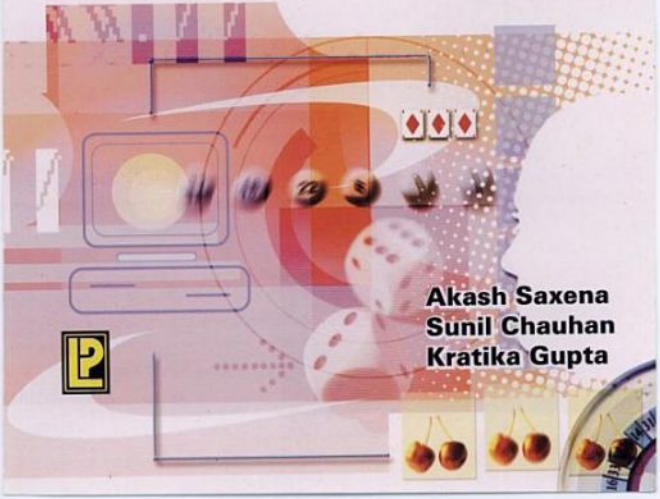
Fundamentals of Computer



Akash Saxena
Sunil Chauhan
Kratika Gupta

Premier12

CONTENTS

PART – A

1 INTRODUCTION TO COMPUTER SYSTEM

INTRODUCTION

Computer is an electronic device capable of carrying out sequence of instruction. It helps to accept (read) the input data, process it according to the instructions given and produces (write) processed output information that is results.

It combines five elements:

1. Hardware
2. Software
3. People
4. Procedures
5. Data / Information

Computer has ability of:

- Accept data
- Input, store & execute instructions
- Perform mathematical & logical operations
- Output results according to user requirements

Data: It is a collection of raw facts.

Processing: It is the work done by the computer to process input data and to produce output data.

Processing includes:

- Calculations: it includes equal +, –, %, *,
- Comparison: it includes equal to == <>
- Decision-making: it includes branching to a different path depending on condition.
- Logic: it includes the sequence or flow of step to be followed to get the desired result.

Information: It is the processed data that generates after data processing, which can be used to help people to make decisions. It refers the facts and figure or statistics that have meanings.

Average marks secured by each student from "data" from which class average is calculated that is the information generated.

CHARACTERISTICS OF THE COMPUTER

Speed

It is related to the volume of data processed per unit of time. A computer is very fast device. The speed of computer is measured in:

Micro seconds 10^{-6}

Nano seconds 10^{-9}

Pico seconds 10^{-12}

To classify speed of different computer's criteria of MIPS "Million instruction per seconds" is used. Data processors can compare computer-processing speed by the comparing the number of instructions a computer can perform in one second. Fastest computer can execute nearly one Billion instructions per second.

Accuracy

Errors are usually due to human rather then technological or may due to inaccurate data computer does not make mistakes. High accuracy for any computer system is desired. The degree of accuracy is depends on the design model of the computer. Computer is based on the principle of (GIGO) Garbage-In-Garbage-Out if wrong data inputted then a wrong output will be produced.

Reliability

It is exceptionally. It operates under the most advance conditions without showing any signs of fatigue. It is free tiedness, monotony lack of concentration etc. therefore it is used for type of jobs, which required great accuracy. It does not loose concentration.

Memory Capability

It has unlimited capacity to store the data and recall any amount of information because of its secondary storage.

Storage

Large no of the programs and data can be stored on the computer. We don't have to feed programs every time a job is to be done on the computer. Once the program for a job has been stored, it can be recalled and the computer can be asked to execute it.

Diligence

A computer can perform number of functions without suffering from tiredness, lack of concentration etc. if an instruction is given to perform 1 lakh calculation then the machine will perform all the calculation with same speed and accuracy.

Versatility

A computer can control a variety of jobs used in various fields. For performing any job a computer needs a set of program and its related data.

Drawbacks of Computers

- ✔ Lack of decision-making power
- ✔ No I.Q.
- ✔ No heuristics

The Evolution of Computers

1. **The Mark I Computer (1937–44)**

   Also known as automatic sequence controlled calculator, this was first fully automatic calculating machine design by Howard. The mark I was not an electronic computer, rather, it was an electromechanical computer. The size of this computer was huge and its design was very complex, but it was quite reliable. Its design was based on the techniques already developed for punched card machinery.

2. **The Atanasoff Computer (1939–42)**

   The electronic machine was developed by Dr. John Vincent Atanasoff to solve certain mathematical equation. It used 45 vacuum tubes for internal logic and capacitors for storage.

3. **The ENIAC (1943–46)**

   The Electronic Numeric Integrator And Calculator (ENIAC) was the first electronic computer. ENIAC was developed because of military need, and was used for many year to solve ballistic problems. It used about 19000 vacuum tube. The addition of two numbers was achieved in 200 microsecond and multiplication in 2000 microsecond.

4. **The EDVAC (1946–52)**

   The Electronic Discrete Variable Automatic Computer (EDVAC) was the effort to developed a stored instruction computer. The idea of storing both instruction and data in binary form, instead of decimal number system used by human being.

5. **The EDSAC (1947–49)**

   The Electronic Delay Storage Automatic Computer (EDSAC) . In this machine , addition was accomplished in 1500 microseconds and multiplication in 4000 microseconds.

6. **The UNIVAC I (1951)**

   The Universal Automatic Computer (UNIVAC) was the first digital computer. The commercially available digital computer which could be used for business and scientific applications, had arrived.

GENERATION OF COMPUTERS

First Generation (1942–1955)

We have already known about some of the early computer ENIAC, EDVAC etc. These machine and other of their time were made possible by the invention of "vacuum tube" which was a fragile glass device that could and amplify electronic signals. These vacuum tube computers are referred to first generation computer.

Advantages

- ✔ Vacuum tubes were the only electronic components available during those days. Vacuum tube technology made possible the advent of electronic digital computers.
- ✔ These computers were the fastest calculating devices of their time.
- ✔ They could perform computations in milliseconds.

Disadvantages

- ✔ Too bulky is size and unreliable.
- ✔ Thousand of vacuum tubes that were used emitted large amount of heat and burnt out frequently.
- ✔ Air conditioning required.
- ✔ Non-portable.
- ✔ Commercial production was difficult and costly.
- ✔ Limited commercial use.

Second Generation (1955–1964)

The second generation of computer was marked by the use of transistors in place of vacuum tube. Invention of transistors at Bell Labs by John Bardeen William Shockley and walter Brattain Transistors had a number of advantages over the vacuum tube. As transistors were made from pieces of silicon , so they were more compact than vacuum tubes.

Advantages

- ✔ Smaller in size as compared to the first generation computer.
- ✔ More reliable.
- ✔ Less heat generated and less prone to hardware failures.
- ✔ Better portability.
- ✔ Wider commercial use.
- ✔ They were slightly-faster.

Disadvantages

- ✔ Air condition required.
- ✔ Frequent maintenance required.
- ✔ Commercial production was difficult and costly.

Third Generation (1964–1975)

They were still more compacting faster and less expensive than previous generation along with the previous generation. This computer used "Integrated Circuit". These replacing the transistors, which was used in second-generation computer. It uses integrated circuits (IC). It integrates large no of circuit's elements into surface of silicon chips. These computers used IC for CPU computer. In the beginning they use magnetic core memory but later they used semi conductor memory. Semi conductor memory was LSI chip magnetic disk drums & tape was used as secondary memory. Cache memory was also incorporated.

Microprogramming, parallel programming, multi programming, multi user system (time sharing system) etc. were used. Multiprocessing too. The concept of virtual memory was also introduced. Example IBM 370 series, CDC 7600.

Advantages

- ✔ Smaller in size as compare to previous generation.
- ✔ More reliable.
- ✔ Lower heat generated
- ✔ Computational time was reduced from micro second to nano second

- ✔ Maintenance cost was low because hardware failure was rare
- ✔ Portable
- ✔ Used for commercial application
- ✔ Less power required than previous of generation of computer
- ✔ Commercial production was easier and inexpensive
- ✔ They were totally generally purpose machine.

Disadvantages

- ✔ Air condition was required in many cases
- ✔ Highly sophisticated technologies was required for the manufacturing of IC chip

Fourth Generation (1975 onwards)

It integrated up to 100 components on a single chip that is Medium Scale Integration (MSI)

LSI (Large Scale Integration)

It contain over 30,000 components on a single chip

VLSI (Very Large Scale Integration)

That is over 1 million components on a single chip.

Fourth generation use VLSI chips for both CPU and memory. CPU consists of one or more microprocessors. The latest microprocessors can contain one million transistors. Cache memory is provided on CPU chip.

Advantages

- ✔ Smaller in size because of high density of component in a small chip.
- ✔ Very reliable
- ✔ Heat generation very low
- ✔ Less powerful air conditioning
- ✔ Faster computation
- ✔ Minimum maintenance is required
- ✔ In expensive among all generation
- ✔ They were totally generally purpose machines

Disadvantages

- ✔ Highly sophisticated technology required for the manufacturing of LSI chip

Fifth Generation (yet to come)

These computers required genuine IQ that is ability to reason logically and contain real knowledge. The structure will be parallel. They will be able to perform multiple tasks simultaneously. Data and knowledge are processed they shall follow 'kips' than dips/lips.

KIPS (Knowledge Information Processing System)

DIPS (Data Information Processing System)

LIPS (Logical Information Processing System)

Japan has started research using prolog languages as operating system.

Classification of Computer

Microcomputer

Two major events led to the introduction of the first microcomputer: the development of the first microprocessor in 1969, and the creation of the first general purpose microprocessor chip in 1971. Microcomputers had a major economic impact due to their small size and drastically improved cost-effectiveness. These powerful, yet easy to operate, machines have been called home computers, laptop computers, notepad computers, personal computers (PC), palmtop computers, and micros. The microcomputer can perform jobs once only handled by the largest computers. A typical microcomputer memory unit stores between 32 and 1,000 million characters of data. Since these computers are slower than their mainframe and minicomputer counterparts, processing speeds are measured in Megahertz (MHz) or Gigahertz (GHz) instead of MIPS. Generally, the larger the processing speed, the faster the computer is. A computer running at 2.53 GHz (2,530 MHz) is faster than one running at 450 MHz.

Minicomputer

These are powerful than microcomputer and supports several uses. They use large RAM and backing storage capacity and are quick in data processing *e.g.*, PDP ii VAX 7500. The minicomputer was introduced in 1963. It is a scaled-down version of the mainframe computer. It is capable of performing many jobs that old mainframe computers used to accomplish. Due to its cost-effectiveness, the minicomputer's most important contribution has been the introduction of distributed computing, where a number of small computers can be used instead of one giant central computer.

Mainframe Computer

These computers are very powerful large general-purpose compute. They are faster and more powerful than minicomputer. The word length 48,60 or 64 bits. They are used for complex calculation where large amount of data is to be processed. They are useful in research organization banks, airline, reservations where large capacity of these computer have increased *e.g.*, IBM4381, ICL39 series.

The first general-purpose computer to be developed, in the 1940s, was the mainframe. Mainframe computers can be used by many people at the same time, and can handle very complex problems or large volume jobs. Mainframes are generally used for data processing for business and scientific applications. For example, major banks can process bills for credit card holders all over the world with the aid of a mainframe. Large research projects also use the mainframe's vast memory, storage capacity, and super-fast processing speed to conduct tests and co-ordinate operations.

Mainframes can have memory capacities measured in billions of characters and more. The largest mainframes can process over 3,000 Million Instructions Per Second (MIPS).

Super Computer

They are much faster and more powerful than main computer. There processing speed lines in a range of 400 MIPS to 10000 MIPS. Word length is 64 to 96 bytes. These computers are design to maximize the number of FLOPS and it is usually more than one GFLOPS. They contain number of CPU, which operate in parallel to make it faster. They are used for whether for casting, rocketing in aero dynamics, atomic nuclear and plasma physics *e.g.*, CRAY-XMA. India has designed super computer indigenously called PARAM 10000.

Broad Classification

Analog Computer

Analog computers measure, analyses, and control events that are continuous, such as the flow of electrical current along power lines.

Uses : In scientific and engineering fields use then most because quantity is very constantly.

Digital Computer

Digital computers, on the other hand, control events that are discrete or finite *e.g.*, process in automobile, control fuel, braking system.

Hybrid Computer

This machine combines feature of both analog and digital computer *e.g.*, ECG machine. Analog machine measure patient temperatures then they are converted to numbers and supply digital computer.

□□□

2 COMPUTER ORGANIZATION

ARCHITECTURE OF COMPUTER SYSTEM

The internal architecture design of computers differs from one system model to another. However, basic organization remains same for all computer systems. A block diagram of the basic computer organization is shown in figure.

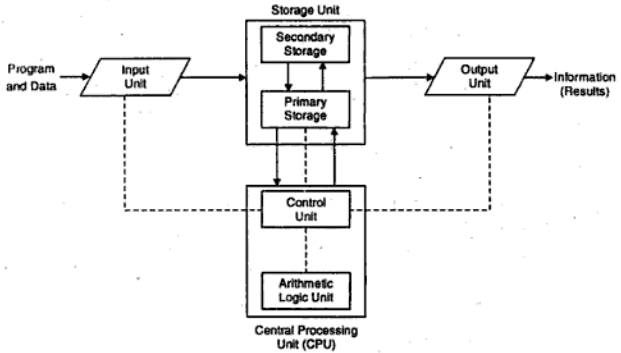


Figure 2.1. Basic organization of a computer system.

There are three major unit of an electronic digital computer. The units are Input unit, Output unit, and Central Processing Unit (CPU).

Input Unit

The function of input unit is to read necessary data into computer system or we can say that unit links the external environment with the computer system. Some of the more common input devices are keyboards, punched card and punched paper tape reader, magnetic tape

reader etc. Data and instructions are taken in various forms depending on input devices used. For example, data through keyboard entered similar to typing, and this differs from the way in which data is entered through a punched card. However regardless of the forms in which they accept input, all input devices transformed that into binary codes that the primary memory of the computer is designed to accept. This transformation is accomplished by units called input interfaces.

The following functions are performed by an input unit:

- ✔ It must provide a computer with data and instructions that are necessary to perform the task.
- ✔ It converts that input data in computer acceptable form.
- ✔ It supplies the converted instructions and data to the computer system for further processing.

Output Unit

The function of an output unit is just reverse of that of input unit. Output unit supplies the computed data to the outside world. Thus it links computer with the external environment. As computer work with the binary code, the results produced are also in the binary form. Hence before supplying the results to the outside word, it must be converted to human readable form. This task is accomplished by units are called output interfaces. Common output devices are CRT displays, card-punching machines, and magnetic tape drives etc.

The following functions are performed by an output unit:

- ✔ It accepts the results produced by the computer.
- ✔ It converts these coded results to human readable form.
- ✔ It supplies the converted results to the outside world.

Central Processing Unit (CPU)

The CPU is the brain of a computer system. In a computer system all major calculations and comparisons are made inside the CPU and the CPU is also responsible for activating and controlling the operations of other units of a computer system. The two basic components of a CPU are the control unit and the arithmetic logic unit.

Arithmetic Logic Unit (ALU)

The Arithmetic Logic Unit (ALU) of a computer system is the place where the actual execution of the instructions takes place during the processing operation. In general, all calculations (addition, subtraction, multiplication and division etc.) are performed and all comparisons (decisions) are made in the ALU. The data stored in the primary storage are transferred to ALU where processing takes place and intermediate and final results are transferred to storage. The control unit tells the ALU which operation to perform and then see that necessary data to be supplied.

Control Unit

The control unit sequences the operation of the computer, controlling the actions of all other units. That interprets the instructions and then directs the rest of the machines in its operation. Although, it does not perform any actual processing on the data, the control unit acts as a central nervous system coordinates the entire computer system.

Storage Unit

The input data, final and intermediate results produced by the computer must be kept somewhere inside the computer system. The storage unit of a computer is designed to fulfill to all these needs.

The functions of storage unit are to hold:

- ✔ All the data to be processed and the instructions required for processing.
- ✔ Intermediate results of processing.
- ✔ Final results of processing before these results are released to an output device.

The storage unit of all computers is comprised of the following two types of storage:

Primary Storage

The primary storage, also known as main memory, is used to hold pieces of program instructions and data, intermediate results of processing, and recently produced results of processing. However, the primary storage can hold information only while the computer system is on. As soon as the computer system is switched off or reset, the information held in the primary storage disappears. The primary storage of modern computer systems is made up of semiconductor devices.

Secondary storage

The secondary storage is used to take care of limitations of the primary storage; it is used to supplement the limited storage capacity and the volatile characteristic of primary storage. Secondary storage is much cheaper than primary storage and it can retain information even when computer system is switched off or reset. The most commonly used storage medium is **the magnetic disk.**

□□□

3 NUMBER SYSTEM

INTRODUCTION

It is system for representing numbers in systematic format. It does provide us basic rules for using numbers and performing computation on it.

Number systems are basically of two types:

1. Non-positional
2. Positional.

Non-Positional Number System

In this system uses an additive approach or the non-positional number system such as Roman number system, Unary number system. As it is non-positional system, since each symbol represents the same value regardless of its position in the number and the symbols are simply added to find out the value of particular number. Since it is very difficult to perform arithmetic with such a number system.

Unary Number System

In this only one symbol (1) is used to represent numbers such as 5 represented as 11111.

Roman Number System

In this system, the symbols I, V, X, L and C to represent the numbers 1, 5, 10, 50 and 100 respectively. Sequences of these symbols can be used to represent other numbers *i.e.*, 62 represented as LXII.

Positional Number Systems

These systems have only a few symbols called digits. Digits represent different values depending on the position they have in the number. Determination of a number require following considerations:

1. The digit;
2. The position of the digit in the number;
3. The base of the number system.

Characteristics of Positional Number Systems determined by the Value of the Base

- ✔ Determination of the total number of different digits possible in the number system (first digit is always 0).

✔ Determination of the maximum value of a single digit that is always one less than the value of the base.

In all number systems, it is important only to keep track of the base of the number system in which we are working. In that number system any number can be represented by sequence $d_n d_{n-1} \ldots\ldots\ldots d_2 d_1$ and value of it can be computed using following formula:

$V = d_n * b^{n-1} + d_{n-1} * b^{n-2} + \ldots\ldots\ldots\ldots + d_2 * b^1 + d_1 * b^0$, where $d_n, d_{n-1}, \ldots\ldots\ldots, d_2, d_1$ represent value of corresponding digit and b is base of number system. As this formula represents that right most digit associated with lowest weight and **leftmost** bit associated with largest weight, so the rightmost digit and left most digit is known as least significant bit (LSB) and most significant bit (MSB). $d_n d_{n-1} \ldots\ldots\ldots d_2 d_1$ is largest number if each digit has highest value available in specific number system.

Advantages over non-positional number system

✔ Systematic approach

✔ Provides easy computation

COMMONLY USED NUMBER SYSTEM

Number System	Base	Available Digits
Decimal	10	0, 1, 2, 3, 4, 5, 6, 7, 8, 9
Binary	2	0, 1
Octal	8	0, 1, 2, 3, 4, 5, 6, 7
Hexadecimal	16	0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F

Decimal Number System

Most commonly used number system is decimal number system. This system allows use of ten symbols or digits (0, 1, 2, 3, 4, 5, 6, 7, 8, 9) because its base is equal to 10. Each position represents a specific power of the 10.

For example 4234_{10} in decimal number system can be written as

$= (4 \times 10^3) + (2 \times 10^2) + (3 \times 10^1) + (4 \times 10^0)$

$= (4 \times 1000) + (2 \times 100) + (3 \times 10) + (4 \times 1)$

$= 4000 + 200 + 30 + 4 = 4234$

Binary Number System

The binary number system has base 2, so only two symbols or digits (0 and 1) can be used in it. Note that the largest digit is 1(one less than the base). Each position in a binary number represents a power of the base (2).

The value of the binary number can be computed using defined formula.

