num[0]=57;
Num[1]=20;
Num[2]=56;
Num[3]=17;
Num[4]=23;
```

The table below shows the values that are stored in the particular numbers.

Num[0]	57
Num[1]	20
Num[2]	56
Num[3]	17
Num[4]	23

Two-dimensional arrays:

There are certain situations where a table of values will have to be stored. C allows us to define such table using two-dimensional arrays.

Two-dimensional arrays are declared as follows:

Type array_name [row size][column size]

In c language the array sizes are separated by its own set of brackets.

Two-dimensional arrays are stored in memory as shown in the table below. Each dimension of the array is indexed from zero to its maximum size minus one; the first index selects the row and the second index selects the column within that row.

	Column0	Column1	Column2
	[0][0]	[0][1]	[0][2]
Row 0	210	340	560
	[1][0]	[1][1]	[1][2]
Row 1	380	290	321
	[2][0]	[2][1]	[2][2]
Row2	490	235	240
Row3	[3][0]	[3][1]	[3][2]

240 350 480

Declaration and initialization of arrays:

The arrays are declared before they are used in the program. The general form of array declaration is

type variable_name[size];

The type specifies the type of element that will be contained in the array, such as int, float, or char and the size indicates the maximum number of elements that can be stored inside the array.

Example:

float weight[40]

Declares the weight to be an array containing 40 real elements. Any subscripts 0 to 39 are valid.

Similarly,

int group1[11];

Declares the group1 as an array to contain a maximum of 10 integer constants.

The C language treats character strings simply as arrays of characters. The size in a character string represents the maximum number of characters that the string can hold.

For example:

char text[10];

Suppose we read the following string constant into the string variable text.

"HOW ARE YOU"

Each character of the string is treated as an element of the array text and is stored in the memory as follows.

'H'
'O'
'W'
'A'
'R'
'E'
'Y'
'O'
'U'
'\o'

When the compiler sees a character string, it terminates it with an additional null character. Thus, the element text [11] holds the null character '\o' at the end. When declaring character arrays, we must always allow one extra element space for the null terminator.

Initialization of arrays:

The general form of initialization of arrays is:

static type array-name[size]={ list of values};

The values in the list are separated by commas.
For example, the statement below shows

static int num[3]={2,2,2};

Will declare the variable num as an array of size 3 and will assign two to each element. If the number of values is less than the number of elements, then only that many elements will be initialized. The remaining elements will be set to zero automatically.
For example:

static float num1[5]={0.1,2.3,4.5};

Will initialize the first three elements to 0.1,2.3 and 4.5 and the remaining two elements to zero. The word static used before type declaration declares the variable as a static variable.
In some cases the size may be omitted. In such cases, the compiler allocates enough space for all initialized elements. For example, the statement

static int count[ ]= {2,2,2,2};

Will declare the counter array to contain four elements with initial values 2.

Character arrays may be initialized in a similar manner. Thus, the statement

static char name[ ]={ 'S 'W,'A,'N}

Declares the name to be an array of four characters, initialized with the string "SWAN"

There certain draw backs in initialization of arrays.

1. There is no convenient way to initialize only selected elements.
2. There is no shortcut method for initializing a large number of array elements.

Reading Writing and manipulation of above types of arrays.

Program to read and write two-dimensional arrays.

```
#include<stdio.h>
main()
{
        int a[10][10];
         int i, j row,col;
        printf("\n Input row and column  of a matrix:");
        scanf("%d %d",  &row,&col);
        for(i=0; i<row;i++)
           for(j=0;j<col;j++)
```

```
            scanf("%d", &a[i][j]);
for(i=0;i<row;i++)
{
        for(j=0;j<col;j++)
        printf("%5d", a[i][j]);
printf("\n");
}
```

Program showing one-dimensional array

```
main()

{
        int i;
        float a[10],value1,total;
  printf("Enter 10 Real numbers\n");
  for(i=0;i<10;i++)
{
        scanf("%f", &value);
        x[i]=value1;
}

total=0.0;
for(i=0;i<10;i++)
total=total+a[i]*a[i];

printf("\n");
for(i=0;i<10;i++)
printf("x[%2d]= %5.2f\n", i+1, x[i]);
printf("\ntotal=%.2f\n", total);
}
```

Programming examples:

Program to print multiplication tables

```
#define R1   4
#define  C1   4

main()
{
        int row,col,prod[R1][C1];
        int i,j;
        printf(" MULTIPLICATION TABLE \n\n");
        printf("   ");
```

```
        for(j=1;j<=C1;j++)
          printf("%4d",j);
          printf("\n");
           printf("-----------------------------------------\n");
          for(i=0;i<R1;i++)
           {
                row=i+1;
                printf("%2d|", R1);
          for(j=1;j<=C1;j++)
            {
                col=j;
                prod[i][j]=row*col;
                printf("%4d", prod[i][j]);
             }
                printf("\n");
          }
        }
```

Output

```
        MULTIPLICATION TABLE
        1  2   3   4
  ---------------------------------------
1   |    1   2   3    4
2   |    2   4   6    8
3   |    3    6   9   12
4   |    4    8   12  16
```

1) Write a C program to convert a binary number to decimal number using one-dimensional array.

```
        #include<stdio.h>
        #include<math.h>
    main()
        {
            int b[10],sum=0,i,dig;
printf("\n enter the number of digits (MAX 8): \t");
scanf("%d", &dig);
printf("\n Enter the binary digits (Left to Right): \t");
for(i=1;i<=dig;i++)
```

```
        scanf("%d", &bin[i]);
for(i=dig;i<=1;i--)
sum+=bin[i]*pow(2,dig-1);
printf("\n the decimal number is= \t %d n", sum);
}
```

2) Program to calculate the total cost of tubes

```
#include<stdio.h>

main()
{
        int st[5], watt;
        float cost[5], total;
        total=0;

for(watt=0;watt<=4;++watt)
  {
        scanf("%d %f", &st[watt], &cost[watt])
        {
                scanf("%d  %f",  &st[watt], &cost[watt]);
                total+= (float) st[watt]*cost[watt];
        }
printf("%f \n", total);
}
```

3) Program to show swapping of numbers.

```
#include<stdio.h>

main()
{
        int x[10], i,j,temp;
        for(i=0;i<=9;++i)
        scanf("%d", &x[i]);
for(j=0;j<=8;j+=2)
    {
        temp=x[j];
        x[j]=x[j+1];
   x[j+1]=temp;
}

for(i=0;i<=9;++i)
printf("%d", x[i]);
printf("\n");
}
```

4) Program to sort a list of numbers:

```
#define N 10
        main()
{
        int i,j,k;
float a[N],t;
printf("Enter the number of items\n");
scanf("%d", &n);
printf("Input %d values \n", n);
for(i=1; i<=n ;;i++)
  scanf("%f", &a[i]);
for(i=1;i<=n-1;i++)
{
   for(j=1;j<=n-i; j++)
        {
           if)a[j]<=a[j+1])
{
        t=a[j];
        a[j]=a[j+1];
        a[j+1]=t;
}
else
    continue;
}
}
for(i=1;i<=n;i++)
 printf("%f", a[i]);
}
```

5) Program to calculate standard deviation

```
#include<math.h>
#define MAX 100

main()
{
        int i,n;
float val[MAX],deviation,sum,ssqr,mean,var,stddev;
sum=ssqr=n=0;
printf("Input values: input-1 to end\n");
for(i=1;i<MAX;i++)
{
        scanf("%f", &val[i]);
        if(val[i]==-1)
          break;
```

```
        sum+=val[i];
        n+=1;
}

mean=sum/(float)n;
for(i=1;i<=n;i++)
{
        deviation=val[i]-mean;
        ssqr+=deviation*deviation;
}
 var=ssqr/(float)n;
stddev=sqrt(var);
printf("\n Number of items:%d\n",n);
printf("Mean: %f \n", mean);
printf("Standard deviation: %f\n", stddev);
}
```

6) Program to find the largest and smallest of numbers:

```
#include<stdio.h>
main()
{
        int I,s, largcount,smcount;
        float num[30],lar,small;

printf("\n size of array (MAX 30): \t");
scanf("%d", &size);

printf("\n Array elements:\t");

for(i=0;i<size;i++)
        scanf("%f", &num[i]);

for(i=0;i<size;i++)
        printf("%f", &num[i]);

lar=small=num[0];
larcount=smcount=1;
for(i=1;i<size;i++)
{
        if(num[i]>lar)
{
        lar=num[i];
        larcount=i+1;
}

elseif(num[i]<small)
```

```
{
        small=num[i];
        smcount=i+1;
}
}
printf("\n Largest value is % f found at %d", lar,larcount);
printf("\n Smallest value is  %f found at %d ", small, smcount);
}
```

Arrays:

The meaning of an array:

A group of related data items that share a common name is called an array. For example, we can define an array name marks to represent a set of marks obtained by a group of students. A particular value is indicated by writing a number called index number or subscript in brackets after the array name.
Example,

Marks[7]

Represents the marks of the 7th student. The complete set of values is referred to as an array, the individual values are called elements. The arrays can be of any variable type.

One-dimensional array:

When a list of items can be given one variable name using only one subscript and such a variable is called a single-subscripted variable or one dimensional array.

In C language ,single-subscripted variable xi can be represented as

x[1],x[2],x[3]……x[n]

The subscripted variable xi refers to the ith element of x. The subscript can begin with number 0. For example, if we want to represent a set of five numbers, say (57,20,56,17,23), by a array variable num, then we may declare num as follows

Int num[5];

And the computer reserves five storage locations as shown below:

Num[0]
Num[1]
Num[2]
Num[3]
Num[4]

The values can be assigned as follows:

Num[0]=57;
Num[1]=20;
Num[2]=56;
Num[3]=17;
Num[4]=23;

The table below shows the values that are stored in the particular numbers.
Num[0]

Num[1]
Num[2]
Num[3]
Num[4]

57
20
56
17
23

Two dimensional arrays:

There are certain situations where a table of values will have to be stored. C allows us to define such table using two dimensional arrays.

Two dimensional arrays are declared as follows:

Type array_name [row_size][column_size]

In c language the array sizes are separated by its own set of brackets.

Two dimensional arrays are stored in memory as shown in the table below. Each dimension of the array is indexed from zero to its maximum size minus one; the first index selects the row and the second index selects the column within that row.

	Column0	Column1	Column2
	[0][0]	[0][1]	[0][2]
Row 0	210	340	560
	[1][0]	[1][1]	[1][2]
Row 1	380	290	321
	[2][0]	[2][1]	[2][2]
Row2	490	235	240
Row3	[3][0]	[3][1]	[3][2]
	240	350	480

Declaration and initialization of arrays:

The arrays are declared before they are used in the program. The general form of array declaration is

Type variable_name[size];

The type specifies the type of element that will be contained in the array, such as int,float,or char and the size indicates the maximum number of elements that can be stored inside the array.

Example:

Float weight[40]

Declares the weight to be an array containing 40 real elements. Any subscripts 0 to 39 are valid.

Similarly,

Int group1[11];

Decalres the group1 as an array to contain a maximum of 10 integer constants.

The C language treats character strings simply as arrays of characters. The size in a character string represents the maximum number of characters that the string can hold.

For example:

Char text[10];

Suppose we read the following string constant into the string variable text.

"HOW ARE YOU"

Each character of the string is treated as an element of the array text and is stored in the memory as follows.

'H'
'O'
'W'
'A'
'R'
'E'
'Y'
'O'
'U'
'\o'

When the compiler sees a character string, it terminates it with an additional null character. Thus, the element text[11] holds the null character '\o' at the end. When declaring character arrays, we must always allow one extra element space for the null terminator.

Initialization of arrays:

The general form of initialization of arrays is:

Static type array-name[size]={ list of values};

The values in the list are separated by commas.
For example, the statement below shows

Static int num[3]={2,2,2};

Will declare the variable num as an array of size 3 and will assign two to each element. If the number of values is less than the number of elements, then only that many elements will be initialized. The remaining elements will be set to zero automatically.
For example:

Static float num1[5]={0.1,2.3,4.5};

Will initialize the first three elements to 0.1,2.3 and 4.5 and the remaining two elements to zero. The word static used before type declaration declares the variable as a static variable.
In some cases the size may be omitted. In such cases, the compiler allocates enough space for all initialized elements. For example, the statement

Static int count[ ]= {2,2,2,2};

Will declare the counter array to contain four elements with initial values 2.

Character arrays may be initialized in a similar manner. Thus, the statement

Static char name[ ]={ 'S 'W,'A,'N}

Declares the name to be an array of four characters, initialized with the string "SWAN"

There certain draw backs in initialization of arrays.

3. There is no convenient way to initialize only selected elements.
4. There is no shortcut method for initializing a large number of array elements.

Reading Writing and manipulation of above types of arrays.

Program to read and write two dimensional arrays.