For example value of 11100_2

$= (1 \times 2^4) + (1 \times 2^3) + (1 \times 2^2) + (0 \times 2^1) + (0 \times 2^0)$

$= 16 + 8 + 4 + 0 + 0$

$= 28$

Octal Number System

The octal number system has the base 8, so in only eight symbols or digits: 0, 1, 2, 3, 4, 5, 6, 7. The largest digit is 7 (one less than base). Again, each position in an octal number represents a power of the base (8).

The value of the octal number can also be computed also using same as in above number systems.

For example 4131 (written as 4131_8) is

$= (4 \times 8^3) + (1 \times 8^2) + (3 \times 8^1) + (1 \times 8^0)$

$= 4 \times 512 + 1 \times 64 + 3 \times 8 + 1 \times 1$

$= 2048 + 64 + 24 + 1$

$= 2137$

Since there are only 8 digits in the octal number system, so 3 bits ($2^3 = 8$) are sufficient to represent any octal number in binary.

Hexadecimal Number System

The hexadecimal number system has a base of 16 and it allows choices of 16 single character digits or symbols. The first 10 digits are the digits of a decimal system 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 and the remaining six digits are denoted by A, B, C, D, E, and F respectively. In the hexadecimal number system, therefore, the letters A through F are number digits representing value 10 to 15. Thus, largest single digit is F or 15 (one less than the base).

2BD is equivalent to:

$= (2 \times 16^2) + (B \times 16^1) + (D \times 16^0)$

$= (2 \times 256) + (11 \times 16) + (13 \times 1)$

$= 512 + 176 + 13$

$= 601$

Thus $2BD_{16} = 601_{10}$

Since there are only 16 digits in the hexadecimal number system, so 4 bits($2^4 = 16$) are sufficient to represent any hexadecimal number in binary.

CONVERTING FROM ONE NUMBER SYSTEM TO ANOTHER

The input and the final output values are in decimal so it is often required to convert number in other number systems to decimal and vice-versa. Any number value represented in one number system can be represented in other number system.

Converting To Decimal From Another Base

The following steps are performed in sequence:

- ✔ Determine the positional value val1 of each digit or symbol.
- ✔ Determine corresponding positional value val2 in some power of base for each digit according its position.
- ✔ Multiply the obtained values val1, val2 obtained for corresponding positions.
- ✔ Sum the products calculated in step 2 and it is the equivalent to value in decimal.

Example 1. $101001_2 = ?_{10}$.

Solution:

Position	Positional value val2
1	$2^0 = 1$
2	$2^1 = 2$
3	$2^2 = 4$
4	$2^3 = 8$
5	$2^4 = 16$
6	$2^5 = 32$

$= 32 \times 1 + 16 \times 0 + 8 \times 1 + 4 \times 0 + 2 \times 0 + 1 \times 1$

$= 32 + 0 + 8 + 0 + 0 + 1$

$= 41$

Hence, $101001_2 = 41_{10}$.

Example 2. $312_4 = ?_{10}$.

Solution: $312_4 = (3 \times 4^2) + (1 \times 4^1) + (2 \times 4^0)$

$= 3 \times 16 + 4 + 2$

$= 48 + 4 + 2$

$= 54$

Hence, $312_4 = 54_{10}$.

Example 3. $7012_8 = ?_{10}$.

Solution: $7012_8 = (7 \times 8^3) + (0 \times 8^2) + (1 \times 8^1) + (2 \times 8^0)$

$= 7 \times 512 + 0 \times 64 + 1 \times 8 + 2 \times 1$

$= 3584 + 0 + 8 + 2$

$= 3594$

Hence, $7012_8 = 3594_{10}$.

Example 4. $BFA_{16} = ?_{10}$.

Solution: $BFA_{16} = (12 \times 16^2) + (15 \times 16^1) + (10 \times 16^0)$

$= 12 \times 256 + 15 \times 16 + 10 \times 1$

$= 3072 + 240 + 1$

$= 3313_{10}$

Hence, $BFA_{16} = 3313_{10}$.

Example 5. $A1A_{12} = ?_{10}$.

Solution: $A1A_{12} = (10 \times 12^2) + (1 \times 12^1) + (A \times 12^0)$

$= 10 \times 144 + 1 \times 12 + 10 \times 1$

$= 1440 + 12 + 10$

$= 1462$

Hence, $A1A_{12} = 1462_{10}$.

Converting From Base 10 To a New Base

The following steps are performed in sequence:

(1) Divide the decimal number by the new base value.

(2) Record the remainder from step 1 as the rightmost digit of the new base number.

(3) Divide the quotient of the step1divide by the new base.

(4) Record the remainder from step 3 as the next digit of the new base number.

(5) Repeat steps 3 and 4, recording remainders from right to left, until the quotient becomes zero in step 3.

Note that the last remainder thus obtained will be the most significant digit (MSD) of the new base number.

Example 6. $41_{10} = ?_2$.

Solution:	**Division**	**Quotient**	**Remainder**
	41/2	20	1 ↑
	20/2	10	0 |
	10/2	5	0 |
	5/2	2	1 |
	2/2	1	0 |
	1/2	0	1 |

The remainders have to be arranged in the reverse order.

Hence, $41_{10} = 101001_2$.

Example 7. $623_{10} = ?_{16}$.

Solution:	**Division**	**Quotient**	**Remainder**
	623/16	38	15(F) ↑
	38/16	2	6 |
	2/16	0	2 |

Hence, $623_{10} = 56F_{16}$.

Example 8. $5012_{10} = ?_8$.

Solution:	**Division**	**Quotient**	**Remainder**
	5012/8	624	0 ↑
	624/8	78	0 |
	78/8	9	6 |
	9/8	1	1 |
	1/8	0	1 |

Hence, $5012_{10} = 116000_8$.

Example 9. $247_{10} = ?_{12}$

Solution:

Division	Quotient	Remainder
247/12	20	7 ↑
20/12	1	8 |
1/12	0	1 |

Hence, $247_{10} = 187_{12}$

Example 10. $47_{10} = ?_5$

Solution:

Division	Quotient	Remainder
47/5	9	2 ↑
9/5	1	4 |
1/5	0	1 |

Hence, $47_{10} = 142_5$

Converting From A Base Other Than 10 To A Base Other Than 10

For that the following two steps are used :

(1) First convert given number to decimal number and then,

(2) Convert decimal number obtained in step 1 to the new base.

Example 11. $314_5 = ?_4$

Solution: $314_5 = (3 \times 5^2) + (1 \times 5^1) + (4 \times 5^0)$

$= 3 \times 25 + 1 \times 5 + 4 \times 1$

$= 75 + 5 + 4$

$= 84_{10}$

Now in required base

Division	Quotient	Remainder
84/4	21	0 ↑
21/4	5	1 |
5/4	1	1 |
1/4	0	1 |

Hence, $314_6 = 84_{10} = 1110_4$

Example 12. $3AB_{15} = ?_{11}$

Solution: $3AB_{15} = (3 \times 15^2) + (10 \times 15^1) + (11 \times 15^0)$

$= 3 \times 225 + 10 \times 15 + 11 \times 1$

$= 675 + 150 + 11$

$= 836_{10}$

Now in required base

Division	Quotient	Remainder
836/11	76	0 ↑
76/11	6	10(A) |
6/11	0	6 |

Hence, $3AB_{15} = 836_{10} = 6A0_{11}$

Shortcut Method For Binary To Octal Conversion

It is performed using following method:

(1) Starting from the right form the given binary into group of three digits if leftmost group has fewer bits, attach the required number of leading 0s to complete this group.

(2) Determine equivalent one octal digit for each group.

Example 13. $100110101_2 = ?_8$

Solution:

Step 1 100 110 101

Step 2 100_2

$= (1 \times 2^2) + (0 \times 2^1) + (0 \times 2^0)$

$= 4 + 0 + 0 = 4_8$

110_2

$= (1 \times 2^2) + (1 \times 2^1) + (0 \times 2^0)$

$= 4 + 2 + 0 = 6_8$

101_2

$= (1 \times 2^2) + (0 \times 2^1) + (1 \times 2^0)$

$= 4 + 0 + 1 = 5_8$

Hence, $100110101_2 = 465_8$

Shortcut Method For Octal To Binary Conversion

It is performed using following method:

(1) First find out equivalent binary group (3 digit) for each octal digit and, then

(2) Target binary number obtained by combining binary groups obtained in first step.

Example 14. $623_8 = ?_2$

Solution:

$6_8 = 110_2$

$2_8 = 010_2$

$3_8 = 011_2$

110 010 011

Hence, $623_8 = 110010011_2$

Shortcut Method For Binary To Hexadecimal Conversion

It is performed using following method:

(1) Starting from the right form the given binary into group of four digits if leftmost group has fewer bits, attach the required number of leading 0s to complete this group.

(2) Determine equivalent one hexadecimal digit for each group.

$10011011 = ?_{16}$

$\underline{1001}\ \underline{1011}$

$1001_2 = (1 \times 2^3) + (0 \times 2^2) + (0 \times 2^1) + (1 \times 2^0)$

$= 8 + 0 + 0 + 1 = 9_{16}$

$1011_2 = (1 \times 2^3) + (0 \times 2^2) + (1 \times 2^1) + (1 \times 2^0)$

$= 8 + 0 + 2 + 1 = 11_{10} = B_{16}$

Hence, $10011011_2 = 9B_{16}$.

Shortcut Method For Hexadecimal To Binary Conversion

It is performed using following method:

(1) First find out equivalent binary group (4 digit) for each hexadecimal digit and, then

(2) Target binary number obtained by combining binary groups obtained in first step.

$D3B_{16} = ?_2$

$D_{16} = 13_{10} = 1011_2$

$3_{16} = 3_{10} = 0011_2$

$B_{16} = 11_{10} = 1011_2$

$2AB_{16} = \underline{1011}\ \underline{0011}\ \underline{1011}$

Hence, $2AB_{16} = 001010101011_2$.

Fractional Numbers

In all other number system, fractional numbers are formed in the same general way as n the decimal system.

For example,

$$.357 = (3 \times 10^{-1}) + (5 \times 10^{-2}) + (7 \times 10^{-3})$$

and

$$29.41 = (2 \times 10^1) + (9 \times 10^0) + (4 \times 10^{-1}) + (1 \times 10^{-2})$$

Procedure explained in this example can be used for all other systems.

Other Base System To Decimal Number System

Example 15. $32.21_4 = ?_{10}$

Solution : $32.21_4 = (3 \times 4^1) + (2 \times 4^0) + (2 \times 4^{-1}) + (1 \times 4^{-2})$

$= 12 + 2 + 2/4 + 1/16$

$= 12 + 2 + 0.50 + 0.0625 = 14.5625_{10}$

Example 16. $562.13_9 = ?_{10}$

Solution: $562.13_9 = (5 \times 9^2) + (6 \times 9^1) + (7 \times 9^0) + (1 \times 9^{-1}) + (3 \times 9^{-2})$

$= 5 \times 81 + 6 \times 9 + 7 \times 1 + 1/9 + 3/81$

$= 405 + 54 + 7 + 0.111 + 0.0370$

$= 466.148$

Decimal To Other Base System

That conversion is performed using following method:

Convert non-fractional part into decimal equivalent as we convert integer number and then Convert fractional part into decimal equivalent. Finally combines results of both steps.

Process for conversion of fractional part into integer. Multiply it with new base number, and then Save non-fractional part and again multiply obtained fractional part with base. Repeat this process either 0 fractional part is obtained or sufficient no. of times(if large no repetition required to obtain 0 fractional part). Finally combine saved non-fractional part in above steps.

Example 17. $0.11_2 = ?_{10}$

Solution:

$0.11_2 = 0.11 \times 2 = 0.22$	0
$= 0.22 \times 2 = 0.44$	0
$= 0.44 \times 2 = 0.88$	0
$= 0.88 \times 2 = 1.76$	1
$= 0.76 \times 2 = 1.52$	1
.....	
.....	

Hence $0.11_2 = 0.00011_{10}$

Relationship between Decimal, Binary, Hexadecimal and Octal Number Systems

Decimal	Hexadecimal	Binary	Octal
0	0	0	0
1	1	1	1
2	2	10	2
3	3	11	3
4	4	100	4
5	5	101	5
6	6	110	6
7	7	111	7
8	8	1000	10
9	9	1001	

10	A	1010	
11	B	1011	
12	C	1100	
13	D	1101	
14	E	1110	
15	F	1111	
16	10	10000	

□□□

4 BINARY ARITHMETIC

INTRODUCTION

Method for performing arithmetic operations in binary system is same as decimal system.

Binary Addition

Rule for binary addition is as follows:

0 + 0 = 0
0 + 1 = 1
1 + 0 = 1
1 + 1 = 0 (with a carry of 1)

Since 1 is the largest digit in the binary system, any sum greater than 1 requires that a digit be carried over. Carry-over is performed in the same way as in decimal arithmetic.

Example 1.

```
Carry      1   1
          110101
        + 10001
        -------
         1000110
```

Example 2.

```
Carry      1
         1010101
       + 101001
       --------
        1111110
```

Example 3.

```
Carry     1111
            11011
          + 01111
          -------
           101010
```

Example 4.

```
Carry    1111111
          1011111
        + 1111111
        ---------
         110111110
```

Binary Subtraction

Consideration for performing binary subtraction is same as decimal system. Firstly compare the subtrahend (number to be subtracted) with minuend (number from which other number subtracted). If subtrahend is lesser then subtraction is performed. If subtrahend is greater, then borrow is required from left column. As in decimal 10 is borrowed, in binary 2 is borrowed.

Rules for binary subtraction is as follows:

$0 - 0 = 0$

$0 - 1 = 1$ (with a borrow of 1)

$1 - 0 = 1$

$1 - 1 = 0$

Example 5.

Borrow

$$\begin{array}{r} 110101 \\ -\ 10001 \\ \hline 100100 \end{array}$$

Example 6.

Borrow 202

$$\begin{array}{r} 1010101 \\ -\ 101001 \\ \hline 101100 \end{array}$$

Example 7.

Borrow 012

$$\begin{array}{r} 10011 \\ -\ 01111 \\ \hline 00100 \end{array}$$

Example 8.

Borrow 012

012

$$\begin{array}{r} 10101001 \\ -\ 10011111 \\ \hline 00001010 \end{array}$$

Additive Method of Subtraction

This method of subtraction is known as complementary subtraction.

It involves the following steps:

(1) Computing the complement of subtrahend;

(2) Then add result of step1 to the minuend;

(3) If a carry is obtained, add it to obtain the result;

(4) If no carry is obtained, recomplement the sum and attach a negative sign to obtain the result.

Example 9. *Subtract 10001_2 from 11001_2 using additive approach.*

Solution: Complement of $10001_2 = 01110_2$

$$\begin{array}{rl} & 11001_2 \\ + & 01110_2 \\ \hline & 00111 \quad \text{(with carry of 1)} \\ + & \quad\ 1 \\ \hline \text{Result} = & \quad 01000 \end{array}$$

Example 10. *Subtract 1001_2 from 0101_2 using additive approach.*

Solution: Complement of $1001_2 = 0110_2$

$$\begin{array}{rl} & 0101_2 \\ + & 0110_2 \\ \hline & 1011_2 \quad \text{(with no carry)} \\ \hline \text{Result} = & -\ 0100_2 \\ \hline \end{array}$$

Binary Multiplication

Rules for performing binary multiplication is as follows:

$0 \times 0 = 0$

$0 \times 1 = 0$

$1 \times 0 = 0$

$1 \times 1 = 1$

From observing rules it is clear that copy digit when multiplied with one otherwise put 0.

Example 11.

```
   1110
 × 1011
------
   1110
  1110×
 0000××
1110×××
--------
10011010
```

Example 12.

```
   11101
 × 1100
-------
   00000
  00000×
 11101××
11101×××
--------
101011100
```

Additive Method of Multiplication

Performing multiplication using additive approach is simpler but lengthy. Mostly computers use this approach due to their high speed to perform operations.

Example 13.

```
  1001
 × 11
------
  1001
 1001×
------
 11011
```

Binary Division

Rules for performing binary division is as follows:

$0/1 = 0$

$1/1 = 1$

The Method for binary division is as follows:

(1) Starting from left compare divisor with dividend.

(2) If dividend is greater, take value of the quotient 1 and subtract the divisor from the corresponding digits of dividend.

(3) If divisor is greater, take value of the quotient 0 and repeat whole process till sufficient digits in dividend.

Example 14.

```
                1010        (Quotient)
(Divisor) 101 ) 110011      (Dividend)
                101
                -----
                00101
                  101
                -----
                 0001       (Remainder)
```

Example 15.

```
                01110       (Quotient)
(Divisor) 111 ) 1100111     (Dividend)
                111
                -----
                 1011
                  111
                -----
                  1001
                   111
                -----
                  00101     (Remainder)
```

Additive Method of Division

Method can be understood from this example.

Example 16. *Divide* 10001_2 *by* 110_2.

Solution : 10001 – 110 = 01011 — 1

1011 – 110 = 0101 (Remainder) — 1

10 (quotient)

REPRESENTATIONS OF CHARACTER

BCD Code

The binary coded decimal(BCD) code is one of the early memory codes. It is based on the idea of converting each digit of a decimal number into its binary equivalent rather than converting the entire decimal value into a pure binary form. In the BCD form 4 bits each represent all decimal digits. When only 4-bits are used a total of 24 *i.e.,* 16 configurations are possible.

In BCD format to represent characters 6 bits are used. In the 6-bit code the four BCD numeric place position (1,2,4 and 8) are retained, but two additional Zone Positions are used in combination with the numeric bits to represent alphabetic and special positions are used in combination with the numeric bits to represent alphabetic and special characters. When only 6 bits are used a total of 26 *i.e.,* 64 configurations are possible, means 64 different characters can be represented. These are sufficient to code 10 decimal digits, 26 alphabets and other 28 special characters.

Alphabetic And Numeric Characters IN BCD Along With Their Octal Equivalent

Character	BCD Code		Octal Equivalent
	Zone	Digit	
A	11	0001	61
B	11	0010	62
C	11	0011	62
D	11	0100	64
E	11	0101	65
F	11	0110	66
G	11	0111	67
H	11	1000	70
I	11	1001	71
J	10	0001	41
K	10	0010	42
L	10	0011	43
M	10	0100	44
N	10	0101	45
O	10	0110	46
P	10	0111	47
Q	10	1000	50
R	10	1001	51
S	01	0001	22
T	01	0010	23
U	01	0011	24
V	01	0100	25
W	01	0101	26
X	01	0110	27

Y	01	1000	30
Z	01	1001	31
1	00	0001	01
2	00	0010	02
3	00	0011	03
4	00	0100	04
5	00	0101	05
6	00	0110	06
7	00	0111	07
8	00	1000	10
9	00	1001	11
0	00	1010	12

EBCDIC

In the BCD code only 64 characters can be represented, which are not sufficient to provide decimal numbers (10), lower case letters (26), capital letters (26) and large number of other special characters. Hence , the BCD code was extended from a 6-bit code to an 8-bit code. This coding scheme is called as EBCDIC for Extended binary coded decimal interchange code.

In this code, it is possible to represent 256(2^8) different characters instead of 64 (2^6). Because EBCDIC is an 8-bit code, it can be easily divided into two 4-bit groups. Each of these 4-bit groups can be represented by 1 hexadecimal digit.

Alphabetic And Numeric Characters in EBCDIC Along With Their Hexadecimal Equivalent

Character	EBCDIC Code		Hexadecimal Equivalent
	Zone	Digit	
A	1100	0001	C1
B	1100	0010	C2
C	1100	0011	C3
D	1100	0100	C4
E	1100	0101	C5
F	1100	0110	C6
G	1100	0111	C7

H	1100	1000	C8
I	1100	1001	C9
J	1101	0001	D1
K	1101	0010	D2
L	1101	0011	D3
M	1101	0100	D4
N	1101	0101	D5
O	1101	0110	D6
P	1101	0111	D7
Q	1101	1000	D8
R	1101	1001	D9
S	1111	0001	E2
T	1111	0010	E3
U	1111	0011	E4
V	1111	0100	E5
W	1111	0101	E6
X	1111	0110	E7
Y	1111	1000	E8
Z	01	1001	E9
0	1111	0000	F0
1	1111	0001	F1
2	1111	0010	F2
3	1111	0011	F3
4	1111	0100	F4
5	1111	0101	F5
6	1111	0110	F6
7	1111	0111	F7
8	1111	1000	F8
9	1111	1001	F9

ASCII

Another computer code that is very widely used is the American Standard Code for Information Interchange (ASCII).