```
#include<stdio.h>
main()
{
        int a[10][10];
         int i, j row,col;
        printf("\n Input row and column  of a matrix:");
        scanf("%d %d",  &row,&col);
        for(i=0; i<row;i++)
           for(j=0;j<col;j++)
          scanf("%d", &a[i][j]);
```

```
for(i=0;i<row;i++)
{
        for(j=0;j<col;j++)
        printf("%5d", a[i][j]);
printf("\n");
}
```

Program showing one-dimensional array

```
main()

{
        int i;
        float a[10],value1,total;
  printf("Enter 10 Real numbers\n");
  for(i=0;i<10;i++)
{
        scanf("%f", &value);
        x[i]=value1;
}

total=0.0;
for(i=0;i<10;i++)
total=total+a[i]*a[i];

printf("\n");
for(i=0;i<10;i++)
printf("x[%2d]= %5.2f\n", i+1, x[i]);
printf("\ntotal=%.2f\n", total);
}
```

Programming examples:

Program to print multiplication tables

```
#define R1   4
#define  C1   4

main()
{
        int row,col,prod[R1][C1];
        int i,j;
        printf(" MULTIPLICATION TABLE \n\n");
        printf("   ");

        for(j=1;j<=C1;j++)
```

```
            printf("%4d",j);
            printf("\n");
             printf("-----------------------------------------\n");
            for(i=0;i<R1;i++)
             {
                    row=i+1;
                    printf("%2d|", R1);
            for(j=1;j<=C1;j++)
              {
                    col=j;
                    prod[i][j]=row*col;
                    printf("%4d", prod[i][j]);
               }
                    printf("\n");
             }
           }
```

Output

```
        MULTIPLICATION TABLE
        1  2   3   4
  ---------------------------------------
1   |    1   2   3    4
2   |    2   4   6    8
3   |    3    6   9   12
4   |    4    8   12  16
```

1) Write a C program to convert a binary number to decimal number using one-dimensional array.

```
        #include<stdio.h>
        #include<math.h>
    main()
        {
            int b[10],sum=0,i,dig;
printf("\n enter the number of digits (MAX 8): \t");
scanf("%d", &dig);
printf("\n Enter the binary digits (Left to Right): \t");
for(i=1;i<=dig;i++)
        scanf("%d", &bin[i]);
```

```
for(i=dig;i<=1;i--)
sum+=bin[i]*pow(2,dig-1);
printf("\n the decimal number is= \t %d n", sum);
}
```

6) Program to calculate the total cost of tubes

```
#include<stdio.h>

main()
{
        int st[5], watt;
        float cost[5], total;
        total=0;

for(watt=0;watt<=4;++watt)
  {
        scanf("%d %f", &st[watt], &cost[watt])
        {
                scanf("%d  %f",  &st[watt], &cost[watt]);
                total+= (float) st[watt]*cost[watt];
        }
printf("%f \n", total);
}
```

7) Program to show swapping of numbers.

```
#include<stdio.h>

main()
{
        int x[10], i, j, temp;
        for(i=0;i<=9;++i)
        scanf("%d", &x[i]);
for(j=0;j<=8;j+=2)
    {
        temp=x[j];
        x[j]=x[j+1];
   x[j+1]=temp;
}

for(i=0;i<=9;++i)
printf("%d", x[i]);
printf("\n");
}
```

8) Program to sort a list of numbers:

```
#define N 10
        main()
{
        int i,j,k;
float a[N],t;
printf("Enter the number of items\n");
scanf("%d", &n);
printf("Input %d values \n", n);
for(i=1; i<=n ;;i++)
  scanf("%f", &a[i]);
for(i=1;i<=n-1;i++)
{
   for(j=1;j<=n-i; j++)
        {
           if)a[j]<=a[j+1])
{
        t=a[j];
        a[j]=a[j+1];
        a[j+1]=t;
}
else
    continue;
}
}
for(i=1;i<=n;i++)
 printf("%f", a[i]);
}
```

9) Program to calculate standard deviation

```
#include<math.h>
#define MAX 100

main()
{
        int i,n;
float val[MAX],deviation,sum,ssqr,mean,var,stddev;
sum=ssqr=n=0;
printf("Input values: input-1 to end\n");
for(i=1;i<MAX;i++)
{
        scanf("%f", &val[i]);
        if(val[i]==-1)
          break;
        sum+=val[i];
```

```
        n+=1;
}

mean=sum/(float)n;
for(i=1;i<=n;i++)
{
        deviation=val[i]-mean;
        ssqr+=deviation*deviation;
}
 var=ssqr/(float)n;
stddev=sqrt(var);
printf("\n Number of items:%d\n",n);
printf("Mean: %f \n", mean);
printf("Standard deviation: %f\n", stddev);
}
```

6) Program to find the largest and smallest of numbers:

```
#include<stdio.h>
main()
{
        int I,s, largcount,smcount;
        float num[30],lar,small;

printf("\n size of array (MAX 30): \t");
scanf("%d", &size);

printf("\n Array elements:\t");

for(i=0;i<size;i++)
        scanf("%f", &num[i]);

for(i=0;i<size;i++)
        printf("%f", &num[i]);

lar=small=num[0];
larcount=smcount=1;
for(i=1;i<size;i++)
{
        if(num[i]>lar)
{
        lar=num[i];
        larcount=i+1;
}

elseif(num[i]<small)
{
```

```
        small=num[i];
        smcount=i+1;
}
}
printf("\n Largest value is % f found at %d", lar,larcount);
printf("\n Smallest value is  %f found at %d ", small, smcount);
}
```

Functions:

Need for user-defined functions:

The function main() is a specially recognized function in C. Every program must have a main function to indicate where the program has to begin its execution. It is possible to code any program utilizing only main function, it leads to a number of problems. The program may become too large and complex and as a result the task of debugging, testing and maintaining becomes difficult. If a program is divided into functional parts, then each part may be independently coded and later combined into a

single unit. These subprograms called 'functions' are much easier to understand, debug, and test. Sometimes it is also called "divide and conquer ".

There are times when some type of operation or calculation is repeated at many points throughout a program. In such situations, we may repeat the program statements wherever they are needed. There is another way to design a function that can be called and used whenever required. This saves both time and space.

This approach clearly results in a number of advantages.

1. The figure below shows to-down modular programming. In this programming style, the high level logic of the overall problem is solved first while the details of each lower-level function are addressed later.
2. The length of a source program can be reduced by using functions at appropriate places.
3. It is easy to locate and isolate a faulty function for further investigations.
4. A function may be used by many other programs.

Main Program

Function 1 Function2 Function3 Function4 Function5

Top-down modular programming using functions

Defining and using functions:

All the functions have the form:

```
Function-name(argument list)
{
        local variable declarations;
        executable statement1;
        executable statement2;
        -------------------------;
        -------------------------          ;
        return(expression);
}
```

All parts are not essential. Some may be absent. For example, the argument list and its associated argument declaration parts are optional. The declaration of local variables is required only when any local variables are used in the function. A function can have any number of executable statements. A function that does nothing, may not include any executable statements at all.

```
Do_nothing() {}
```

The return statement is the mechanism for returning a value to the calling function. This is also optional statement. Its absence indicates that no value is being returned to the calling function.

Function name:

A function must follow the same rules of formation as other variable names in C.

Argument List:

The argument list contains valid variable names separated by commas. The list must be surrounded by parentheses. The argument variables receive values from the calling function, thus providing a means for data communication from the calling function to the called function. Some examples of functions with arguments are:

```
Quadratic(a,b,c)
Power(x,n)
Mul(a,b)
```

All the argument variables must be declared for their types after the function header and before the opening brace of the function body.

```
Power(x,n)
float x;
int n;
{
  ---------
  ---------
   --------
}
```

Category of functions:

A function may depend on whether arguments are present or not and whether a value is returned or not. It may belong to one of the following categories.

Category 1: Functions with no arguments and no return values.
Category 2: Functions with arguments and no return values.
Category 3: Functions with arguments and return values.

No arguments and no return values:

When a function has no arguments, it does not receive any data from the calling function. Similarly, when it does not return a value, the calling function does not receive any data from the called function. In effect, there is not data transfer between the calling function and the called function. The dotted lines indicate that there is only a transfer of control but not data.

Function1() No input function2()

```
        {                                       {
         ----------------
        function2()                               ----------
         ----------------                         ----------
         ----------------                         ----------
                          }                     }
```

A program with three user-defined functions is given below.

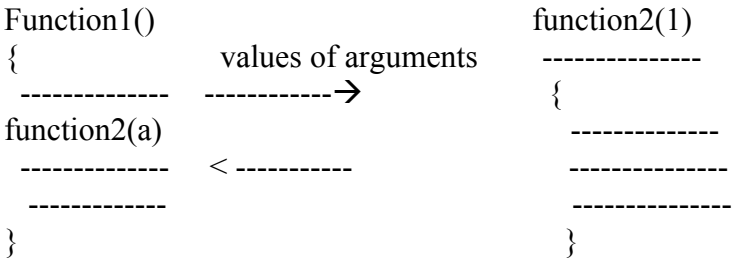
Functions with no arguments, no return values:

```
main()
     {
       printtext();
        value();
        printtext();
  }
printtext()
{
     int I;
     for (i=1;i<=40;i++)
      printf("%c", '-');
      printf("\n");
}

value()
{
     int year, period;
     float inrate, sum principal;
     printf("Principal amount?");
     scanf("%f", &principal);
     printf("Interest rate?   ");
     scanf("%f", &inrate);
     printf("Period?            ");
     scanf("%d",   &period);
       sum=principal;
       year=1;
      while(year<=period)
       {
              sum=sum * (1+inrate);
              year=year+1;
        }
printf(" \n 8.2f %5.2f   %5d  %12.2f\n", principal,inrate,period,sum);
}
```

Functions with arguments and no return values:

We can make the calling function to read data from the terminal and pass it on to the called function. The nature of data communication between the calling function and the called function with arguments but no return values is shown below.



For example.

```
Printtext(ch)
Value(p,r,n)
```

The arguments ch,p,r and n are called the formal arguments. The calling function can now send values to these arguments using function calls containing appropriate arguments. For example, the function call

Value(100,0.23,10)

Would send the values 100,0.23,and 10 to the function.

Value(p,r,n)

And assign 100 to p, 0.23 ,to r, and 10 n. The values 100,0.23 and 10 are the actual arguments which become the values of the formal arguments inside the called function.

The actual and formal functions should match in number, type and order. The values of actual arguments are assigned to the formal arguments on a one to one basis starting with the first argument.

```
main()
{
Function
call
-----→      function1(a1,a2,a3……am)
            ------------
}
Called function
------ >      function1(f1,f2,f3,……fn)
            ----------
            ----------
            {
                ----------
                ----------
             }
```

The formal arguments must be valid variable names, the actual arguments may be variable names, expressions or constants. The variables used in actual arguments must be assigned values before the function call is made.

When a function call is made, only a copy of the values of actual arguments is passed into the called function.

The function call value(prin, rate, per);

Passes information it contains to the function value.

The function header of value has three formal arguments p,r and n which correspond to the actual arguments in the function call, namely, prin,rate and per. The formal arguments are declared immediately after the function header. On execution of the function call, the values of the actual arguments are assigned to the corresponding formal arguments.

```
p=prin;
r=rate;
n=per;
```

The variable declared inside a function are known as local variables and therefore their values are local to the function and cannot be accessed by any other function.

The function value calculates the final amount for a given period and prints the result. Control is transferred back on reaching the closing brace of the function. Note that the function does not return any value.

Program to show functions with arguments but no return values

```
Main()
  {
        float prin.rate,amt;
        int per;
        printf("Enter principal amount, interest");
        printf("rate and period\n");
        scanf("%f %f %d", &prin,&rate,&per);
        printtext('Z');
        value(prin,rate,per);
        printtext('A');
}

  printtext(ch)
   char ch;
{
     int j;
     for(j=1;j<=52;j++)
      printf("%c", ch);
       printf("\n");
}

value(p,r,n)
 int n;
  float p,r;
```

```
        {
          int year;
           float sum;
           sum=p;
           year=1;
           while(year<=n)
               {
                 sum=sum*(1+r)
                 year=year+1;
               }
    printf("%f %f %d  %f \n", p, r, n, sum);
}
```

Arguments with return values:

We may not always wish to have the result of a function displayed. We may use it in the calling function for further processing. Moreover , to assure a high degree or portability between programs, function should generally be coded without involving any I/O operations. Different programs may require different output Formats for displaying results. This can be overcome by handing over the result of a function to its calling function where the returned value can be required by the program.

```
Function1()                                  function2(f)
{                        values of
                            arguments            ------------
  -----------          ---------------→          {
  -----------                                       -------------
                            Function result         ------------
   function2(a)       <-----------------            ------------
    -------------                                    return(e)
     -----------                                     }
}
```

Program to show functions with arguments and return values

```
main()
 {
        float prin.rate,amt;
        int per;
        printf("Enter principal amount, interest");
        printf("rate and period\n");
        scanf("%f %f %d", &prin,&rate,&per);
        printtext('*', 52);
```

```
            amt=value(prin,rate,per);
            printf("\n %f %f %d %f \ n \n", prin,rate,per,amt);
            printtext('=', 52);
    }

     printtext(ch,l)
     int l;
      char ch;
    {
        int j;
        for(j=1;j<=52;j++)
         printf("%c", ch);
          printf("\n");
}

    value(p,r,n)
     int n;
      float p,r;
       {
         int year;
          float sum;
          sum=p;
          year=1;
          while(year<=n)
              {
                sum=sum*(1+r)
                year=year+1;
              }
            return(sum);
}
```

1. The function call transfers the control along with the copies of the values of the actual arguments to the function value where the formal arguments p,r and n are assigned the values of prin, rate and per respectively.
2. The called function value is executed line by line in a normal fashion until the return (sum); statement is encountered. The value of sum is passed back to the function call in the main program.
3. The calling statement is executed normally and the returned value is thus assigned to amt.

Returning non-integer value from functions:

The function value mentioned above in the example does all the calculations using floats but the return statement

```
return(sum);
```

returns only the integer part of sum. This is due to the absence of the type-specifier in the function header. Some times it might be necessary to receive float or double type of data.

We should follow the steps below to enable a calling function to receive a non-integer value from a called function.

1. The explicit type_specifier, corresponding to the data type required must be mentioned in the function header. The general form of the function definition is.

```
Type-specifier function-name(argument list)
        argument declaration;
        {
           function statements;
        }
```

The type-specifier tells the compiler, the type of data the function is to return.

2. The called function must be declared at the start of the body in the calling function like any other variable. This is to tell the calling function the type of data that the function is actually returning.

Program showing the transfer of a floating-point value between functions.

```
main()
{
    float x,y, mul();
    double div();
    x= 34.238;
    y= 6.78;
    printf("%f \n", mul(x,y));
    printf("%f \n", div(x,y));
  }
    float mul(x,y)
    float x,y;
    {
      return(x * y);
    }

    double div(p,q)
    double p,q;
    {
       return(p/q);
    }
}
```

The declaration part of main function declares not only the variables but the functions mul and div as well. This only tells the compiler that mul will return a float-type value and div a double type value. Parantheses that follow mul and div specify that they are functions instead of variables.

If we have a mismatch between the type of data that the called function returns and the type of data that the calling function expects, we will have unpredictable results. We must be very careful to make sure that both types are compatible.

Array in functions.

It is possible to pass the values of an array to a function. To pass an array to a called function, it is sufficient to list the name of the array without any subscripts, and the size of the array as arguments.

The largest(a,n):

Will pass all the elements contained in the array a of size n. The called function expecting this call must be defined appropriately. The largest function header might look like:

```
float largest(array,size)
float array[ ];
int size;
```

The function largest is defined to take arguments , the array name and the size of the array to specify the number of elements in the array. The declaration of the formal argument array is made as follows:

```
float array[ ];
```

The pair of brackets informs the compiler that the argument is an array of numbers. It is not necessary to specify the size of the array here.

Consider the following example:

```
main()
{
  float large();
   static float value[5]={ 3.4,-12.78,1.23,5.67,8.90};
  printf("%f \n", large(value,5));
}
   float large(x,n)
float a[ ];
int n;
     {
          int j;
      float max;
      max=a[0];
          for(j=1; j<n; j++)
            if(max< a[j])
               max=a[j];
        return(max);
}
```

When the function call large(value,5) is made the value of all elements of the array value are passed to the corresponding elements of array a in the called function. The large function finds the largest value in the array and returns the result to the main.

Global variables:

Variables that are both alive and active throughout the entire program are known as global or external variables. The global variables can be accessed by any function in the program. External variables are declared outside a function.

The external declaration of integer number and float length might appear as:

```
int number;
float length=7.5;
main()
{
  ----------
   ---------
}

function1()
{
  ------------
  ----------
}

function2()
{
 ---------
  --------
}
```

The variables number and length are available for use in all the three functions. When the local variable and a global variable have the same name, the local variable will have precedence over the global one in the function where it is declared.

Consider the following example:

```
int cnt;
main()
{
   cnt=10;
    ------------
    -----------
}
 function()
   {
        int cnt=0;
         -----------
         ----------
           cnt=cnt+1;
    }
```

When the function references the variable cnt, it will be referencing only its local variable, not the global one. The value of cnt in main will not be affected.

A program to show the properties of global variables.