ASCII is of two types :

1. ASCII-7
2. ASCII-8

ASCII-7 is a 7-bit code that allows 128 (2^7) different characters. The first 3 bits are used as zone bits and the last 4 bits indicate the digit. Microcomputers using 8-bit byte (group of 8 bits for one byte) use the 7-bit ASCII by leaving the leftmost first bit of each byte as a zero.

ASCII-8 is an extended version of ASCII-7. It is an 8-bit code that allows 256 (2^8) different characters rather than 128.

Alphabetic And Numeric Characters in ASCII Along With Their Hexadecimal Equivalent

Character	ASCII	Code	Hexadecimal Equivalent
	Zone	Digit	
0	011	0000	30
1	011	0001	31
2	011	0010	32
3	011	0011	33
4	011	0100	34
5	011	0101	35
6	011	0110	36
7	011	0111	37
8	011	1000	38
9	011	1001	39
A	100	0001	41
B	100	0010	42
C	100	0011	43
D	100	0100	44
E	100	0101	45
F	100	0110	46
G	100	0111	47

H	100	1000	48
I	100	1001	49
J	100	1010	4A
K	100	1011	4B
L	100	1100	4C
M	100	1101	4D
N	100	1110	4E
O	100	1111	4F
P	101	0000	50
Q	101	0001	51
R	101	0010	52
S	101	0011	53
T	101	0100	54
U	101	0101	55
V	101	0110	56
W	101	0111	57
X	101	1000	58
Y	101	1001	59
Z	101	1010	6A

REPRESENTATION OF INTEGERS IN COMPUTER

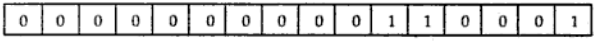
Computer can only work with binary numbers. So anything which has to be tored in the memory must be converted to a binary form and then the bits(bit pattern) can be used.

Storing positive integers in a memory location:

To store any positive integer the location is being filled with bits from right from to left. The extra bits are filled up with 0s.

i.e., to store 25_{10} in a memory location of word length of 16 bits.

The binary equivalent 25_{10} is 110001.



The following table shows the largest positive integer which can be stored in words having different word lengths.

Word Length	Largest Positive Integer	Bit Pattern
2	$2^2 - 1 = 3$	11
4	$2^4 - 1 = 15$	1111
8	$2^8 - 1 = 255$	11111111
16	$2^{16} - 1 = 65535$	1111111111111111

Storing signed integers in a memory location:

There are three representations possible for a signed integer

(*a*) Sign Magnitude

(*b*) 1's Complement

(*c*) 2's Complement

Sign Magnitude Representation

The left most bit in the word to represent the sign. If the leftmost bit is 0 then the number stored in the word will be treated as a positive integer. On other hand if left most bit is 1 then the number treated as a negative number. That bit is called as sign bit and the remaining part of the word is called Magnitude.

i.e., +48 has to be stored in a 16-bit word

0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0

i.e., –48 has to be stored in a 16-bit word

1	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0

In the sign magnitude representation, the number of bits used for store the magnitude will be one less than the word length of the word. Therefore the maximum magnitude will be when all these bits are 1.

If the sign bit is 0(+ve) and the magnitude bits are all 1s then it is the maximum value, which can be represented in the word using the sign magnitude representation.

If the sign bit is 1(–ve) and the magnitude bits are all 1s then it is the minimum value, which can be represented in the word using the sign magnitude representation.

Therefore the range of values, which can be represented in an n bit word, using the sign magnitude is -2^{n-1} through 0 to $+ 2^{n-1} - 1$.

1's Complement Representation

In the 1's complement representation, the representation of positive numbers is identical to that used in the sign magnitude system. It consists of a left most sign bit in the leftmost position, followed by the magnitude bits. For a positive number two representations are identical.

In case of negative numbers, the magnitude bits are represented by their one's complement. A one's complement of a number is obtained by inverting each bit, including the sign bit,

i.e., 25_{10} , its 1's complement is $1\ \ 00110_2$

i.e., –5 is represented by 1010_2

In general, for n bit number x (excluding the sign bit) the 1's complement is given by $(2^n - 1 - x)$. Thus, –14 is represented by 1111 – 1110 = 0001, excluding the sign bit.

2's Complement Representation

2's complement representation of a –ve number is obtained by adding a 1 to the 1's complement of that number. Thus the 2's complement representation of – 5 is 1011. Another method of obtaining a two's complement of a binary number is to scan the number from right to left and complement all the bits appearing after the first appearance of a 1. Thus, the 2's complement of 0, 1110 is obtained as 1,0010.

REPRESENTATION OF REAL NUMBER IN COMPUTER SYSTEM

A real number can be divided into two parts, an integer part and a fractional part. The number can be signed or unsigned. In the computer decimal point is not stored, a position is assumed for it. Since the decimal point is not actually stored it is referred to as the assumed decimal point.

Two most commonly used representations are :

(*a*) Fixed Point Representations

(*b*) Floating Point Representations

Fixed Point Representation

In this the position of the decimal point is fixed. The position of the decimal point is assumed to be after the first 8 bits. Therefore the first 8 bits store the integer portion of the number and the last eight bits store the fractional part.

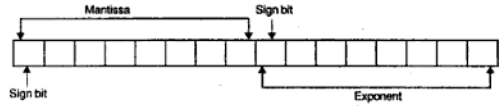
The bits in integer part are filled from right to left and the remaining bits (in the integer part) are padded with 0's. If the real number which is to be stored has a sign then the left most bit will be reserved for storing the sign bit. The representation of -101001011.001001_2 in the 16 bit word will be as follows :

1	1	0	1	0	0	1	0	1	1	0	0	1	0	0	1

Floating Point Representation

The floating point representation can be used to represent numbers in any positional number system.

It requires that the number be written in the format $m*b^e$. Here m is the mantissa, e the exponent and b is the base of the number being represented using floating point notation. The mantissa is a signed fixed-point number. The exponent is a signed integer, which determines the actual position of the decimal point. In this representation mantissa and exponent have their own signs.



Range of values in fixed-point representations:

The maximum value can be obtained by the general formula $+[2^{ni} - 1 . (1 - 2^{nf})]$, where *ni* is the number of bits in the integer part and *nf* is the number of bits in the fractional part.

The minimum value can be obtained by the general formula $-[2^{ni} - 1 . (1 - 2^{nf})]$, where *ni* is the number of bits in the integer part and *nf* is the number of bits in the fractional part.

Range of values in floating-point representations:

The maximum value can be obtained by the general formula $[(1 - 2^{-nm}) * 2 \wedge (2^{ne} - 1)]$ m, where *nm* is the number of bits in the integer part and *ne* is the number of bits in the fractional part.

The minimum value will be when all the bits in the mantissa are 1 and all the bits sign bits of the mantissa is 1(negative).

The *r*'s complement

The *r*'s complement of a positive numbers N is defined as $r^n - N$, where

r is the base.

n is the number of digits in the integer part of the number N.

Example :

The 10's complement of $231_{10} = 10^3 - 231 = 769$.

The 10's complement of $125_{10} = 10^3 - 125 = 875$.

The $(r - 1)$'s complement

The $(r - 1)$'s complement of a positive number N is defined as $r^n - r^{-m} - N$, where

r is the base

n is the number of digits in the integer part

m is the number of digits in the fractional part.

Example :

The 9's complement of $231_{10} = 10^3 - 10 - 231 = 769$.

The 9's complement of $125_{10} = 10^3 - 125 = 825$.

□□□

5 ALGORITHMS AND FLOWCHART

ALGORITHMS

An algorithm is a well-defined set of steps, if followed in sequence provide solution of specific problem.

Characteristics of an algorithm

- ✔ An algorithm should have zero or more input.
- ✔ An algorithm should exhibit at least one output.
- ✔ An algorithm should be finite means if executed it should be finished in finite number of steps.
- ✔ An algorithm should be definite means each instruction in an algorithm should be defined clearly such as $x + 2\ 3$ is meaningless instruction.
- ✔ An algorithm should be effective means each instruction used in algorithm should be basic and easy to perform.

Representing an Algorithm

An algorithm can be represented in many ways. It can be represented in natural language like English or graphical tool flowchart but they work well only if the algorithm is small and simple. Another way used for representing an algorithm is known as pseudo-code.

Pseudo-code Conventions

(*i*) An identifier begins with a letter. Data types of variables aren't explicitly defined.

(*ii*) Assignment of values to variables

Variable: = expression

(*iii*) Input and output are done using the instruction read and write. No format is specified for size of input and output.

(*iv*) Conditional statements has the following form

If <condition> then <stmt>

If <condition> then <stmt> else <stmt>

Here condition is Boolean expression if it is true then if stmt executed otherwise else stmt executed if exists.

(*v*) Looping statements:

while loop:

```
while<condition> do
{
    stmt 1
    .
    .
    .
    stmt n
}
```

Loop is executed as long as condition defined is true and finishes when it becomes false.

for loop:

```
for variable:= val1 to val2 step val3 do
    {
        stmt 1
        .
        .
        .
        stmt n
    }
```

Here, val1 , val2 are arithmetic expressions and type of variable can be real or integer. Part step val 3 is optional and if taken have form of +1 or –1

repeat-until

```
repeat
    stmt 1
        .
        .
    stmt n

until(condition)
```

Condition is computed after execution of statements if true then repetition occurs otherwise not.

(*vi*) Logical operator and, or and not and relational operator, $<, \leq, <, \geq, \neq, =$ are provided.

(*vii*) Elements of an array accessed using square [] brackets,

(*viii*) Comments are denoted using symbol // for single line.

An algorithm consists of a heading and body. The heading of an algorithm takes the following format:

Algorithm name (<parameter list>

Where name is the name of the process and <parameter list> is a list of process parameters. The body has one or more statements enclosed within braces { and }.

An algorithm to calculate average of ten numbers

Average(sum, avg)

Step 1: Begin

Step 2: set sum:=0 and avg:=0.0
Step 3: for I:=1 to 10 do
Step 4: read a
Step 5: sum:= sum+a(end of for loop)
Step 6: avg:= sum/10
Step 7: Write avg
Step 8: End

An algorithm to interchange two numbers

Interchange(A,B)
Step 1: Begin
Step 2: Set A=A+B
Step 3: Set B=A-B
Step 4: Set A=A-B
Step 5: Write A,B
Step 8: End

An algorithm to find out number and sum of digit of a number

Interchange(N)
Step 1: Begin
Step 2: Set sum=0,C=0
Step 3: while N>0
Step 4: Set x=N%10
Step 5: sum=sum+x
Step 6: N=N/10
Step 7: C=C+1
Step 8: Write number of digit =C and sum of digits of a number=sum
Step 9: End

Algorithm to perform sorting of n elements

SortingN(a,n)
Step 1: Begin
Step 2: for I:=1 to 10 do
Step 3: for j:=1 to 10 do
Step 4: if(a[j]>a[j+1])
Then interchange them.
(end of for loop)
(end of for loop)
Step 5: End

FLOWCHART

It is a graphical tool for representing the defined solution of a problem. It uses different symbols to represent different terminology.

Different Symbols Used In Flowcharts

(*i*) **Terminal:** It is first and last symbol in the flowchart. It is a symbol used to indicate starting, ending of program flow having oval shape.

(*ii*) **Input/Output:** Symbol used to denote input/output function of any input/output device. It has shape of parallelogram.

(*iii*) **Process:** It is represented by rectangle to indicate processing.

(*iv*) **Flow direction:** It uses arrow to indicate flow of program.

(*v*) **Decision:** The diamond represents a decision-making operation in program.

(*vi*) **Connector:** Circle used to represent link between parts of flowchart, if flow chart is large and unfit in a single page.

(*vii*) **Predefined process:** Double-sided rectangle used to represent any predefined process used in the program. That process is defined at elsewhere.

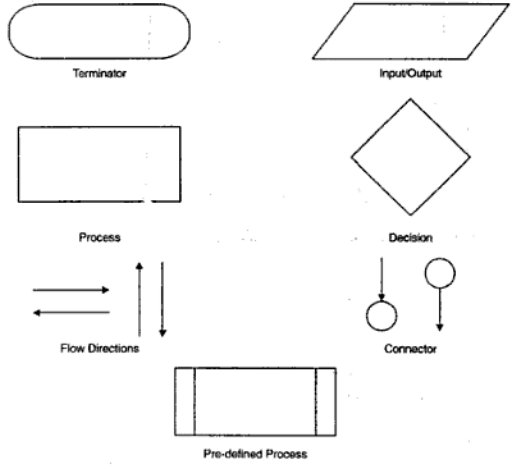


Figure 5.1. Different symbols used in flowchart.

Making a flowchart requires lots of practice.

Rules for drawing a flowchart:

- ✔ It should contain only one start and one end symbol.
- ✔ The direction of arrows should be top to bottom and left to right.
- ✔ It should be drawn clearly and neatly means crossing of lines should be avoided and symbols should be drawn having exact shape.
- ✔ It should contain sufficient details not much.
- ✔ It should be as simple as possible.
- ✔ The branches of decision box must be labelled.

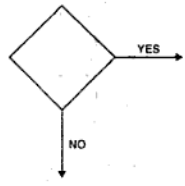


Figure 5.2.

- ✔ Only standard symbols should be used.
- ✔ The language in the symbol should not be computer dependent language.

Advantages of flowchart

- ✔ Provides convenient method to understand the solution.
- ✔ Good tool for documentation
- ✔ Provide guide for coding.

Disadvantages of flowcharts

- ✔ Not suitable for large problems
- ✔ Modification in logic isn't easy
- ✔ Time consuming

Example 1. *Let us consider a simple flowchart to convert temperature given in fahrenheit to celsius.*

Solution:

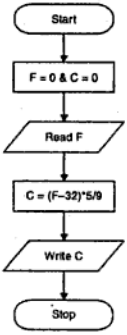


Figure 5.3.

Example 2. *Draw a flowchart to convert days into month and days.*

Solution:

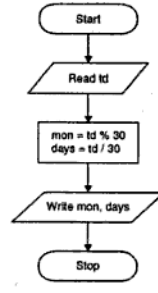


Figure 5.4.

Example 3. *Draw a flowchart to calculate the simple interest.*

Solution:

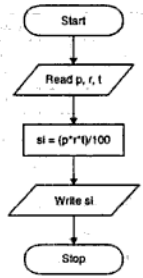


Figure 5.5.

Example 4. *Draw a flowchart to interchange the value of two numbers.*

Solution:

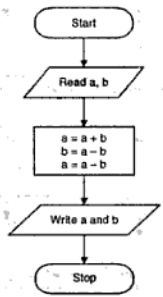


Figure 5.6.

Example 5. *Draw a flowchart of the given problem read marks of four subjects and print grade of the student according to total marks obtained.*

Solution:

Total Marks	Grade
Above 800	A
601–800	B
401–600	C
201–400	D

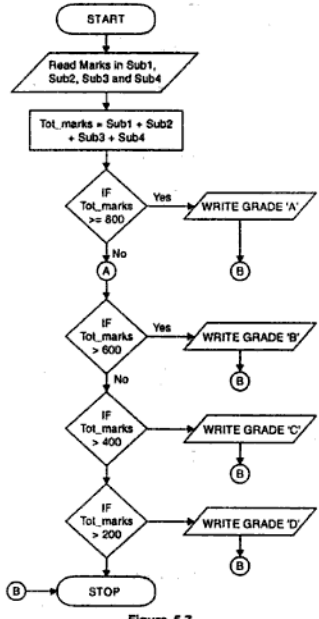


Figure 5.7.

Example 6. *Draw a flowchart to calculate a factorial of a number.*

Solution:

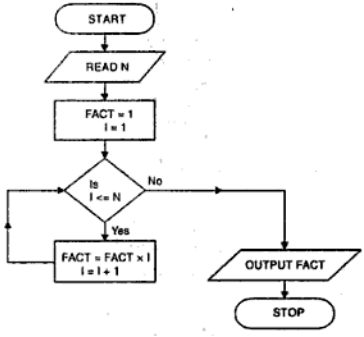


Figure 5.8.

Example 7. *Draw a flowchart to read two number and find out highest common factor.*

Solution :

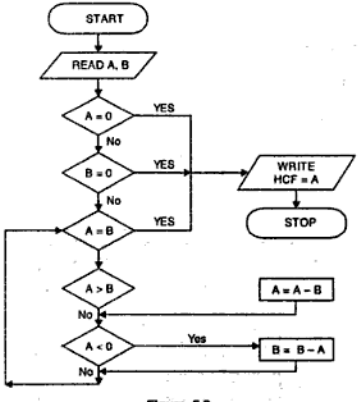


Figure 5.9.

Example 8. *Draw a flowchart to read a number and reverse it.*
Solution:

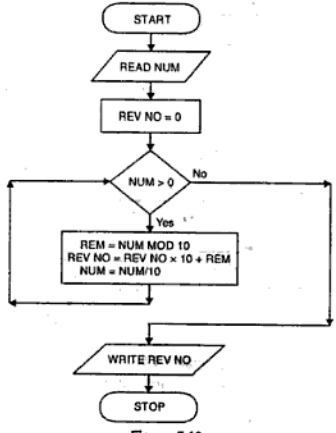


Figure 5.10.

Example. 9. *Draw a flowchart to generate the series* $1^2 + 2^2 + 3^2 + \ldots\ldots\ldots + N^2$.
Solution:

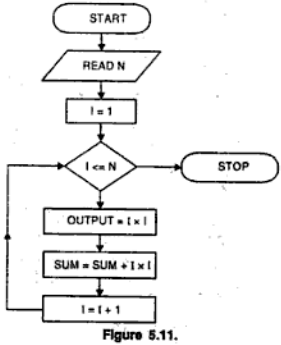


Figure 5.11.

Example 10. *Draw a flowchart to check whether a number is Armstrong or not.*

Solution:

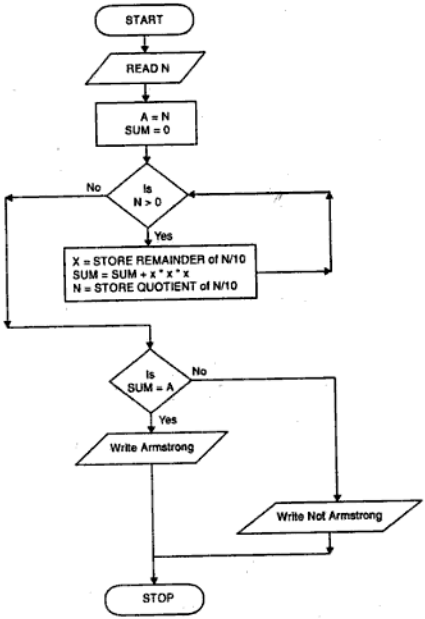


Figure 5.12.

Example 11. *Draw a flowchart to display the following pattern for n lines.*

Solution:

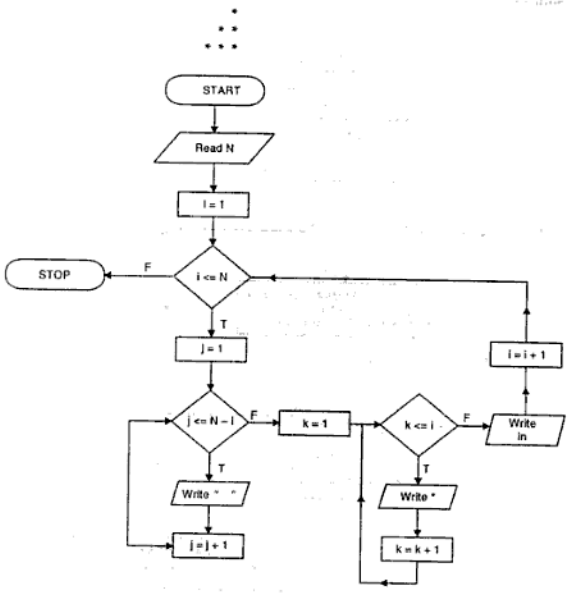


Figure 5.13.