```
int x;
main()
{
        x=15;
        printf("x= %d \n", x);
        printf("x=%d \n", fun1());
        printf("x=%d\n", fun2());
        printf("x=%d\n", fun3());
}
fun1()
{
        x=x+15;
        return(x);
}

fun2()
{
        int x;
        x=1;
        return(x);
}
fun3()
{
        x=x+15;
        return(x);
}
```

Local variables:

These type of variables are declared inside a function in which they are to be utilized. They are created when the function is called and destroyed automatically when the function is exited. Hence these variables are also known as automatic variables.

A variable declared inside a function without storage class specification is by default a local variable.

```
        main()
        {
            int number;
             ------------
             ------------
        }
```

Program to show the working of local variables.

```
main()
{
        int m=1000;
        function2();
        printf("%d\n", m);
}

function1()
{
        int m=10;
        printf("%d\n", m);
}

function2()
{
        int m=1000
        function();
        printf("%d\n", m);
}
```

Register variable:

Some times we can tell the compiler that a variable should be kept in one of the machine's registers, instead of keeping in the memory. Since the access of register variables is faster than the memory access. The general format is as follows:

```
Register int variable name;
```

Since only a few variables can be placed in the register, it is important to carefully select the variables for this purpose. Once the limit is reached C will automatically convert register variables into non-register variables.

Static variable:

A variable can be declared static using the keyword static like

```
static int a;
static float y;
```

A static variable may be either an internal type or external type, depending on the place of declaration.

Internal static variables are those which are declared inside a function. The scope of internal static variables extend up to the end of the function in which they are defined. These internal static variables are similar to local variables except that they remain in existence throughout the remainder of the program. Therefore , internal static variables can be used to retain values between function calls.

Program to show static variable

```
main()
{
        int j;
        for(j=1;j<=3;j++)
        stat();
}

        stat()
{
        static int y=0;
        y=y+1;
        printf("y=%d \n",y);
}
```

Programming examples:

1. **Program to calculate standard deviation using function.**

```
#include<math.h>
main()
{
        float value[5], stddev();
        int i;
        printf("Enter the values\n");
        for(i=0;i<5;i++)
        scanf("%f", &value[i]);
        printf("Std.deviation is %f\n", stddev(value,5);
}
float stddev(x,n)
float x[ ];
int n;
{
        int i;
        float mean(),y,sum=0.0;
        y=mean(x,n);
        for(i=0;i<n;i++)
        {
                sum=sum+(y-x[i])*(y-x[i]);
                return(sqrt(sum/(float)n));
      }
float mean(x,n)
float x[ ];
```

```
int n;
{
int i;
float sum=0.0;
for(i=0;i<n;i++)
  sum=sum+x[i];
return(sum/(float)n);
}
```

Program to show sorting of array elements.

```
        Main()
        {
                int i;
                static int num[6]={34,56,23,12,95,67};
                printf("The numbers before soring\n:");
                for(i=0;i<6;i++)
                   printf(i=0;i<6;i++)
                    printf("\n");
}
sort(n,y)
int m, x[ ];
{
        int i,j,temp;
        for(i=1 ; i<=n;i++)
          for(j=1;j<=n-i;j++)
             if(x[j-1]>=x[j])
                {
                   temp=x[j-1];
                    x[j-1]=x[j];
                     x[j]=temp;
  }
 }
```

Strings

String variable:

A string is an array of characters. Any group of characters defined between double quotation marks is called a constant string.

Example:

"Good Morning Everybody"

Character strings are often used to build meaningful and readable programs.
A string variable is any valid C variable name and is always declared as an array.

Declaring and initializing string variables:

The general form of string variable is

```
char string_name[size];
```

The size determines the number of characters in the string-name.

Some examples are:

```
char  state[10];
char  name[30];
```

When the compiler assigns a character string to a character array, it automatically supplies a null character('\0') at the end of the string.

Character arrays may be initialized when they are declared. C permits a character array to be initialized in either of the following two forms:

```
Static char state[10]=" KARNATAKA";
Static char state[10]={'K','A','R','N','A','T','A','K','A'\O'};
```

The reason that state had to be 10 elements long is that the string KARNATAKA contains 10 characters and one element space is provided for the null terminator.

C also permits us to initialize a character array without specifying the number of elements.

For example, the statement

static char string[ ] ={'H', 'E', 'L', 'L', 'O' \O};

Defines the array string as a six element array.

Reading and writing strings:

To read a string of characters input function scanf can be used with %s format specification.

Example:

```
char add[20];
Scanf("%s", add);
```

Note that unlike previous scanf calls, in the case of character arrays, the &(ampersand) is not required before the variable name. The scanf function automatically terminates the string that is read with a null character and therefore the character array should be large enough to hold the input string plus the null character.

Program to read a series of words using scanf function

```
main()
{
   char text1[50],text2[50],text3[50],text4[50];
   printf("Enter   text:\n");
   scanf("%s  %s", text1,text2);
   scanf("%s", text3);
   scanf("%s", text4);
   printf("\n");
   printf("text1= %s\n text2=%s\n", text1,text2);
   printf("text3=  %s\n text4= %s\n", text3,text4);
}
```

Writing strings:

The printf function with %s can be used to display an array of characters that is terminated by the null character.

Example:

```
printf("%s",  text);
```

Can be used to display the entire contents of the array name.

We can also specify the precision with which the array is displayed. For example, the specification

%12.4

indicates that the first four characters are to printed in a field width of 12 columns.

Program to illustrate writing strings using %s format