Example 12. *Draw a flowchart to check whether a number is prime or not.*

Solution:

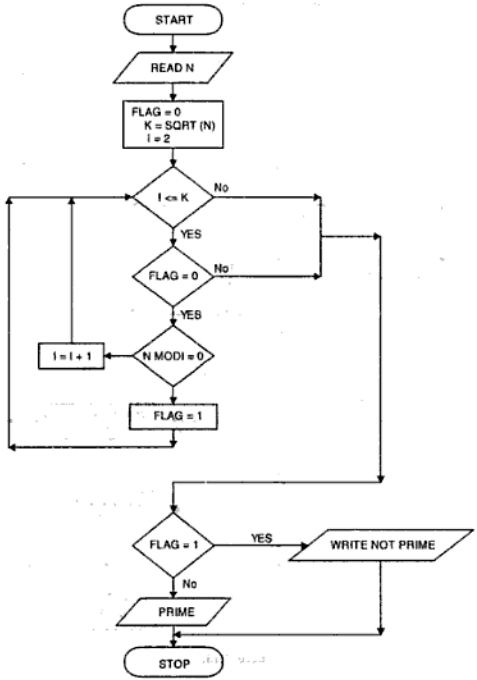


Figure 5.14.

Example 13. *Draw a flowchart to perform a linear search in an array of size N.*

Solution:

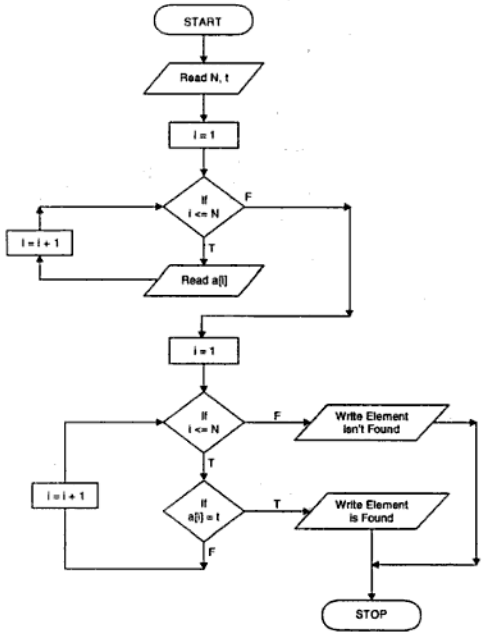


Figure 5.15.

Example 14. *Draw a flowchart to insert a element in a array of size N at given position.*

Solution:

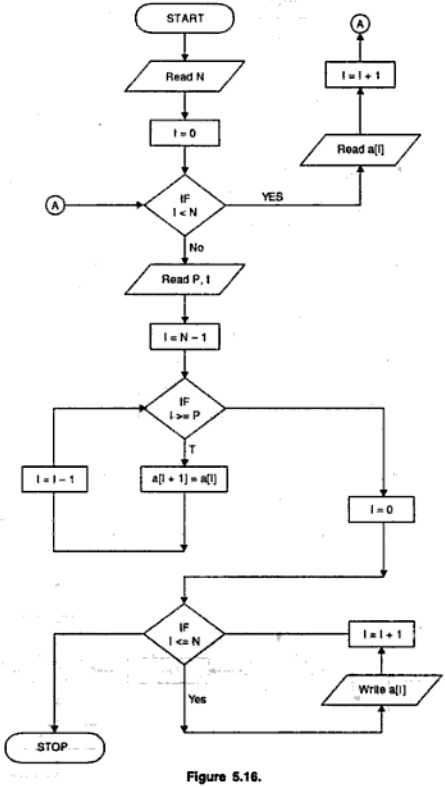


Figure 5.16.

Example 15. *Draw a flowchart to delete an element from an array of size N.*

Solution:

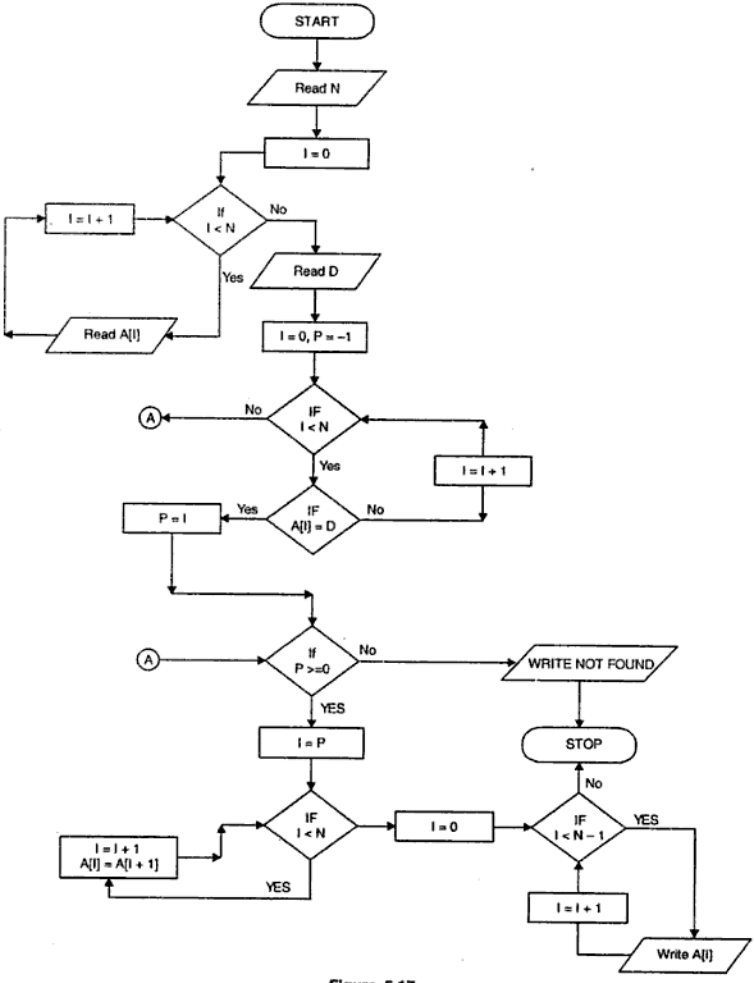
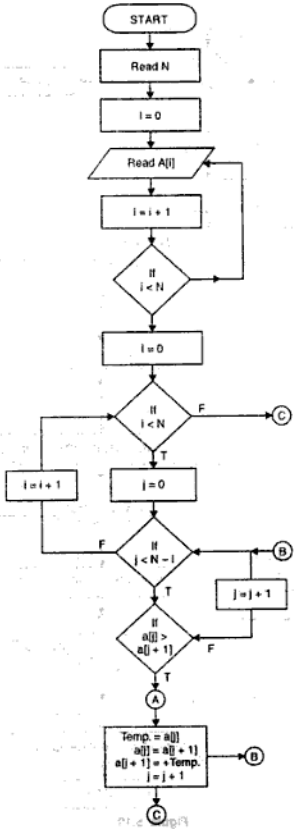


Figure 5.17.

Example 16. *Draw a flowchart to sort an element using a bubble sort method.*

Solution:



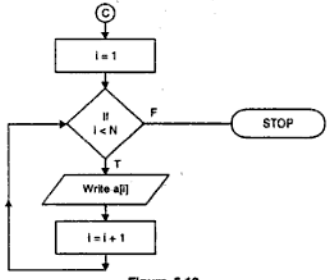


Figure 5.18.

Example 17. *Draw a flowchart to count the no words in a given string.*
Solution:

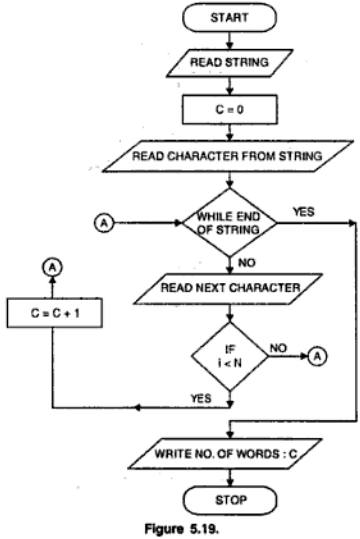


Figure 5.19.

Example 18. *Draw a flowchart to reverse a string.*

Solution:

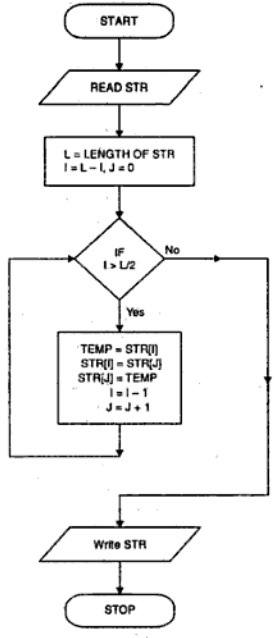


Figure 5.20.

Example 19. *Draw a flowchart to calculate length of string.*

Solution:

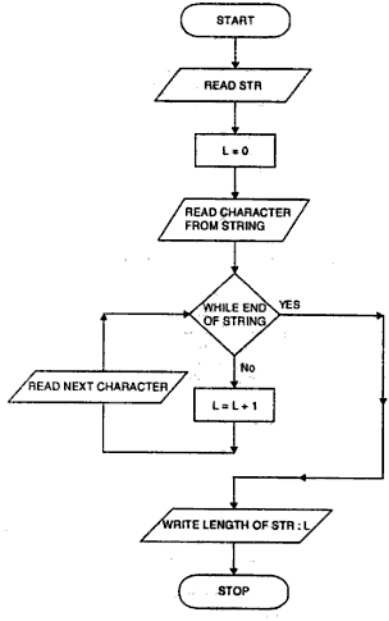


Figure 5.21.

□□□

6 COMPUTER LANGUAGES

MACHINE LANGUAGE

A language that is directly understood by the computer without any translation is called machine language. A machine language is a string of binary 0s and 1s. Since the entire circuitry of a computer is designed in such a way that it understands the machine language instruction and converts them into electrical signal required to run the computer. Therefore all the computer instructions are coded and stored in the form of 0s and 1s in its memory. Thus a machine language program is a set of binary instructions that can be directly understood by the computer without any translations.

An instruction written in any machine language has two parts:

(*i*) Operation code

(*ii*) Operand address or location.

Advantages

- ✔ Execution speed is very fast.
- ✔ Efficient use of primary memory.
- ✔ It does not require any translation because machine code is directly understood by the CPU.

Disadvantages

- ✔ **Machine dependent:** Since internal circuitry of different computers is different. Therefore computer needs different electrical signals to operate. The machine language is also different for different types of computer. Machine language is determined by the design of ALU, CU, size and word length of the memory unit of the computer.
- ✔ **Difficult to write program:** It is very difficult to write a program in machine language because it requires memorizing dozens of codes for the different commands.
- ✔ **Error prone:** To write a program in machine language, the programmer must remember operation codes (opcodes) and he has to keep a track of the storage locations of data and instructions. This causes error in programming.
- ✔ **Difficult to modify:** It is very difficult to modify the machine language programs because locating the errors is very tedious and difficult job.

HIGH-LEVEL LANGUAGE

We know that writing a program in machine language or assembly language is very difficult because it requires a deep knowledge of computer hardware and memorizing a long list of machine codes is a very laborious job. The programmer should concentrate mainly on logical analysis of the problem rather than hardware structure. To overcome these problems the high level languages were developed.

High-level languages are generally problem oriented and machine independent. It does not use mnemonic codes.

Examples : COBOL, C, PASCAL, FORTRAN etc.

Advantages

- ✔ Readability.
- ✔ High-level languages programs are easy to read and learn than machine or assembly language program because HLL programs are very similar to the English like languages.
- ✔ Machine independent.

  HLL programs are machine independent. A program written in high-level language can be run on different types of computer with no change or minor modifications.
- ✔ Easier to maintain.
- ✔ Insertion, deletion and modification in the HLL program are easy as compared to machine language.
- ✔ Fewer errors and easy debugging.
- ✔ In HLL programs, syntax and logical errors are easy to detect and remove.
- ✔ Lower program preparation cost.

  HLL programs can be developed in comparative small time and efforts. This leads to lower program preparation cost.

Disadvantages

- ✔ **Speed:** High-level language programs takes more time to run as compared to machine or assembly languages.
- ✔ **Memory:** High-level language programs require more main memory than machine or assembly language programs.
- ✔ **Lack of flexibility:** Some tasks cannot be performed in high level language and if done then it is very difficult to design a program for these tasks.

ASSEMBLY LANGUAGE

An assembly language is a low-level language program. For writing a program in assembly language a programmer needs the knowledge of hardware specifications.

Assembly languages use mnemonics in place of machine codes to represent operation codes. The word mnemonic is a memory aid. The use of these mnemonics helped a lot in reducing the problem of remembering all the machine codes related to various operations. This made the programming easy and efficient.

Halt, add, subtract etc. operation can be represented as HLT, ADD, SUB in assembly languages. These are called mnemonics.

Advantages

- ✔ **Easier to memorize and use:** Assembly language program are easy to use, understand and memorize because it uses mnemonic codes in place of binary codes. For example ADD, SUB, HLT are easy to understand and memorize than binary codes.
- ✔ **Easy to write input data:** In assembly language program the input data can be written in decimal number system, later on they are converted into binary from assembler.
- ✔ **Easy print out:** Assembly language programs can be printed easily. This helps the programmer to detect error made in the program.
- ✔ Better detect the error.
- ✔ Good library facility.

Disadvantages

- ✔ Machine dependent.
- ✔ Knowledge of hardware is required.
- ✔ Time consuming.

SOFTWARE

A computer is a dumb machine. It has to be given instructions for every task. The instructions for performing a particular task are given in the form of a program. A set of one or more programs to do a task is called software. These programs are written in the programming language.

Software is a collection of programs, procedures and associated documents (Manuals) to perform a specified task.

There are two types of software:

1. System Software
2. Application Software.

System Software

System software are designed to make the computer easier to use or System software are designed to use the machine resources effectively and efficiently. An example of system software is an operating system ,which consists of many other programs for controlling input/output devices.

Example of System Software:

- Operating system
- Translators
  1. Compiler
  2. Interpreter
  3. Assembler
- Editors
- Utility programs.

Application Software

Application programs are designed for specific applications, such as payroll processing, inventory control, library management. To write a program for payroll or other application the

programmer does not need to control the basic circuit of a computer. Some of the most commonly known application of software are:

(*i*) **Word_Processing software:** A word processing software enables us to make use of a computer system for creating, editing, viewing, formatting, storing and printing documents. Some of the popular word processor are MS-Word, WordStar and software.

(*ii*) **Spreadsheet software:** A spreadsheet software is a numeric data analysis tool. A spreadsheet, which can be used for dong data analysis. The worksheet can be viewed as a, matrix of rows and columns. The intersections of row and column in worksheet are called a cell. These packages are very useful for analyzing budgets, cash flow, profitability and many similar problems. Some of the popular spreadsheet packages are MS–Excel, Lotus 123.

(*iii*) **Graphics software:** A graphics software enables us to use a computer system for creating editing viewing ,storing and printing design drawing, pictures, graphs and anything else that can be drawn in the traditional manner.

(*iv*) **Personal assistance software:** A personal assistance software allows us to use personal computers for storing and retrieving our personal information and planning and managing our schedules contacts, financial and inventory of important items.

(*v*) **Education software:** Education software allows a computer system to be used as a teaching and learning tool.

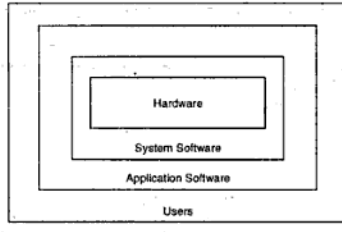


Figure 6.1.

(*vi*) **Entertainment software:** Entertainment software allows a computer system to be used as an entertainment tool. A good example of such an application is a computer video games.

(*vii*) **Database management systems:** A computer can be used for managing large volumes of data. A typical organization may need to maintain databases on customers suppliers, employees etc. on the computer.

TRANSLATOR

Translator is system software and used to perform translation of high-level language to low level language. Low-level languages are machine language and assembly language. All

programming languages consider as high-level language *i.e.*, C, C++. Input to a translator is source language and output is target language. There are three types of translator:

1. Assembler;
2. Interpreter;
3. Compiler.

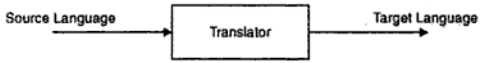


Figure 6.2.

Assembler

An assembler accepts assembly language as source language and produce machine language. It is required because computer can understand only machine language.

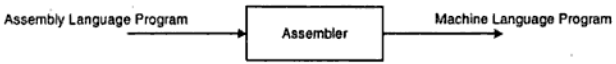


Figure 6.3.

Interpreter

The work of interpreter is same as compiler. It is also used for converting the code of high level language program into machine level language but it checks the errors of program statement by statement. After checking the one statement, it converts that statement into machine code and then executes that statement.

Compiler

A computer can understand only machine language instruction. A compiler is a translator that translates the high-level language program into machine language. Compiler translates whole program at once and produce errors. Compiled programs generally run faster than interpreted programs.

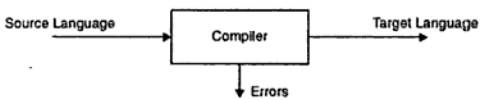


Figure 6.4.

□□□

7 INPUT OUTPUT DEVICES

INTRODUCTION

The input-output devices provide the means of communication between the computer and the outer world. They are also known as peripheral devices because they surround the CPU. Input devices are used to enter data into primary storage and output devices accept results from the primary storage.

Input Devices

Input devices are generally of two types—Online and Offline.

1. Offline: An operation that does not directly involve a computer.
2. Online: An operation that directly involves a computer.

Offline Input Devices

Input devices that allow data entry operations without the direct involvement of a computer are called off-line input devices. Generally the input entered through these device is recorded on some media first and then processed by a computer later. For example key-to- punch, floppy etc. Most of lines are now obsolete and are very rarely used.

Online Input Devices

Online input devices provide direct and interactive communication between the user and the computer. The data from that device is sent directly to the computer with no need for intermediate media. These devices are economical when the volume of data is low and irregular. For example Keyboard, Mouse, Punched cards, Light Pen, joystick etc.

Punched Cards

There are two types of punched cards, one has 80-columns and the other has 96-columns.

The 80-columns card: It is 19.3 cm in length, 9.5 cm in width, and 0.018 cm in thickness. The card is divided from left to right into 80 vertical columns numbered 1 to 80. It is again divided into 12 rows numbered 12,11,0,1,2,3,4,5,6,7,8 and 9 from top to bottom. So a single card can represent a maximum of 80 characters. The digits 0 to 9 are represented by punching just one hole in the corresponding row position. The alphabets A to Z are represented by a combination of two holes in two of the row positions. The top three rows—2,11 and 0 is zone-punching positions and the rows 0 to 9 are numeric punching positions. A logical combination of zone and numeric punches is required to represent alphabets. The coding system used in this is known as Hollerith code.

The 96-columns card: It is only one third the size of an 80-columns card. The 96-columns are separated into three 32-columns sections or ties. The upper portion of the card, which is not used for punching holes, is used as a print area. These cards have round holes instead of rectangular holes of 80-columns cards. The standard 6-bit BCD code is used instead of Hollerith code for recording data on a 96-columns card. Each of the 96-columns has six punch positions. The upper two are zone positions and the remaining four are numeric positions. The presence of a hole in a punch position indicates a 1-bit.

Keyboard

The keyboard is a universally used online input device. This input method is similar to typing on a typewriter.

The following keys are present on most keyboards.

Alphabets A–Z

Numeric digits 0–9

Delimiters ',','.',';','.','.',':'etc.

Operators '+','*','–','/' etc.

Special keys CTRL, ALT, ENTER, SHIFT etc.

Function keys F1, F2, and F3 etc.

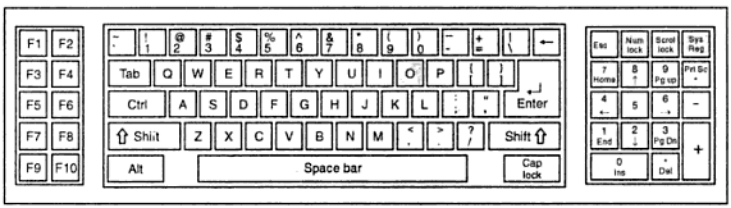


Figure 7.1. The layout of keys on a QWERTY keyboard.

Keyboards are usually available in two kinds of layout that is QWERTY and DVORAK keyboard. Most typewriters and computer keyboards are QWERTY keyboards.

The speed of input entry on a keyboard is limited by the typing speed of the operator.

Mouse

Mouse is a pointing device, which controls the position of the cursor or pointer on the screen. The mouse is a palm-size device with a ball built into the bottom. It has one or more buttons and attached to the computer by a cable.

When the user rolls the mouse across the flat surface, such as a desk, the ball on its undersides rotates. This causes the cursor to move in a corresponding direction on the monitor. If the user rolls the mouse to the left the pointer on the screen also moves to the left same as in the right side. The cursor can be moved in any of the four directions. The cursor is positioned on an appropriate object on the screen and the button on the mouse is clicked to select the object. The mouse interface is also called point and click interface.

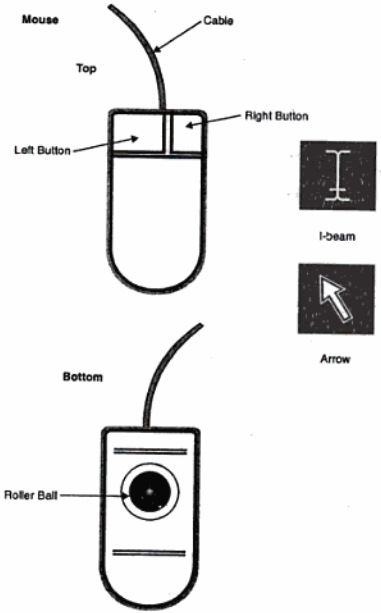


Figure 7.2.

1. Mechanical Mouse

The first mouse was a mechanical design based on a small ball that protruded through its bottom and rotated as the mouse was pushed along a surface. Switches inside the mouse detected the movement and relayed the direction of the ball's rotation to the host computer.

The mechanical mouse works on just about any surface. In general, the rotating ball has a coarse texture and is made from a rubbery compound that grips even on smooth surfaces.

The mechanical mouse requires that you move it across a surface of some kind, but all too many desks do not have enough free space to give the mouse a good run. Also, the mechanical mouse tends to pick up dirt and lint that can impede its proper operation. Clean it regularly even if you think your desktop is spotless.

2. **Optical Mouse**

The alternative technology to the mechanical mouse is the optical mouse. Instead of a rotating ball, the optical mouse uses a light beam to detect movement across a surface. No moving parts means that the optical mouse has less of a chance to get dirty or break.

Joystick

A joystick is a pointing device which works on the same principle as a trackball. Joystick has a gearshift-like lever that is used to move the pointer on the screen. The lever can be pushed in any direction. When released it returns to its original position. On most joysticks, a button on top is used to select option. Joysticks are commonly used to play games.

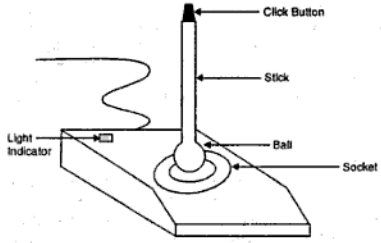


Figure 7.3. A joystick.

Touch Pad

The touch pad is a stationary pointing device that many people find less tiring to use than a mouse or a track ball. The movement of a finger across a small touch surface is translated into cursor movement on the computer screen. The touch sensitivity surface may be just 1.5–2 inch square, so the finger does not have to move much. Its size makes it most suitable for the laptops.

Light Pen

A light pen is also a pointing device having the size and shape similar to that of a regular pen. The pen consists of a photocell placed in a small tube. As the user moves the tip of the pen over the screen surface, the light pen senses the light coming from the screen area and sends a signal to the computer. Thus to identify a specific location, the light pen is very useful. But the light pen provides no information when held over a blank part of the screen because it is a passive device with a sensor only.

The light pen is primarily used for graphics work. A typical use of light pens is in drawing lines of various thickness.

Voice Input

Voice input is also used for entering data online. Voice input is relatively new technology. A microphone or telephone is used to convert human speech into electric signals .The signal patterns are then transferred to the computer where they are compared dictionary of pattern recognized by the computer. When a close match found, the computer as input recognizes word.

Scanners

Scanner is also input device and used to enter information directly into the computer.

Scanners are capable of recognizing marks or characters. The two major type of scanners are Optical scanners and Magnetic scanners.

Optical Scanners

In these scanners light source and sensors are used to recognize marks/characters. Common optical scanners are optical character readers and optical mark reader.

Optical Character Reader

This device can detect characters printed on paper. No special ink is required for that. These characters may be either type written or hand written. Handwritten characters require more care than type written character. OCR device examine each character as if it were made up of a collection of minute spots. Once a whole character has been scanned, it is compared with the machine has been programmed to recognize. Special pattern recognition software is used to recognize the character. An OCR eliminates duplication of human efforts required to input data to the computer. It also improves data accuracy.

OCR devices are expensive and are used only for large volume processing applications.

Optical Mark Reader

These scanners are capable of detecting a mark made by pencil or pen. This is generally used to check objective type questions/answers, market surveys and order forms etc.

OMR detect the mark by focusing light on the page and reflected light pattern from the marks. This method is used where one out of a few numbers of options is to be selected and marked.

Example: Which operating system is mostly used ?

- ■ Windows-98
- ❑ UNIX
- ❑ DOS
- ❑ WindowXP

Magnetic Ink Character Reader (MICR)

MICR is widely used to assist bank industry to processes large volumes of cheques and deposits written everyday. For that purpose magnetic ink is used to write character on the cheques and deposit forms, which are to be processed by an MICR. The MICR reads the character and compared with magnetized patterns stored in memory, thus identify them.

The bank identification code and the customer's account number are printed on cheques with magnetic ink. As the cheques entered the reading unit, they pass through a magnetic field, which causes the particles in the ink to become magnetized. The read head interprets these characters and send then to computer for processing. Up to 3000 cheques are processed per minute by MICR.

Bar Code Reader

Bar coding is the process of representing data in the form of bars. A bar code reader is a device used for reading bar coded data. Some bar code readers use a laser beam to read the bar codes. It is found on book stickers, cloth labels, credit cards etc. The most commonly used bar code is the Universal Product Code (UPC).

Digitizer

A digitizer is an input device , which is used for converting pictures , maps and drawing into digital form for storage in computer.

Output Devices

The output normally can be produced in two ways – either on a display unit/device or on a paper. The various types of output devices on the basis of the kind of output they produce.

- Soft copy devices
- Hard copy devices

Soft Copy Devices

That device does not produce output on paper. They offer the following advantages:

✔ No expenditure on stationary

✔ Output can be seen faster

A monitor produces soft copy output.

Hard Copy Devices

They can produce output on a paper, printer and plotter produced hard copy output. The output is permanent in nature and cannot be changed.

Printer

Printers are the most commonly output devices and used for producing output on paper. There are several types of printer, which can be classified according to the print quality and the printing speed.

Printers are basically of two types:

- Impact
- Non-impact

Impact Printers

All impact printers use a mechanical hamming device – also called a head to print each character. The head is forced against on inked ribbon to produce a sharp image on paper.

Impact printer makes a lot of noise while printing. They can produce multiple copies of a document at the same time. Impact printers are of two types according to their printing speed, that are serial and line printers.

For example drum, chain, dot matrix and daisy wheel printers.

Dot Matrix Printer

A dot matrix printer prints the character as a group of dots. They have a single head comprises a matrix of tiny needles. Mostly DMP have seven rows with nine needles in each (9 * 7 matrix). The character is printed by impressing the pins on the ribbon in the form of tiny dots. DMP can also be used to print graphics or alternate character sets.

Since dot matrix printer prints character by character therefore these are very slow in speed that vary from 40 to 250 character per second. The print quality of dot matrix printer is not very good.

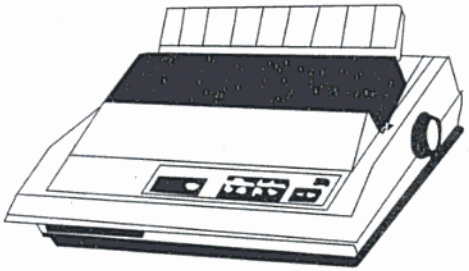


Figure. 7.4. Dot matrix printer.

Drum Printer

Drum printers are line printers that print one line at a time. It consists of a drum, which can rotate at a very high speed. On its surface characters are embossed in the form of circular bands or tracks, where each track corresponds to print position on line.

Thus a drum printer with 132 characters per line and supporting a character set of 96 characters will have altogether 12,672 (132 × 96) characters embossed on its surface.

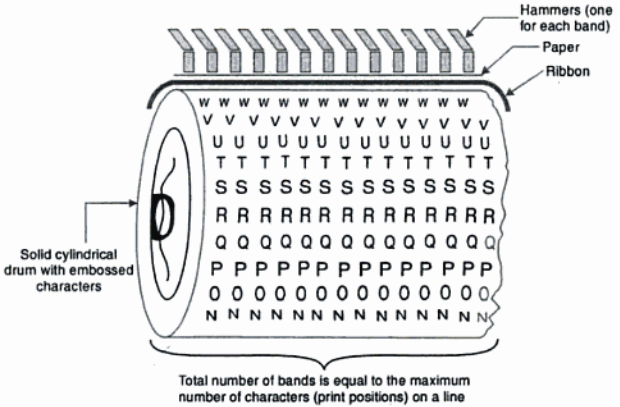


Figure 7.5. The printing mechanism of a drum printer.

It also consists of hammers or print head, one for each character position in the line, are mounted in the front of the drum. The total number of hammers is equal to the total number of print positions. The paper to print is placed between the drum and the hammers. Drum printers are expensive. The user cannot change printer drums so they have fixed character set.

Chain Printers

A chain printer consists of a metallic chain on which all the characters of the character set supported by the printer are embossed. A standard character set may have 48, 64 or 96 characters. Row of hammers/print heads is mounted in the front of the chain. The total number of hammers is equal to the total number of print positions. The chain rotates continuously on horizontal plane. The hammer corresponds to a position on the line strikes the paper when the specific character to be printed at that position comes below the hammer.

In chain printer to the character in the character set are embossed several times that enhance the quality of printing.

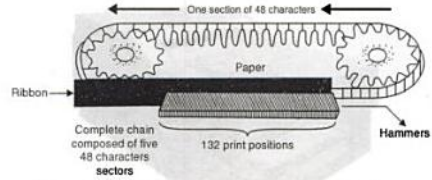


Figure 7.6. The printing mechanism of a chain/band printer.

An advantage of the chain printer that chain can be easily changed, so different character set can be used.

Non-impact Printer

These printers use thermal, electrostatic, or optical technology for printing. They do not make noise while printing and can operate at high speed.

For example, laser and ink-jet printers.

Ink-Jet Printer

Ink-jet printers are non-impact printers. This is a printer, which prints the characters in more quality form as compared to dot matrix printer. They print characters by spraying small drops of ink onto paper. Ink-jet printer produce high quality output because the characters are formed by dozens of tiny dots. Modern ink-jet printers can come close to producing near laser quality output. Ink-jet and other non-impact printers cannot produce multiple copies of a document in a single printing. Modern ink-jet printer can come close to producing near laser quality output. Colored ink-jet printers are also available.

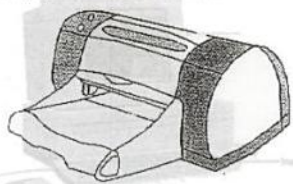


Figure 7.7. Ink-jet printer.

Laser Printer

Laser printer are non-impact printer and also known as quality printer. In these printers contain a drum with photosensitive surface. The laser beams or some other light sources are used to produce the image on a photosensitive drum. The image on the drum can be transferred onto the paper. The paper is passed through a heat chamber and that fuses the toner particles so the characters or images are formed permanently on the paper. After this process drum is discharged and cleaned. This process is done for one page after this the drum is ready for printing the next page. Colored laser printers are also available in the market, which can print millions of colors.

Figure 7.8. Laser printer.

Monitors

Monitor is very commonly used as an output device. A monitor is usually associated with keyboard and together they form a Video Display Terminal (VDT). A VDT is now very commonly used as an input/output device. Input to the computer is given through keyboard and output is displayed on monitor. The two basic types of monitors are:

- Cathode Ray Tube (CRT)
- Flat-panel

The CRT monitors look much like a television and are used with non-portable computer systems. On the other hand the flat-panel monitors are thinner and lighter and are commonly used with portable computer systems like notebook computers.

Figure 7.9. Monitor with keyboard.

Plotter

A plotter is a specialized output device designed to produce high quality graphics in a variety of colors. Plotters are used to create hardcopy items such as maps,architectural drawings and three dimensional illustrations which are usually too large for regular printers.

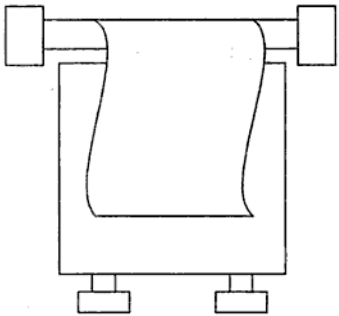


Figure 7.10. Plotter.

There are three kind of plotters:

(1) Pen plotters

(2) Electrostatic plotters

(3) Large format plotters

1. **Pen Plotters**

   It uses one or more colored pens to draw on paper or transparencies, it produces continuous lines, not patterns of paper or transparencies, it produces continuous lines not patterns of dots.

2. **Electrostatic Plotters**

   In an Electrostatic Plotters, paper lines partially flat on a table like surface, and toner is used in a photocopier-like manner.

3. **Large Format Plotters**

   In this type of plotters operate somewhat like an ink jet printer but on a much larger scale. This type of plotter is often used by graphics artists.

□□□

8 STORAGE DEVICES

INTRODUCTION

As we already know that in computer system, we have two types of memory primary and secondary storage. Every storage unit has following properties:

- ✔ **Access time:** This is the time that storage unit takes to locate and retrieve data. A fast access time is preferred.
- ✔ **Storage capacity:** It is the amount of data that can be stored in the storage unit. A large capacity is desired.
- ✔ **Cost per bit of storage:** Goal is to minimize this cost.

Primary Memory

A primary or internal memory is made up of cells. The cells are small storage area in primary memory. Cells have fixed length means they can store only fixed number of bits called word length of that particular primary storage. Each cell has a unique address. The size of memory is generally measured in some power of 2.

Each element of memory is directly accessible and can be examined and modified without affecting other cells and hence primary memory also called Random Access Memory (RAM). Another Part of main memory is Read Only Memory (ROM).

RAM

In this kind of Memory it is possible to read and write data on any memory location, hence it is also known as R/W Memory. It is volatile memory means if the power is switched off, the information stored in this memory are erased. In this memory access to memory location can be direct or random and takes equal time to access any location.

RAM is the main working memory: CPU instructions can only be executed from RAM; software has to be loaded into RAM before the instructions can be executed, and data has to be loaded into RAM before it can be manipulated by those instructions. It is fastest memory and expensive.

DRAM

It is a type of RAM that only holds its data if it's continuously accessed by special logic called a refresh circuit. Many time this circuitry reads and then re-writes the contents of each memory cell, whether the memory cell is being used at that time by the computer or not.

ROM

Read Only Memory (ROM) is the other type of internal memory. In this memory we can perform only read operation, write operation is not permitted. When the power supply is switched off, the information stored inside a ROM is not lost as it is in the case of a RAM chip. In this memory information is stored at the time of manufacturing. Once the information is stored it cannot be changed. Such memories are also known as field stores, permanent stores or dead stores. ROM is used to store the basic set of instructions, called the basic input output system (BIOS).

Features

- ✔ Non-destructive read out
- ✔ Long data life
- ✔ Non-volatile

ROMs can be further classified as:

- PROM
- EPROM
- EEPROM

Programmable Read Only Memory (PROM)

PROM can be programmed using special equipment known as PROM- programmer. However once the chip has been programmed, the recorded information cannot be changed, *i.e.*, the PROM becomes a ROM and it is only possible to read the stored information. It is useful for companies, which makes their own ROMS for their software. For small quantities it is cheaper than ROM.

Erasable Programmable Read Only Memory (EPROM)

EPROM is a ROM that can be erased and reprogrammed using a special PROM- programmer facility. In EPROM chip the Information can be erased by exposing it for some time to ultraviolet light. It is much useful than a regular PROM, but it does require the erasing light.

EPROM's are mainly used by R & D personnel (experimenters) because they frequently change the micro programs to test the efficiency of the computer system with new programs.

Electrically Erasable Programmable Read Only Memory (EEPROM)

In this memory information stored in EPROM chip are erased by electrical signal.

Regardless of the type of ROM chip used, they all serve to increase the efficiency of a CPU by controlling the performance of a few specialized tasks.

Memory Type	Write Time	Order of Read Time	Number of Cycles allowed
ROM	Once	Nano seconds	One
PROM	Hours	Nano seconds	One
EPROM	Minutes	Nano seconds	Hundreds
EEROM	Milliseconds	Nano seconds	Thousands

Disadvantage of Primary Memory

- ✔ The primary memory is volatile in nature.
- ✔ The cost per unit storage is very high for the primary memory.
- ✔ The design of computer generally limits the maximum possible size of primary memory.

Cache Memory

A CPU spends lots of its program execution time in either fetching instructions and data or storing results in the primary memory. Hence the speed with which a CPU can execute a program depends on the speed of the memory. The problem of speed mismatch between CPU and primary memory can be overcome by having a high-speed memory, known as cache memory. It is present as an interface between the CPU and the primary memory. It is an optional memory and may not be present in all computers. It is not directly accessible to users but CPU can access it direct.

It stores the segments of programs that are currently being executed and or temporary store very active data and instructions during processing. By making available data and program data rapid rate, the performance of CPU can be increased. It is very expensive memory as compared to other memories.

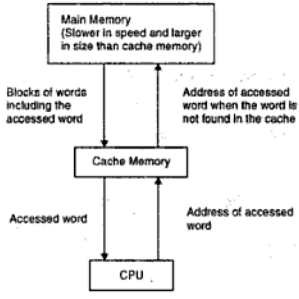


Figure 8.1. Operation of cache memory.

Secondary Memory

Secondary memory is also known as permanent memory used to store large volume of data.

It is characterized by low cost per bit stored, but it in generally has operating speed slower than that of the primary storage. This memory also referred as backup storage. Data are stored in the same way as in main memory.

Basically there are two methods of access provided by these devices. Hence secondary storage devices are classified into two types that are:

- Sequential (Serial) access devices,
- Random (Direct) access devices.

In sequential access device data can be accessed by sequencing through locations or we can say that data is retrieved in the same sequence in which it is stored. Sequential access is useful in application where each data needs to be accessed in turn *i.e.*, preparation of monthly pay slips. Some sequential access storage device are punched paper tape, magnetic tape etc.

In several applications immediate access to data is required. For example in an airline reservation system, immediate access to data may be required to find out the number of seats currently available on a particular flight. In such cases, random access devices will be useful. That devices provide immediate access to data. Random access devices available to suit the requirement for different levels of cost and performance. For example hard disks, floppy disks and CD-ROM etc.

Magnetic Tape

Magnetic tape is one of the most popular sequentially access storage mediums for large data. Magnetic tape was the first magnetic mass storage devices. The tape is plastic ribbon usually ½ inch or ¼ inch wide and 50 to 2400 feet long. Tape is coated on one side with an iron-oxide material, which can be magnetized and it is mounted on a reel for easy handling. It is similar to the tape used on a tape recorder expect that it is of higher quality and more durable.