```
Main()
{
        static char state[15]= "MADHYA PRADESH";
        printf("\n \n");
        printf("----------------------------------\n");
        printf("%13s\n", state);
        printf("%5s\n", state);
        printf("%15.6s \n", state);
        printf("%15.0s\n", state);
        printf("%.3s\n", state);
}
```

String functions:

C library supports a large number of string functions. The list given below depicts the string functions

Function	Action
strcat()	concatenates two strings
strcmp()	compares two strings
strcpy()	copies one string with another
strlen()	finds the length of a string.

String Concatenation :strcat() function:

The strcat function joins two strings together. The general form is

strcat(string1,string2);

string1 and string2 are character arrays. When the function strcat is executed. String2 is appended to string1. It does so by removing the null character at the end of string1 and placing string2 from there. The string at string2 remains unchanged.

Example:

Text1= VERY \0
Text2= GOOD\0
Text3= BAD\0

Strcat(text1,text2);

Text1= VERY GOOD\0
Text2= GOOD\0

Strcat(text1,text3);

Text1= VERY BAD
Text2= BAD

We must make sure that the size of string1 is large enough to accommodate the final string.
Strcat function may also append a string constant to string variable.
For example:
strcat(text1,"GOOD");

C permits nesting of strcat functions. The statement
strcat(strcat(string1,string2),string3);
Is allowed and concatenates all the three strings together. The resultant string is stored in string1.

String comparison/strcmp() function:

The strcmp function compares two strings identified by the arguments and has a value 0 if they are equal.
The general form is :
strcmp(string1,string2);
String1 and string2 may be string variables or string constants.
Examples are:
strcmp(name1,name2);
strcmp(name1, "ABHI");
strcmp("ROM", "RAM");

We have to determine whether the strings are equal, if not which is alphabetically above.

String copying/strcpy() function:

The strcpy() function works almost like a string-assignment operator. The general format is
strcpy(string1,string2);
It copies the contents of string2 to string1. string2 may be a character variable or a string constant.
For example, the statement
strcpy(city , "BANGALORE");
Will assign the string "BANGALORE" to the string variable city.

The statement strcpy(city1,city2); will assign the contents of the string variable city2 to the string variable city1. The size of the array city1 should be large enough to receive the contents of city2.

Finding the length of a string/strlen();

This function counts and returns the number of characters in a string.
The general syntax is n=strlen(string);

Where n is an integer variable which receives the value of the length of the string. The argument may be a string constant. The counting ends at the first null character.

Implementing the above functions without using string functions:

String concatenation:

We cannot assign one string to another directly,we cannot join two strings together by the simple arithmetic addition. The characters from string1 and string2 should be copied into the string3 one after the other. The size of the array string3 should be large enough to hold the total characters.

Program to show concatenation of strings:

```
Main()
{
        int i,j,k;
        static char first_name={"ATAL"};
        static char sec_name={"RAM"};
        static char  last_name={"KRISHNA"};
        char name[30];

        for(i=0;first_name[i]!='\0';i++)
            name[i]=first_name;
        for(i=0;second_name[j]!='\0'; j++)
            name[i+j+1]=sec_name[j];
             name[i+j+1] =' ';
        for(k=0;last_name[k]!='\0';k++)
             name[i+j+k+2]=last_name[k];
             name[i+j+k+2]='\0';
              printf("\n \n");
               printf("%s \n", name);
}
Output
        ATAL RAM KRISHNA
```

String comparison:

Comparison of two strings cannot be compared directly. It is therefore necessary to compare the strings to be tested, character by character. The comparison is done until there is a mismatch or one of the strings terminates into a null character.
The following segment of a program illustrates this,

```
-----------------------------
i=0;
while(str1[i]==str2[i] && str1[i]!='\0'
        && str2[i]!='\0')
        i=i+1;
  if(str1[i]=='\0' && str2[i]=='\0')
        printf("strings are equal\n");
    else
        printf("strings are not equal\n");
-------------------------------
```

String copying:

Program to show copying of two strings:

```
main()
  {
        char string1[80],string2[80];
        int j;
        printf("Enter a string\n");
        printf("?");
        scanf("%s", string2);

        for(j=0;string2[i]!='\0';j++)
          string1[j]=string2[j];
          string1[j]='\0';
          printf("\n");
          printf("%s\n",string1);
          printf("Number of characters=%d\n", j);
}
```

Program to find the length of a string:

```
#include<stdio.h>
main()
{       char line[80],character
        int c=0,i;
        printf("Enter the text\n");
```

```
            for(i=0;line[i];!='\0';i++)
{           character=getchar();
             line[i]=character;
             c++;
 }
  printf("The length of the string \n", c);
}
```

Arithmetic operations on characters:

We can manipulate characters the same way we do with numbers. Whenever a character constant or character variable is used in an expression, it is automatically converted into an integer value by the system. The integer value depends on the local character set of the system.

To write a character in its integer representation , we may write it as an integer. For example:

```
y='a';
printf("%d\n", y);
```

will display the number 97 on the screen.

It is also possible to perform arithmetic operations on the character constants and variables. For example:

```
y='z'-1;
```

Is a valid statement. In ASCII , the value of 'z' is 122 and therefore , the statement will assign the value 121 to the variable Y.

We may also use character constants in relational expressions. For example:

```
ch>='a' && ch<='z'
```

Would test whether the character contained in the variable ch is an lower-case letter.

We can convert a character digit to its equivalent integer value using the following relationship :

```
y=character –'0';
```

Where y is defined as an integer variable and character contains the character digit.
Example: Let us assume that the character contains the digit '7', then,

```
y=ASCII value of '7'-ASCII value of '0'
    = 55-48
     =7
```

C library has a function that converts a string of digits into their integer values. The function takes the form

```
y=atoi(string);
```

y is an integer variable and string is a character array containing a string or digits
Consider the following example:

```
num="1974"
year=atoi(num);
```

Num is a string variable which is assigned the string constant “1974”. The function atoi converts the string “1974” to its numeric equivalent 1974 and assigns it to the integer variable year.

Programming examples:

Program to sort strings in alphabetical order:

```
#define ITEMS 10
#define MAX   25
main()
{
        char str [ITEMS][MAX], dum[MAX];
        int i=0;j=0;
        printf(“Enter names of %d items \n”, ITEMS);
        while(i<ITEMS)
        scanf(“%s”, str[i++]);

        for(i=1;i<ITEMS;i++)
{
        for(j=1;j<=ITEMS-i;j++)
         {
           if(strcmp(string[j-1],string[j])>0)
                strcpy(dummy,string[j-1]);
                strcpy(str[j-1],str[j]);
                strcpy(str[j],dummy);
        }
 }
        for(i=0;i<ITEMS;i++)
          printf(“%s”, str[i]);
}
```

Program to show string handling functions:

```
        #include<string.h>
        main()
        {
          char s1[20],s2[20],s3[20];
           int y,len1,len2,len3;
           printf(“\n Enter two string constants\n”);
           printf(“?”);
           scanf(“%s %s”, s1,s2);
            x=strcmp(s1,s2);
              If(y!=0)
                   {
                     printf(“\n\n Strings are not equal\n”);
```

```
        strcat(s1,s2);
    }
 else
    printf("\n\n Strings are equal\n");
    strcpy(s3,s1);
    len1=strlen(s1);
    len2=strlen(s2);
    len3=strlen(s3);
    printf("\n s1= %s length= %d character \n",s1,len1);
    printf("\n s1= %s length= %d character \n",s2,len2);
     printf("\n s1= %s length= %d character \n",s3,len3);
    }
```

Program to convert lowercase characters in to upper case characters:

```
#include<stdio.h>
    main()

    {
        char text[85];
        int i=0;
        printf("Enter a line of text in lowercase:\t");
        scanf("%[^\n]",text);
        printf("%s",text);
        printf("\n Converted to uppercase text is :\t");
        while(text[i]!='0')
          printf("%c", toupper(text[i]));
           i++;
        }
            printf("\n");
}
```

Pointers:

Pointer data type:

When we declare a variable name m as type integer we tell the compiler that a location in memory where an integer can be stored should be found and it should be given a name m.

Thus when we write

```
int m;
```

Will pick a memory box and give it a name m. The variable name is a symbolic name for the address of the memory box. We have already used the operator & which was used in scanf function. This operator is called the address operator.

If we write &m the operator & tells the compiler to find the numeric value of the address of a memory box whose symbolic name is m.

If we write:

```
n=&m;
```

Then the address of variable m is stored in n. When a variable stores an address we declare that variable as a pointer data type.

Thus n is declared as:

```
int *n;
```

This declaration says that n will store the address of an integer variable name.

```
n=&m;
```

variable name	address	contents
m	8468	35

Declaring and initializing variables:

Since pointer variables contain addresses that belong to a separate data type, they must be declared as pointers before we use them. The declaration of a pointer variable has the following form:

Data type *pt_name;

This tells the compiler three things about the variable pt_name.

1. The asterisk(*) tells that the variable pt_name is a pointer variable.
2. pt_name needs a memory location.
3. pt_name points to variable of type data type.

For example:

int *q;

Declares the variable q as a pointer variable that points to an integer data type.

Similarly, the statement

float *y;

Declares y as a pointer to a floating point variable.

Once a pointer variable has been declared, it can be made to point to a variable using an assignment statement such as

p=&quantity;

This causes p to point to quantity. That is, p now contains the address of quantity. This is known as pointer initialization.
We must ensure that the pointer variables always point to the corresponding type of data.

Example:

```
Float a,b;
Int x, *p;
p=&a;
b=*p;
```

In the above example the result will give an error because we are trying to assign the address of a float variable to an integer pointer. Care should be taken to avoid wrong pointer assignments.

A pointer variable can be initialized in its declaration .
Example:

```
int x;
*p=&x;
```

Accessing a variable through its pointer:

Once a pointer has been assigned the address of a variable, the question remains as to how to access the value of the variable using the pointer. This is done by using another unary operator *(asterisk), usually known as the indirection operator. Consider the following statements:

```
int qty, *q,m;
```

```
qty=165;
q=&qty;
m=*p;
```

The first line declares qty and m as integer variables and q as a pointer variable pointing an integer. The second line assigns the value 165 to qty and the third line assigns the address of qty to the pointer variable q. The fourth line contains the indirection operator * . When the operator * is placed before a pointer variable in an expression , the pointer will return the value of the variable. The * can be called as 'value at address' . Thus the value of q would be 165. The two statements

```
q=&qty;
m=*p;
```

are equivalent to

```
m=qty.
```

Program for accessing variables using pointers

```
main()

{
        int m,n;
        int *ptr;

        m=15;
        ptr=&m;
        y=*ptr;

        printf("Value of m is %d\n\n", m);
        printf("%d is stored at address %u \n", m,&m);
        printf("%d  is stored at address  %u \n", &m, &m);
        printf("%d is stored at address  %u \n", ptr, &ptr);
        printf("%d is stored at address  %u \n", y, &y);
          *ptr=30;
        printf("\n Now m=%d\n",x);
}
```

Pointers and one dimensional arrays:

When an array is declared , the compiler allocates a base address and sufficient amount of storage to contain all the elements of the array in memory locations. The base address is the location of the first element(index 0) of the array.

```
static int y[4]= {1,2,3,4,5};
```

Suppose the base address of y is defined as a constant pointer pointing to the first element , y[0] and therefore the value of y is 1000.

That is,

y=&y[0]=1000

If we declare p as an integer pointer , then we can make the pointer p to point to the array y by the following assignment:

p=y;

This is equivalent to

p=&y[0];

The relationship between p and y is shown below.

P =&y[0] (=1000)
P+1 =&y[1](=1002)
P+2 =&y[2](=1004)
P+3 =&y[3](=1006)
P+4 =&y[4](=1008)

Program to show pointers in one-dimensional arrays:

```
main()
{
        int *p,sum,j;
        static int y[5]={4,5,7,8,0};
        j=0;
        p=y;
        sum=0;
        printf("Element value address\n\n");
        While(j<5)
        {
           printf(" y [%d]     %d    %u  \n", j, *p, p);
            sum=sum+p;
            i++;
            p++;
        }
         printf("\n Sum= %d \n", sum);
         printf("\n &x[0]=%u \n", &x[0]);
         printf(" \n p = %u \n ", p);
}
```