Magnetic tapes are reusable. Old data is automatically erased when new data are recorded in the same area.

The tape is divided into vertical columns called frames and horizontal rows called channels or tracks. Information is recorded on the tape in the form of tiny invisible magnetized and non-magnetized spots (representing 1's and 0's) on the iron-oxide side of the tape. A character is recorded per frame using one of the computer codes.

Older tapes had 7 tracks and they used the 6- bit BCD code format for data recording. As in BCD code, the letter A is represented on this tape by the code 110001. The first six tracks are used for recording the 6 bits of the BCD code. The seventh track is used for recording the parity bit.

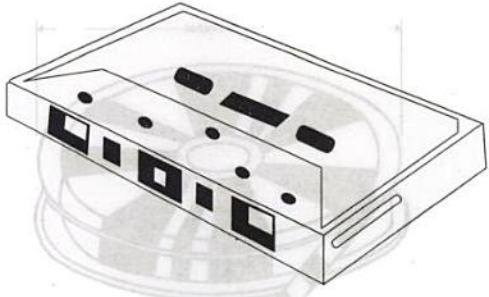


Figure 8.2. Magnetic tape cassette.

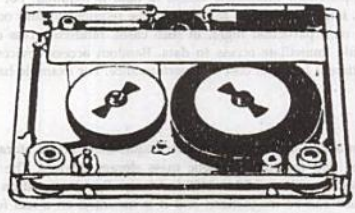


Figure 8.3. Magnetic tape cartridge

Most of the modern magnetic tapes have 9 tracks and they use the 8-bit EBCDIC code format for data recording.

Magnetic tape drive is a machine that can either read data from a tape into the CPU. Or it can write the information being produced by the computer onto a tape. To read or write information or to a tape, a tape drive is used. The tape on a reel moves through a tape drive in much the same way that a film moves through a movie projector. The tape drive allows two reels to be mounted on it. One acts as supply reel and other as the take-up reel. During processing, the tape moves from a supply reel to a take-up reel via two vacuum channels and through a read /write head assembly. The read/write head assembly is a single unit having one read/write head for each tape track. They either read information, or write information on the tape.

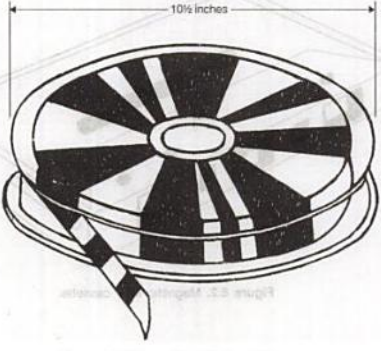


Figure 8.4. Magnetic tape in reel form.

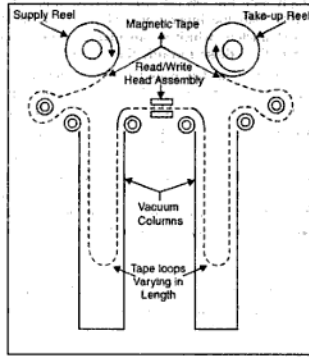


Figure 8.5. A magnetic tape drive.

The two vacuum channels are designed to take up slack tape, acting as buffers to prevent the tapes from snapping or stretching when starting from a stationary position or slowing down from full speed. Several methods are used to prevent tape damage from sudden bursts of speed and different tape drives may use different mechanisms for this purpose.

Advantages of magnetic tape:

- ✔ Unlimited storage
- ✔ High data density
- ✔ Low cost
- ✔ Rapid transfer rate.

Limitations of magnetic tape:

- ✔ No direct access
- ✔ Indirect interpretation
- ✔ Environment problems.

Magnetic Disk

An alternative to tape storage is magnetic disk storage. The two primary types of disks in use are floppy disks and hard disks. The general characteristics and operations of both types are the same. A disk is basically a plotter that has been coated with a magnetic material, or, reflective material, in the case of optical disks. Like tape drives, disk drives are equipped with read/write heads. As the disk spins inside the drive, the read/write head passes over the surface of the disk. A magnetic disk is a thin, circular metal plate/plotter coated on both sides with a magnetic material, it is very similar in appearance to a LP gramophone record.

Information is recorded on the tracks of a disk surface in the form of invisible tiny magnetic spots. The presence of a magnetized spot represents a 1 bit and its absence represents a 0 bit.

A standard binary code, usually 8-bit EBCDIC, is used for recording data. In some systems, the outer tracks contain more bits than the inner track. However in most systems, each track contains the same number of characters, which means that the outer tracks of the disk are less densely packed with characters than those towards the centers.

The more disk surfaces a particular disk pack has, the greater will be its storage capacity. But the storage capacity of the storage capacity of disk systems also depends on the tracks per inch of surface and the bits per inch of track. Although the diameter of a standard sized disk

Access of disk = Seek time + Latency time.

Access Time

Disk access time is the interval between the time a computer makes a request for transfer of data from a disk system to the primary storage and the time this operation is completed. To access information stored on a disk the disk address of the desired data item must be specified. Information is always written from the beginning of a sector and can be read only from the track beginning.

Seek Time

In a disk system, the time required for a read/write head to move to the track where the data to be read or written is stored. Maximum seek time equals the time taken for the head to move from the outermost track to the innermost track.

Latency Time

In case of disk storage, the relational delay time for the desired data (sector) to come under the read /write head positioned over that track. Maximum latency time equals the time taken by disk to rotate once.

Storage capacity of a disk system = Number of recording surfaces * Number of tracks per surface * Number of sectors per track * Number of bytes per sector

Floppy Disk

The floppy disk is primary data exchange medium for PCs and the other most popular backup system. They were introduced by IBM in 1972 and are now being produced in various sizes by many manufacturers. Generally floppy disks are of 3 types:

1. 8 inches
2. 5.25 inches
3. 3.5 inches.

The floppy disk is a plastic circular disk coated with magnetic material. The floppy disk is enclosed with in a square plastic or card board jacket, often referred to as a cartridge which gives protection to the disk surface from heat, moisture and dust particles. It has a special liner that provides a wiping action to remove dust particles that are harmful for the disk surface. The jacket also has a small cut called write-protect-notch. When that notch is closed by a small piece of plastics or by a small plastic built in knob, the disk will not allow any thing to be written in the disk, but read operation is permitted. Both the jacket as well as the disk has a large hole at the center. The spindle of the floppy drive fits into this hole. The area around the hole is most prone to wear and damage, therefore most floppy disks are equipped with protective hub rings.

The data stored on the floppy is organized in tracks and sectors. The recording surface of the floppy gets divided into a number of concentric circles called tracks. Each track is divided

into a fixed number of sectors that is same in each track. Each sector has a fixed storage capacity. The storage capacity of every sector across all the tracks is the same.

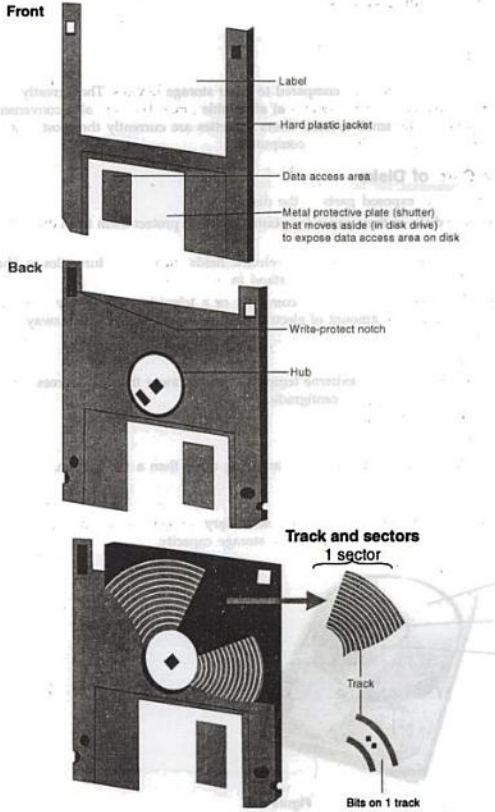


Figure 8.6. Floppy device.

As numbers of floppy disks are available and one can be differentiated from another by means of following factors:

- ✔ Number of recording surfaces
- ✔ Storage capacity
- ✔ Density
- ✔ Size

Floppy disks are very cheap as compared to other storage devices. They greatly enhance the on line storage capacity of small systems at affordable price. They are also convenient off-line storage medium for the small system. Users diskettes are currently the most popular, in expensive storage medium used in micro computers.

How to Take Care of Disks

1. Do not touch the exposed parts of the disk.
2. Always place disks in their paper (or plastic) jacket to protect them from dust, dirt, and smoke.
3. Do not place disks near magnetic and electric fields such as the turnstiles in libraries, stores, or doors that open when you stand in front of them.
4. Do not store disks on the top of a computer or a television set, or any machine that requires a considerable amount of electricity to operate. Keep diskettes away from the telephone.
5. Never bend floppy disks.
6. Do not expose diskettes to extreme temperatures; not more than 62 degrees centigrade and not less than 10 degrees centigrade.
7. Do not use paper clips on disks.
8. Store diskettes in a safe place.
9. Avoid writing on a 3.5" diskette with anything other than a felt-tip pen.

Hard Disk

Hard Disks are currently the most popular secondary storage media, which provide direct and high-speed access to data and offer a large storage capacity.

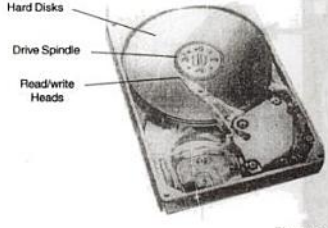


Figure 8.7.

Floppy disks use flexible plastic, but hard disk use metal. Hard disk is thin but rigid metal plotters covered with a substance that allows data to be held in the form of magnetized spots. Hard disks are tightly sealed within an enclosed hard disk unit of prevent any foreign matter from getting inside. Data may be recorded on both sides of the disk plotters. Hard disks are quit sensitive devices.

CD-ROM

CD-ROM is most popular storage device for users of PC of all kind .CD ROM use optical technology for storing and retrieving information. A CD_ROM is a small lightweight disc made of transparent plastic. The disk is covered with a shiny (silvered) reflecting surface. It can be used to store large amount of information. To record the information very tiny holes or pots are burnt into the surface of the disk using laser beams. The presence or absence of a pit indicates a bit value. To read the bits a low intensity laser beam is directed on the disc.

□□□

9 OPERATING SYSTEM

INTRODUCTION

A computer is a complex system. However, it is required that a person with minimal technical skills should be able to use a computer. This is made possible by the operating systems of the computer.

An operating system is a system software, which is set of specialized programs that are used to control the resources of a computer system. Various resources are memory, processor, file system and I/O devices. It is an essential component of the computer system. It is the program running at all times on the computer. Operating systems and computer architecture have a great deal of influence on each other. To facilitate the use of the hardware, operating systems were developed. An operating system is also known as master control program, resource manager and monitor. Operating system is an interface between a user and a computer. Operating systems should be friendly enough that a user can easily interact with the system.

Goals of Operating System

- ✔ Make computer system convenient to use.
- ✔ Managing computer system resources in an efficient manner.

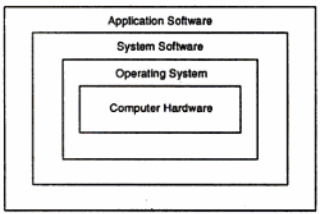


Figure 9.1.

Functions of Operating System

Processor Management

A process is a program in execution. During execution a process needs certain resources such as CPU time, memory space, files and Input/Output devices. (At a particular instance of time a computer system normally consists of a collection of process). The process management module of an operating system takes care for- the creation and deletion of process scheduling of various system resources to the different processes scheduling of various system different process requesting them and providing mechanism for various for synchronization and communication among process. Processor management, that is assignment of processors to different tasks being performed by the computer system.

Device Management

A computer system consists of several Input/Output devices such as terminal, printer, disk and tape. The device management module of an operating system tasks care of controlling all the computer's Input/Output devices. It also provides an interface between the devices and the rest of the system that is simple and easy to use.

Memory Management

To process the job, required data and job must be loaded into the memory. Function of an operating system is to allocate memory in such a way that it improves the CPU utilization and system efficiency.

Command Interpretation

As a user communicates with computer system using specified set of commands. Any command given by user first interpreted by the operating system then appropriate action is taken.

Security

It provides provision for security so that system is safe from unauthorized access *i.e.*, using passwords.

Communication

It provides efficient and easy mean of communication between user and system.

Performance Evaluation

The efficiency of an operating system and the overall performance is depend on the following factors:

1. Throughput

It is measured as total work done by system per unit time. Throughput depends not only on the efficiency of the system but also depends on the job to be processed.

2. Turnaround Time

It is measurement of time elapsed between job submission and job completion.

3. Response Time

It is measurement of time elapsed between job submission and first job response produced.

Types of Operating System

(1) Batch processing

(2) Single user
(3) Multi-user
(4) Multi-tasking
(5) Multi-programming
(6) Multi-processing
(7) Time sharing
(8) Real time.

Batch Processing Operating System

Batch processing system is also known as off line or sequential or stacked processing system. It is one of the oldest operating system. In this operating system jobs with similar need batched together by operator and run as a group on a computer system. Users cannot directly interact with the system, just prepared job and submit it to operator. When output produced it is submitted to appropriate user. The major job of batch operating system is to transfer control automatically from one job to another. It is appropriate for executing large jobs that need little interaction.

Advantages

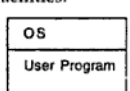
- ✔ User /Customer need to wait in queue
- ✔ Huge Amount of a data can be processed efficiently.
- ✔ It reduces idle time of a computer because operator intervention is not required in automatic job-to-job transition.

Disadvantages

- ✔ It reduces timeless in some cases.
- ✔ Priority scheduling is difficult to achieve.
- ✔ Turnaround time increases in some cases.

Single-user Operating System

These operating systems allow only one user to operate at a time *i.e.*, DOS. DOS is the most popular single user operating system. These operating systems are mainly used on personal computers. In DOS we cannot run another program at same time. For this we will have to close the first program, only then we would be able to work on another program or software. Windows 95,Windows 98 are single user operating system but they support multi-tasking and multi-processing facilities.



Multi-user Operating System

It allows simultaneous access to a computer system through two or more terminals .A dedicated transaction processing system such as railway reservation system that supports hundreds of terminal under control of a single program is an example of multi-user operating system. Multi-processor operation without multi-user can be found in operating system of some advanced personnel computers and in real system. *e.g.*, UNIX, XENIX. It is useful in

a field where simultaneous access to computer system is required *i.e.*, railway reservation system etc.

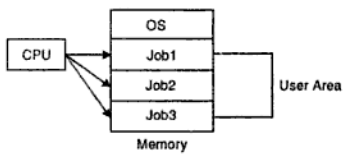
Multi-tasking Operating System

In multi-tasking, two or more program's can be executed by one user concurrently on the same computer with one central processor. You may write a report on your computer with one program while another program plays a music CD.

To do this work the operating system directs the processor to spend a predetermined amount of time executing the instruction for each program one at a time. Thus a small part of the first program is processed and then the processor moves to the remaining programs one at a time, processing small part of each program, this cycle is repeated until the processing is complete. These operating system support simultaneous processing of several tasks *i.e.*, Windows 95, Windows 98, Windows 2000 etc.

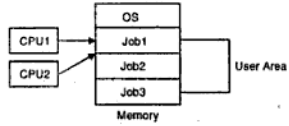
Multi-programming Operating System

In batch operating system job batches are stored in memory for processing. The operating system picks one job and executes it. Eventually jobs have to wait for some Input/Output operation. That CPU sits idle for that time hence system performance degrades. Multiprogramming increase CPU utilization by organizing jobs such that the CPU always has one job to execute. It is possible by allowing switching between jobs. In multiprogramming operating system more than one job resides in main memory and hence the memory is utilized. The CPU picks one job and starts executing it, when the job requires to perform Input/Output operation, the CPU does not sit idle and picks the next job and start executing it's instruction, in this way a single CPU is in demand all the time and hence it is fully utilized.



Multi-processing Operating System

Multi-processing is a processing done by two or more computers or processors linked together to perform work simultaneously and precisely at the same time. This can entail processing instruction from different programs or different instructions within the same program at once. Two possible approaches to multiprocessing are co-processing and parallel processing. In co-processing the controlling CPU works together with a specialized microprocessor called coprocessor each of which handles a particular task such as creating display–screen graphics or performing high-speed mathematical calculations. Many microcomputer systems have co-processing capabilities .In Parallel processing several full-fledged processors work together on the same tasks, sharing memory. Parallel processing is often used in large computer systems designed to keep running if one of the CPU fails.



Time Sharing Operating System

In time sharing operating system the many users can operate computer system simultaneously. The CPU executes multiple jobs and switches between them so frequently that the user may interact with each job while it is running. CPU time is divided between users that use it one-by-one. It provides an interactive online communication between user and the system. Time sharing systems are expensive and more complex than multi-programming operating systems.

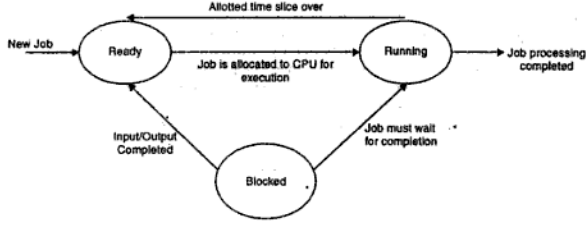


Figure 9.2. The process state diagram for time sharing system.

Real Time Operating System

The real time operating system has well defined, fixed time constraints. Processing must be done within fixed time constraint. This operating system provides quick response time. A real time operating system is considered to function correctly only if it returns the correct result within any time constraint.

Real time operating systems are used when there are rigid time requirement on the operation of a processor or the flow of data, and thus is often used as control device in dedicated application. Systems that control scientific experiments, medical imaging systems, industrial control systems, and some display systems are real-time systems.

There are two types of real time operating systems:

(1) Hard real time operating system

(2) Soft real time operating system

Hard real time operating system assured that critical tasks completes on time. This goal requires that all delays in the system be bounded, from the retrieval of stored data to time that it takes the operating system to finish any request made to it. On the other hand, real time

operating system is a less-restrictive real time operating system, where a critical real-time task gets priority over other tasks, and retains that priority until it completes. Soft real time operating systems have more limited utility than hard real operating systems.

DOS

DOS or Disk Operating system is system software that provides an interface between the computer hardware and the computer user. DOS makes the computer alive. Without the DOS the computer is a dead machine, it cannot be used for any purpose. The most commonly used DOS with the PC is the MS-DOS, developed by microsoft corporation. It is operating system prepared for carrying single user and provide character user interface.

Functions of DOS

DOS acts an interpreter or we can say it acts as an interface between the hardware and the software.The major functions of DOS are summarized below.

1. It creates back-up file.
2. It also controls the operation of hardware like CPU and memory.
3. It allocates memory to programs.
4. The system unit contains the brain of PC–the CPU, hard disk and so on. All of these are derived by DOS which in turn is directed by the commands we type in at the keyboard.
5. It creates new files, deletes old files and can also rename them.
6. It formats the new floppy disks so that they can be made usable.

Wild Cards

A wild card is a character that can represent one or more character in a file name. The DOS has recognized two wildcards

(*i*) The asterisk (*) can represent one or more characters that a group of files have in common.

(*ii*) The question mark (?) represents s a single character that a group of files have a common name.

Example :

```
A*.*
        /*means all file whose name starts with letter A.
RCE*
        /* all file and directory whose names start with R.
?
        /* file and directory whose names contain only one letter.
```

Two types of commands are provided in DOS, internal and external.

Internal vs. External Command

(*i*) Internal command are always there in the primary memory whereas the external command needs to be loaded from the secondary memory into the primary memory each time they are executed.

(*ii*) Internal command are faster in execution as compared to the external commands.

(*iii*) We cannot add internal command from our side but we can add external command from our side.

File naming rules are:

1. A file name can have two parts, a main file name and a name extension.
2. The name can be minimum 1 char long and maximum 8 char long.
3. The name extension can be minimum 0 char long *i.e.*, you can have file name without a name extension and the max. Length of the name extension can be 3 char.
4. A file name and the name extension can contain alphabet as well as numbers and they can start with alphabet or numbers.
5. The file name and the name extension are separated by a period (.) symbol. A file name cannot contain more than one period symbol.
6. Other than alphabet and the numbers a file name and extension can also contain some other character available on the keyboard but many of these characters are not allowed in a file name so it is best to avoid than when giving a file name. Some of special characters not allowed in a file name.

   ^ [ ] : ; " < > + – == *
7. A file name or the name extension cannot contain the space.
8. A file name cannot be the following reserved words. These words have special use for the DOS.

   Com, Con, Lpt, Prn.

Some Basic Command in DOS

1. VER

The ver command display the current DOS Version.

C:\ver and press Enter

MS Dos Version 6.22

2. TIME

The time command can be used to display and change the system time.When the time command is executed, it displays the system time . It also prompts the user for the new time, if you want to change the system time you can type in the new time.

Time and press Enter

Current time is 4 : 32 : 45.56p

Enter new time:

3. DATE

The date command can be used to view and change the system date. When the date command is executed, it displays the system date.

Date and press Enter

Current date is

Enter the date (mm-dd-yy)

4. CLS

This command is executed and clears the screen and places the prompt and the top left corner of the screen.

CLS and press Enter

5. DIR

The DIR command can be used to list selected entries present in a specified directory (To get the list of all the files created in the current directory we can DIR command).

DIR and press Enter

It displays all the directory and files of the current drive.

/P If list of files is long enough not to fit on the screen, then that could be seen page wise. Page wise display one screen on the listing at a time .To see the next screen, press any key.

/W will give a horizontal display of files have five files in a time.

/A display files with specified attributes.

D = Directories.

H = Hidden files.

S = System files.

R = Read only files.

/O : It displays the files in sorted order.

N = In ascending alphabetic order by name.

E = In ascending alphabetic order by extension .

D = Sorted by data and time with earliest first.

S = In Increasing order by size ,smallest first.

L = Display unsorted directory names and filenames in lowercase.

6. MD or MKDIR

This command creates a new directories. We can create a new directory or sub directory in the main directory by using this command.

MD < dir name >

7. CD

This command is used to change the directory or to display the name of the current directory.

CD < dir name> and press Enter

Change to a subdirectory which must be there in the current directory.

CD\ < dir name>

Change the directory to a main directory which must be there in the root directory.

8. RD or RMDIR

Delete a directory. The directory must be empty except for the "." And "..." symbols.

RD < Directory name >

9. REN or RENAME

Changes the name of the file or the directory.

REN < Old file name > < New file name>

10. DEL

This command is used to delete the file.

Del < File name>

11. TREE

This command is used to display the directory structure nested in a specified directory. The directory structure is displayed in the form of tree.

Tree < Directory name >

12. COPY CON

This command is used to make a file. (This is a actually a copy command which is going to get data from the CON (Console) and will store data in the file specified at the end of the command.

COPY CON < File name >

Here Con is a reserved word which is used to console. There is a drawback of this command that we cannot modify the contents of previous line. That means once after moving to a new line we cannot go the previous line.

To save the file by **Ctrl Z or F6.**

13. EDIT

Display the full screen editor while not as fully featured as a word processor ,it is handy for editing files. If we invoke EDIT without parameter the editor appears without a default open file or we can specify file name.

EDIT drive :\path\ file

14. TYPE

This command is used to display the content of a text file.

Type < File name>

15. PROMPT

This command is used to change prompt.

PROMPT < Text>

To go back up to the default PROMPT use PG

16. DOSKEY

This command is used to load the Doskey program into memory. The doskey program recalls MS-DOS Command.

17. VOL

VOL displays the volume label of the specified drive.

VOL :

18. ATTRIB

Displays or changes file attributes.

ATTRIB [+R | –R] [+A | –A] [+S | –S] [+H | –H] [[drive:][path]filename] [/S]

\+ Sets an attribute.

– Clears an attribute.

R Read-only file attribute.

A Archive file attribute.

S System file attribute.

H Hidden file attribute.

/S Processes files in all directories in the specified path.

MS DOS has a certain attribute with a file such as read only , archive and hidden only. To add an attribute for a file we have to use `+' sign with the desired attribute and to remove an attribute we have to `-' sign.

+ R Read only attribute.

– R Remove the read only attribute.

+ H Hidden the file.

– H Remove the hidden attribute.

+ A Add archive attribute.

– A Remove archive attribute.

+ S Add the system.

19. COPY

This command is used to copy one or more files either on the same disk or from one disk to another.

COPY [/A | /B] source [/A | /B] [+ source [/A | /B] [+ ...]] [destination [/A | /B]] [/V] [/Y | /-Y]

source Specifies the file or files to be copied.

/A Indicates an ASCII text file.

/B Indicates a binary file.

destination Specifies the directory and/or filename for the new file(s).

/V Verifies that new files are written correctly.

/Y Suppresses prompting to confirm you want to overwrite an existing destination file.

/–Y Causes prompting to confirm you want to overwrite an existing destination file.

The switch /Y may be preset in the COPYCMD environment variable.

This may be overridden with /–Y on the command line

To append files, specify a single file for destination, but multiple files for source (using wildcards or file1+file2+file3 format).

20. XCOPY

This command is very useful in a situation where it desired to copy files from sub-directories of the current directory to the disk.XCOPY command automatically created the corresponding sub directories on the disk or drive if they do not already exist.

/S Copies directories and subdirectories except empty ones.

/E Copies directories and subdirectories, including empty ones. Same as /S /E. May be used to modify /T.

/W Prompts you to press a key before copying.

/C Continues copying even if errors occur.

/I If destination does not exist and copying more than one file, assumes that destination must be a directory.

/Q Does not display file names while copying.

/F Displays full source and destination file names while copying.

/L Displays files that would be copied.

/H Copies hidden and system files also.

/R Overwrites read-only files.

/T Creates directory structure, but does not copy files. Does not include empty directories or subdirectories. /T /E includes empty directories and subdirectories.

/U Updates the files that already exist in destination.

/Y Overwrites existing files without prompting.

/-Y Prompts you before overwriting existing files.

/N Copy using the generated short names.

21. FORMAT

Formats a disk for use with MS DOS. It a new root directory and file allocation table for the disk . Every new disk must firs be formatted using this command so that MS DOS may be use that.

FORMAT drive: [/V[:label]] [/Q] [/F:size] [/B | /S] [/C]

FORMAT drive: [/V[:label]] [/Q] [/T:tracks /N:sectors] [/B | /S] [/C]

FORMAT drive: [/V[:label]] [/Q] [/1] [/4] [/B | /S] [/C]

FORMAT drive: [/Q] [/1] [/4] [/8] [/B | /S] [/C]

/V is used to tell FORMAT commands that we want a volume\name to be disk label.

/Q Performs a quick format.

/B Allocates space on the formatted disk for system files.

/S Copies system files to the formatted disk.

/1 Formats a single side of a floppy disk.

/4 Formats a 5.25-inch 360 K floppy disk in a high-density drive.

/8 Formats eight sectors per track.

/C Tests clusters that are currently marked "bad."

22. DISKCOPY

This command is used to make a duplicate copy of a disk . It copies the contents of diskette in the source drive to the diskette in the target drive .The target diskette is formatted if necessary, during the copy.

DISKCOPY [drive1: [drive2:]] [/1] [/V] [/M]

/1 Copies only the first side of the disk.

/V Verifies that the information is copied correctly.

/M Force multi-pass copy using memory only.

The two floppy disks must be the same type.

You may specify the same drive for drive1 and drive2.

23. CHKDSK

Checks a disk and displays a status report.

CHKDSK [drive:] [path]filename] [/F] [/V]

[drive:][path]	Specifies the drive and directory to check.
filename	Specifies the file(s) to check for fragmentation.
/F	Fixes errors on the disk.
/V	Displays the full path and name of every file on the disk.

Type CHKDSK without parameters to check the current disk. Instead of using CHKDSK, try using SCANDISK. SCANDISK can reliably detect and fix a much wider range of disk problems.

24. LABEL

Creates, changes, or deletes the volume label of a disk.

LABEL [drive:][label]

25. DISK COMP

This command is used to compare two files to check if they are identical .

DISK COMP [DRIVE 1] [DRIVE 2]

Where,

[DRIVE 1] = is the first drive

[DRIVE 2] = is the second drive to be compared.

26. MORE

Displays output one screen at a time.

MORE [drive:][path]filename

MORE < [drive:][path]filename

command-name | MORE [drive:][path][filename]

[drive:][path]filename	Specifies file(s) to display one screen at a time
command-name	Specifies a command whose output will be displayed.

□□□

10 VIRUSES

INTRODUCTION

A virus is a deviant program stored on a computer hard drive that can cause unexpected and often undesirable effects, such as destroying or corrupting data.

Types of Viruses

1. **Boot Sector Virus**

   The boot sector is that part of the system containing most of the instruction for booting or powering up, the system. The boot sector virus replaces these boot instructions with some of its own. Once the system is turned on the virus is loaded into the main memory before the operating system. Any diskette is moved to another computer, the contagion continues.

   Example of boot sector viruses AntCMOs, AntiExe, NYB(New York Boot) Ripper.

2. **File Viruses**

   File viruses attach themselves to executable files—those that actually begin a program. When a program is run, the virus starts working trying to get into main memory and infecting other files.

3. **Multipartite Viruses**

   A hybrid of the file and boot sector types the Multipartite Viruses infects both file and boot sector, which make it better at spreading and more difficult to detect *e.g.*, Junkie, and Parity Boot.

4. **Macro Virus**

   Macro viruses take advantage of a procedure in which miniature programs, known as macros are embedded inside common data files, such as those created by e-mail or spreadsheets, which are sent over computer networks. Until recently, such documents have typically been ignored by ant virus software. Example of macro viruses are concept, which attaches to word document and e-mail attachments, and Laroux, which attaches to Excel spreadsheet files. Fortunately, the latest versions of word and Excel come with built-in macro virus protection.

5. **Logic Bomb**

   Logic bombs, or simply bombs, differ from other viruses in that they are set to go off at a certain date and time. A disgruntled programmer for a defense contractor created

a bomb in a programmer that was supposed to go off two months after he left. Designed inventory to erase an inventory tracking system the bomb was discovered only by chance.

Virus are passed in computer by two ways:

- **By Diskette:** The first way is via an infected diskette, perhaps obtained from a friend or a repair person.
- **By Network:** The second way is via a network, as form e-mail or an electronic bulletin board. This is why, when taking advantage of all the freebie games and other software available online, you should use virus-scanning software to check downloaded files.

Antivirus

Antivirus software scans a computer's hard disk, floppy disks and main memory to detect viruses and sometimes, to destroy them. Example of antivirus programs are Symantec,s, Nortan AntiVirus, Mcafee Virus Scan, Panda Antivirus Platinum, and Computer Associates 'E Trust EZ Antivirus for Windows and Virex for Macs.

TEST YOURSELF

1. What is a computer and describe its characteristics ?
2. What are capabilities of computer ?
3. What are digital, hybrid and analog computers ?
4. How computers can be classified according to their size ?
5. What is data ? What is the procedure of manipulating data in data processing ?
6. Describe different unit of computer and its function.
7. What is system software ?
8. What is application software ?
9. Draw a labelled diagram computer and its basics.
10. Classify computers physical quantity wise.
11. What is the full form of Bit, ASCII ?
12. What is RAM and how it is differentiate from ROM ?
13. What is operating system and define its various functions ?
14. Explain the three language translators.
15. Differentiate between an impact printer and a non-impact printer.
16. What is firmware ?
17. Of what advantage is CD-ROM ?
18. What are the three types of magnetic disk units ?
19. Explain how input devices are classified.
20. Distinguish between impact and non-impact printer.
21. Convert the following from one system to another:
    (*a*) $(124.64)_{10}$ = $(?)_2$
    (*b*) $(111.01011)_2$ = $(?)_{10}$
    (*c*) $(1100101)_2$ = $(?)_{10}$

(*d*) $(10110100001)_2$ = $(?)_{10}$
(*e*) $(3608)_{10}$ = $(?)_2$
(*f*) $(489032)_{10}$ = $(?)_2$
(*g*) $(59.99)_{10}$ = $(?)_2$
(*h*) $(100.011)_{10}$ = $(?)_2$
(*i*) $(645.456576)_{10}$ = $(?)_8$
(*j*) $(95678)_{10}$ = $(?)_8$
(*k*) $(101011.0101)_{10}$ = $(?)_8$
(*l*) $(2921.564)_{10}$ = $(?)_8$
(*m*) $(73452)_8$ = $(?)_{10}$
(*n*) $(720.101)_8$ = $(?)_{10}$
(*o*) $(11010.0101)_2$ = $(?)_8$
(*p*) $(65471)_8$ = $(?)_{10}$
(*q*) $(51302.A2)_{16}$ = $(?)_{16}$
(*r*) $(19668)_{10}$ = $(?)_{16}$
(*s*) $(689.562)_{10}$ = $(?)_{16}$
(*t*) $(439.678)_{10}$ = $(?)_{16}$
(*u*) $(3DA)_{16}$ = $(?)_{10}$
(*v*) $(1E8D.B54A)_{16}$ = $(?)_{10}$
(*w*) $(380.68)_{16}$ = $(?)_{10}$
(*x*) $(EEF2.FE8)_{16}$ = $(?)_{10}$

□□□

PART – B

1 AN INTRODUCTION TO C

C is a general-purpose high-level structured programming. Its instruction consists of terms that resemble algebraic expressions by certain English keywords such as if, else, do, while, for etc.

It has been closely associated with the UNIX system where it was developed, since both the UNIX system and most of the programs that run on it are written in C. C contains certain additional feature that allows it to be used at a lower level, thus bridging the gap between machine language and high-level languages. This flexibility allows C to be used for systems programming as well as for applications programming.

History of C

C was originally developed in 1970's by Dennis Ritchie at Bell Telephone Laboratories, inc (now AT & T Bell laboratories). C is an outgrowth of two earlier languages, called BCPL and B, which were also developed at Bell laboratories. Many of the important ideas of C stem from BCPL, developed by Martin Richard. The influence of BCPL on C proceeded indirectly through the language B, written by Ken Thompson in 1970 for the first UNIX system on DEC PDP-7.

Features of C

Availability

C compilers are available for almost all operating systems. ANSI has defined standard specifications of the C language, which is supported by almost all compilers available. In fact most operating systems of today are written in C.

The universal availability of C compilers makes C programs portable across different platforms with minimal modifications.

Efficiency

C is a very small language. Most of its operation can be directly expressed in low-level instructions that are present on most computers. C compilers take advantage of this feature to produce highly efficient code for programs.

Data Types

C provides a rich set of data types of varying sizes for different uses. The fundamental types are characters, integers and floating-point numbers of several sizes. In addition, there are derived data types created with pointers, arrays, structures and unions. Pointers provide for machine independent address arithmetic.

Functions

C supports modular programming. A C program can be viewed as a set of function. In fact almost all C programming efforts are directed towards the writing functions. Every C program has one function called main. Execution of a program starts from function main. C supports parameters passing mechanism (call by value and call by reference).

Bitwise Operators

C provides bit wise operators, for bit manipulation. These operators are typically used in systems programming. Bit wise operators may be applied only to integral operands.

Structure of a C Program

```
Preprocessor directives

variable declaration → Global Variables
function declaration → Global functions

main()
{
variable declaration → Local Variables
function declaration → Local Functions
statement-1
statement-2
......
......
statement-n
[return type]   function-name-1([arguments])
{

Function body

}
[return type]   function-name-2([arguments])
{

Function body

}
.
.
.
```

A C program consists of one or more function.

A function is a set of statements and variables. The statements specify the computing operations to be done and the variables hold the intermediate values of the computations.

In some programming languages a function is also called a subprograms, Subroutine or a procedure. Every C program must have a function called main. Execution of a C program begins at the beginning of a main.

Sample Program

Here is an elementary C program that reads in the radius of a circle ,calculates the area and then writes the calculated result.

Example 1. *Program to calculate the area of a circle.*

Solution:

```
#include <stdio.h>

main()
{
float radius,area;

printf (" Radius =?");
scanf ( "%f", &radius );
area = 3.14 * radius * radius
printf (" area = %f \n",area );
}
```

Include directives:

The line

```
# include<stdio.h>
```

causes the compiler to replace this line with the contents of the file "stdio.h". stdio.h is a header file . Header file contains the coding of the function which are used in the program (the function such as printf, scanf): The functions such as printf, scanf are called library function and they are declared in the header file and that is why it is necessary to include the header file so that the compiler can understand the function otherwise the compiler will report error.

* In some compiler the inclusion of the header file is default (Turbo C 2) .

The statements

main()

defines main as a function . The empty parentheses indicate that the main does not accept any arguments or parameters.

The set of statements that make up the body of a function are enclosed in braces {}.The body of a function is also known as a function definition. Functions cannot be nested in C.

* The main function is called **User Defined Function (UDF),** because the coding of the function depends on the programmer.

Statements

Every statement in a C program is terminated by a semicolon. C is a free format language. A single statement can span across multiple lines and multiple statement can be written in a single line.

C is case sensitive.

Variables and Data Types

The line float radius, area;

defines two variables radius and area as type float. This causes the compiler to reserve enough memory space to store, two floating point variables.

Input and Output

printf and scanf are functions defined in the standard library. These functions are declared in stdio.h printf is function that writes output to the standard output. Scanf is a function that reads input from the standard input.

Arithmetic Expressions

C provides a rich set of operators for performing arithmetic computations. The statement

```
Area =3.14 *radius*radius;
```

Multiplies the constant 3.14 with the sequence of radius and assigns the value to area. The = in the assignment operator in C. The assignment operators can be combined with operators. There is no specific assignment statement in C. The * is the multiplication operators.

Comments

A **comment** is a note to yourself (or others) that you put into your source code. The compiler ignores all comments. They exist solely for your benefit. Comments are used primarily to document the meaning and purpose of your source code, so that you can remember later how it functions and how to use it. You can also use a comment to temporarily remove a line of code.

Simply surround the line(s) with the comment symbols. In C, the start of a comment is signaled by the /* character pair. A comment is ended by */. For example, this is a syntactically correct C comment:

```
/* This is a comment. */
```

Comments can extend over several lines and can go anywhere except in the middle of any C keyword, function name or variable name. In C you can't have one comment within another comment. That is, comments may not be nested. Lets now look at our first program one last time but this time with comments:

```
main() /* main function heading */
{
  printf("\n Hello, World! \n"); /* Display message on */
}                                /* the screen */
```

This program is not large enough to warrant comment statements but the principle is still the same.

□□□

2 COMPONENTS OF C LANGUAGE

CHARACTER SET

Character set of a language is set of all the symbols used to write in that language. The characters can be used in the form word, numbers and expressions. The characters in C are grouped in the following categories:

1. Letters : A–Z or a–z
2. Digits : 0–9
3. Special symbols : !, @, #, %, ^, &, *, (), {}, [ ], " ', < >
4. White spaces : blank spaces, horizontal tab, carriage return new line, from feed.

C TOKEN

Keywords

Keywords are word, which have been assigned specific meaning in the contest of C language program. All the keyboards must be written in lowercase. The under score character is also permitted in identifiers. The standard keywords are as follows:

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	volstile
do	if	static	while

Constants

Constant is a fixed value that does not change during the execution of a program. C supports several type of constants.

1. Character
2. Integer
3. Real
4. String
5. Logical

1. Character Constant

A character constant consists of a single character single digit or a single special symbol enclosed within pair single inverted commas.

Examples: `8', `9', `Z', `:'.

2. Integer Constants

An Integer Constants refers to a sequence of digits. There are three type of integers, namely, decimal, octal and hexadecimal.

Decimal integers : 1, 56, 7657, – 87 etc.
Octal integer : 077, – 076, 05 etc. (preceded by zero)
Hexadecimal integer : 0×56, – 0×5D etc. (preceded by zero, 0 ×)
No commas or blanks spaces are allowed in integer constants.

Examples: Decimal constant : 370, 4354
Octal constant : 057, 04321
Hexadecimal constant : 0×8a, 0×9f

3. Real and Floating Point Constant

A number with a decimal point and an optional preceding sign represents a real constant.

Example: 23.3, – 564.33, .4, – 43.

4. String Constants

A string constant is a sequence of one or more characters enclosed within a pair of double quotes (`` ''). If a single character is enclosed within a pair of double quotes, it will also be interpreted as a string constant.

Example: ``Hello to the world of C ''
``Akash everywhere ''

5. Logical Constants

A logical constant can have either true value or false value. In C a non-zero value is treated as true while zero is treated as false.

Variables

A variable is an entity whose value can change during the program execution. Variables are declared in the beginning of the main function or in any other function. Each variable has a name and data type. All the variables must have their type indicated so that the compiler can record all the necessary information about them, generate the appropriate code during translation and allocating required space in memory.

Rules for Constructing Variable Name:

1. No commas or blank space are allowed in a variable name.
2. First character must be an alphabet or an under score (_).
3. Among the special symbols, only underscore can be used in a variable name.
4. No word, having a reserved meaning in C can be used for variable name.
5. The number of characters a variable name can have depends on the compiler in use. (But most compiler accept only first eight characters as variable name.)

Example of variable names:

Variable Name	Valid/Invalid	Remark
SUM_6	Valid	
while	Invalid	While is keyword
Price#	Invalid	# sign is illegal
Total Sum	Invalid	Blank space is not allowed
int_sum	Valid	Keyword may be part of name

Data Type In C

C supports different types of data. Storage representations of these data types are also different in memory. C supports four classes of data types:

(1) Primary data type

(2) User-defined

(3) Derived

(4) Empty data set

There are four fundamental data type in C, which are:

(*i*) **int**–integer: a whole number.

(*ii*) **float**–floating point value: *i.e.*, a number with a fractional part.

(*iii*) **double**–a double-precision floating point value.

(*iv*) **char**–a single character.

There are also several variants on these types.

Short	An integer, possibly of reduced range.
Long	An integer, possibly of increased range.
Unsigned	An integer, with no negative range, the spare capacity is used to increase to positive range.
Unsigned long	Like unsigned, possibly of increased range.
Double	A double precision floating point number.

Different possible data type and byte allocated to them in memory and their range specified in table given below:

Data Type	Size (Bytes)	Range
Char or signed char	1	– 128 to 128
Unsigned char	1	0 to 255
Int or signed int	2	– 32768 to 32767
Unsigned int	2	0 to 65535
Short int or signed short int	1	– 128 to 127
Unsigned short int	1	0 to 255
Long int or signed long int	4	– 2147483648 to 2147483637
Usigned long int	4	0 to 4294967295
Float	4	3.4 E – 38 to 3.4E + 38
Double	8	1.7 E – 308 to 1.7 E + 308
Long double	10	3.4E _4932 to 1.1 E + 4932

Variable Declaration

In C, a variable must be declared before it can be used. Variables can be declared at the start of any block of code, but most are found at the start of each function. A declaration begins with the data type, followed by the name of one or more variables and ending with the semicolon. The general syntax for variable declaration is as follows:

Data type [list of variables separated by comma];

For example,

```
int total, extra, runs[20];
float average;
int a[10];
char name[10];
unsigned int num;
long  r,s,t;
```

Declaration does two things:

1. It tells the compiler what the variable name is.
2. It specifies what type of data the variable will hold.

Initializing Variables

Variables can also be initialized when they are declared, this is done by adding an equals sign and the required value after the declaration.

Examples:

```
int total = 540;
int extra = 30;
float average = 9.87
```

Assigning Value To Variables

After declaring variables, they must be assigned values before they are accessed in the program. It can be done by using assignment operator(=). The general syntax is as follows:

Variable_name = expression;

Result of right hand side expression is assigned to variable.

Examples:

```
a = 10;
extra = 2;
a = a + 10;
```

Declaring Variable as Constant

It can be done by adding keyword const in the beginning of variable declaration. Const is a data type qualifier defined by ANSI standard.

For example :

```
const int a = 10;
const float pie = 3.14;
```

Defining Symbolic Constant

A symbolic constant is a name that substitutes for a sequence of characters. The character may represent a numeric constant, a character constant or string constant. It is usually defined at the beginning of a program. The general syntax is as follows:

```
#define symbolic_name constant_value
```

Rules For Defining Symbolic Constant

- ✔ No blank space is allowed between sign **'#'** and the word **define.**
- ✔ #define statement must not be end with a semicolon.
- ✔ Symbolic names have the same form as variable name(usually written in capital letters to separate from normal variables).
- ✔ After definition, the symbolic name should not be assigned to any other value.
- ✔ Symbolic names are not declared for data types. Its data types depend on the type of constant.
- ✔ #define statement may appear anywhere in the program but before it is referenced in the program.

Examples:

```
#define  PIE    3.14
#define  true 1
#define  ENGG  "MCA"
#define MAX     50
#Define you "me"    /* INVALID, define should be in lower case letters.
#define P$S 12      /* INVALID, $ symbol is not permitted in name.
```

Enumerated Data Types

Enumerated data type variables can only assume values, which have been previously declared.

```
enum month { jan = 1, feb, mar, apr, may, jun, jul, aug, sep, oct, nov, dec };
```

```
enum month exam_month;
exam_month = feb;
```

In the above declaration, month is declared as an enumerated data type. It consists of a set of values, jan to dec. Numerically, jan is given the value 1, feb the value 2, and so on. The variable exam_month is declared to be of the same type as month, then is assigned the value associated with feb. exam_month cannot be assigned any values outside those specified in the initialization list for the declaration of month.

Type Definition

C allows user to create data types using typedef statement. Once a type is defined, it can be used in same manner as the standard C types. The general syntax is as follows :

```
typedef type new_type;
```

where type refers to existing data type and new_type refers to new name given to data type.

Examples:

```
typedef int rollno;
```

In this declaration rollno is user-defined data type, which is equivalent to int.

```
rollno one, two;
```

is equivalent to

```
int one, two;
```

Similarly

```
typedef char name;
name first,second;
```

Operators

C operators can be classified into a number of categories.

1. Arithmetic operators
2. Relation operators
3. Logical operators
4. Assignment operators
5. Increment and decrement operators
6. Conditions operators
7. Bit wise operators
8. Special operators.

1. Arithmetic Operators

There are five arithmetic operators in C.

Operators	Purpose
+	Addition
-	Subtraction
*	Multiplication
/	Division
%	Percentage

2. Relation Operators

Relational operator is used to compare two operands to see whether they are equal to each other, unequal one is greater or lesser than the other. There are 6 relational operators.

Operators	Purpose
= =	Equal to
!=	Not equal to
<	Less than
<=	Less than or equal to
>	Greater than
>=	Greater than or equal to

3. Logical Operators

Logical operators are used to combine expressions C provides three logical operators.

Operators	Purpose
&&	Logical AND
||	Logical OR
!	Logical NOT

4. Assignment Operators

Assignment operators are used to assign the result of an expression to variable. The most commonly used assignment operators is (=). Note that it is different from mathematical equality.

5. Increment and Decrement Operators

C has two very useful operators ++ and – – called increment and decrement operators respectively. These are generally not found in other language.

There are two types of increment prefix increment (++ a) and postfix increment (a ++). Similarly decrement is also of two types *i.e.*, prefix (– –I) and postfix (I– –). ++ is an increment operator. It increments the value of I by 1. – – is such an operators which decrements the value of a variable by 1.

6. Ternary/Conditional Operators

It is called because it uses three expressions. The ternary operators like a shorthand version of the if–else condition. The general format of ternary operators is

Expression1? Expression 2: Expression 3

This operator's works as follows: Expression 1 is evaluated first, if the result is true then Expression 2 is executed otherwise Expression 3 is executed.

7. Bit wise Operators

Bit wise operators are used for manipulation of data at bit level. These operators are used

for testing the bits or shifting them right or left. Bit wise operators may not be applied to float or double data type. It is applicable to integer data type data only

Operators	Purpose
&	Bit wise logical AND
|	Bit wise logical OR
^	Bit wise logical XOR
<<	Left shift
>>	Right shift
~	One's complement

8. Special Operators

C language supports some special operators such as comma operators, size of operators, pointer operators (& and *) and member selection operators (. and →).

Precedence and Associatively

Precedence and associatively rules of arithmetic, relational and logical operators have already been discussed in earlier chapter. The table given below summarizes these rules for all the operators provided by C. Operators listed higher up the table have higher precedence operators with the same precedence are given in the same row.

Operators	Associatively	Number of Operands
() [] -> .	Left to right	Binary
! ~ - + ++ -- * & (type) size of	Right to left	Unary
* / %	Left to right	Binary
+ -	Left to right	Binary
<< >>	Left to right	Binary
< <= > >==	Left to right	Binary
= = !=	Left to right	Binary
&	Left to right	Binary
^	Left to right	Binary
|	Left to right	Binary
&&|	Left to right	Binary
||	Left to right	Binary
?:	Right to left	Ternary
= + = -= *= /= %= &= ^= | = << = >>=	Left to right	Binary

Type Conversion

It is possible to mix the types of values in your arithmetic expressions. Char types will be treated as int. Otherwise where types of different size are involved, the result will usually be of the larger size, so a float and a double would produce a double result. Where integer and real types meet, the result will be a double.

The problems occur where;

- The variable is too small to hold the value. In this case it will be corrupted.
- The variable is an integer type and is being assigned a real value. The value is rounded down.
- This can lead to corrupt results. The solution is to use a method called casting which convert a value to required type.

Type Casting

It is used for converting the value of an expression to a particular data type. The general syntax is as follows:

```
(data_type)  expression;
```

result of expression is converted into specified data_type.

If operands are of different types, lower type is automatically converted into higher type before the operation proceeds.

Examples :

```
a=(int)12.7   /* 12.7 is converted to integer
x=int(y)+z    /*first y is converted to an integer then added to x
```

Data Conversion

The following functions convert between data types.

atof()	converts an ascii character array to a float
atoi()	converts an ascii character array to an integer
itoa()	converts an integer to a character array.

Example:

```
/* convert a string to an integer */
      #include <stdio.h>
      #include <stdlib.h>
      char str[] = "1352";
      main()
      {
        int num;
        num = atoi( string );
        printf("Num = %d\n", num );
      }

/* convert an integer to a string */

      #include <stdio.h>
```

```
#include <stdlib.h>
main()
{
  int num;
  char str[20];
  printf("Enter in an integer ");
  scanf(" %d", &num );
  printf( "As a  string it is %s\n", itoa( num, str, 10 ) );
}
```

Note that itoa() takes three parameters, the integer to be converted, a character buffer into which the resultant string is stored, a radix value (10 = decimal).

In addition, itoa() returns a pointer to the resultant string.

TEST YOURSELF

1. Describe features of 'C'.
2. Describe data type in C and their properties.
3. Write short note on variables and constant.
4. Write a program to print your name and total marks.
5. Write a program to calculate the sum of three numbers.
6. Write a program to find out the simple interest.
7. What is a Keyword ? What are the restrictions for using them ?
8. What is a C statement ? How it's different from expression ?
9. Write a program to convert Kilometers to meters.
10. Write a program to find circumference of a circle.

□□□

3 INPUT/OUTPUT FUNCTIONS

There are many library functions available for console input/output. These functions are divided into two categories.

- Unformatted
- Formatted

The basic difference between formatted and an unformatted input/output function is that the formatted input/output functions allow input and output as per the requirements of the user.

FORMATTED INPUT/OUTPUT FUNCTIONS

The printf Function

The printf function sends output to the standard output device. To print the value of a variable, a format specifier is embedded in the format string. A format specifier is just a sequence of characters beginning with the % sign. This sequence of characters provides information to printf regarding the variable or expression to be printed. It is possible to print more than one variable value using a single printf function call. The general syntax of printf is

printf ("< fmt_string>",[<arg1> , <arg2> ,.....])

where <fmt_string> refers to a string containing formatting information, and arg1, arg2, ... are arguments that represent the individual output data items. The printf function can be used to print simple messages such as :

Example:

```
printf("This is the first edition of C");
```

output

This is the first edition of C.

Example:

```
int a;
a=40
printf("A=%d ",a);   /* here %d is a format specifier for int */
                     /* if format specifier is not used then the */
                     /* value of a will not be printed */
```

output

A = 40

Several of the most frequently used format specified are listed below :

Fomat Specifier	Meaning
%c	Data item is displayed as a single character.
%d	Data item is displayed as a signed decimal character.
%e	Data item is displayed as a floating point value with an exponential.
%f	Data item is displayed as a floating point value without an exponent.
%ld	Signed long decimal integer.
%u	Unsigned decimal integer.
%x	Unsigned hexadecimal integer.
%s	String.

It takes one argument–a string constant. It prints the string as it is to the standard output. This string is also called a format string. By default printf does not print text on a new line. In order to print on a new line printf must be explicitly informed to do so.

The scanf Function

scanf allows formatted reading of data from the standard input device.

The general syntax of scanf is

scanf ("< fmt_string>",<&arg1> , <&arg2> ,....)

scanf allows formatted reading of data from the keyboard. Like printf it has a control string, followed by the list of items to be read. However scanf wants to know the address of the items to be read, since it is a function which will change that value. Therefore the names of variables are preceeded by the & sign. Since a string is already a character pointer, we give the names of string variables unmodified by a leading.

Special Characters

The following special patterns are used to represent a single character in C programs. The leading backslash in the single quotes indicates that more information is to follow.

Code	Meaning
'\n'	New line
'\t	Tab
'\\'	Backslash
'\'	Single quote
'\"'	Double quote
'\b'	Backspace
'\r'	Carriage return
'\f'	Form feed
'\0'	NULL

UNFORMATTED INPUT/OUTPUT FUNCTION

These unformatted input/output function are used in string input/output function. The function performing input /output of one string at a time known as string input/output functions.

Character Input/Output Functions

For character input the function available are getchar( ), getche( ) and getch( ) where as for output the function available is putchar( ).

1. **getchar ( )**

   This function reads character type data from standard input.This function reads one character at a time.

   Example:

   ```
   Char C;
   .....
   C=getchar();
   ```

2. **getch ( )**

   This function reads any alphanumeric character from standard input. The difference between getchar( ) and getch( ) is that getchar( ) continue access the keyboard until the carriage return key is pressed. It stops accessing the keyboard as soon as key is pressed.

   The getch( ) function does not echo/display the character which is being keyed.

3. **getche( )**

   The getche( ) function also accepts a character but it also displays the entered character, which getch( ) does not do.

4. **putchar( )**

   This function prints one character at a time, which is taken by standard input.

   putchar(char);

String Input/Output Functions

For string input, the function is gets () is used and for string output, the function is puts () is used.

1. **gets ( )**

   This function accepts a string from the standard input device. The length of the string is limited by the declaration of the string variable. The input string may consist of multiple words. The input is complete by typing Enter key.

   Syntax : gets(char *)

2. **puts ( )**

   This function outputs a string constant or string variable to the standard output device. (This function prints the string which is already accepted in the character type array.)

   Syntax : puts(char *)

Example:

```
#include<stdio.h>
main()
{
char  arr[100];
printf(" Enter the string ");
gets(arr);
printf(" The accepted  string is ");
puts(arr);
}
```

□□□

4 CONDITIONAL STATEMENT

INTRODUCTION

C provides the conditional statement for decision-making. C has two major decision making statements.

1. if-else statement
2. switch statement

if–else Statement

The if-else statement is used to carry out a logical test and depending on the condition evaluation one of the two possible action is taken. The general syntax for if-else is as follows:

```
if(expression)
      stmt1;      //executed if condition is true
else
      stmt2;      //executed if condition is false
```

where expression specifies the condition to be checked. Statements can be simple or blocked.

Block is used when more than one statement is to be executed, when a condition is true or false.

Block is made by enclosing the statements between curly braces { }.

```
{
   statement-1;
   statement-2;
   .........
   statement-n
}
```

Example 1. *To check number whether even or odd.*

Solution :

```
#include<stdio.h>
#include<conio.h>
main()
{
   int a,b;
   clrscr();
   printf("Enter the number : -");
```

```
    scanf("%d",&a);
    b=a%2;
    if(b==0)
    printf("Number Is Even");
    else
    printf("Number Is Odd");
    getch();
}
```

Example 2. *To check whether the year is leap year or not.*

Solution :

```
#include<stdio.h>
#include<conio.h>
void main()
{
   int a;
   clrscr();
   printf("Enter Year ");
   scanf("%d",&a);
   if(a%100==0)
   {
   if(a%400==0)
   printf("leap year");
   else
   printf("not a leap year");
   }
   else if(a%4==0)
   printf("leap year");
   else
   printf("not a leap year");
   getch();
}
```

Nesting—if-else Statement

When one of number of decisions to be made in program, we require more than if-else statement in nested form. The syntax for that is as follows:

```
if(exp1)
{
   stmt1;
   .......;
   if(exp2)
   {
   stmt2;
   .......;   } may contain nested if-else statement
   }
   else
   {
   stmt3;
   .......;   } may contain nested if-else statement
   }
   else
```

```
{
stmt4;
.......;
}
```

Example 3. *To check whether a given character is digit, special character, and letter.*

Solution :

```
#include<stdio.h>
#include<conio.h>
main()
{
   char a;
   clrscr();
   printf("enter the character");
   scanf("%c",&a);
   if(a<=57 && a>=48)
   {
   printf("Given input is digit");
   }
   else
   if(a<=90 && a>=65) || (a<=122 && a>=97)
   {
   printf("Given input is letter");
   }
   else
   {
   printf("Given input is special character");
   }
   getch(); }
```

Example 4. *Program to find out maximum and minimum out of three numbers.*

Solution :

```
#include<stdio.h>
#include<conio.h>
main()
{
   int a,b,c;
   printf("Enter the three value:-");
   scanf("%d %d %d",&a,&b,&c);
   if(a>b && b>c)
   {
      printf("a is greatest");
      if(b>c)
      printf("c is lowest");
      else
      printf("b is lowest");
   }
   else if(b>c)
   {
      printf("b is greatest");
      if(a>c)
      printf("c is lowest");
```

```
        else
        printf("a is lowest");
    }
    else
    {
      printf("c is greatest");
        if(a>b)
        printf("b is lowest");
        else
        printf("a is lowest");
    }
getch();
}
```

Example 5. *Program to read marks of four subjects and print grade*

Percentage	*Grade*
>=85	*A*
>=70,<85	*B*
>=55,<70	*C*
<55	*D*

Solution :

```
#include<stdio.h>
#include<conio.h>
void main()
{
   int m1,m2,m3,m4,t,p;
   clrscr();
   printf("Enter marks ");
   scanf("%d  %d  %d  %d ",&m1,&m2,&m3,&m4);
   t=m1+m2+m3+m4;
      p=t*100/400;
      printf("percentage of student is %d=",p);
      if(p>=85)
   {
      printf("grade A");
   }
   else
   if(p>=70 && p<85)
   {
      printf("grade B");
   }
   else
   if(p>=55 && p<70)
   {
      printf("grade C");
   }
   else
   {
      printf("grade D");
   }
   getch();
}
```

Example 6. *Write a program in C with nested if else tahth will print different messages at different times of the day. The messages to be printed are :*

12 midnight – 12 Noon	*Good morning*
12 Noon – 4	*Good after noon*
4 PM – 7 PM	*Good evening*
7 PM – 12 midnight	*Good night*

else time invalid.

Solution :

```
#include <stdio.h>
main()
int hour,min ,sec;
printf(" enter current military  time in hours, minutes and seconds \n");
scanf(%d%d %d" & hours & min ,& sec);
if( (hours >24) || (min > 60) || (sec > 60) )
{
   printf (" time Invalid \ n");
exit(1);
}
if(hour s<=12)
printf(" good morning \n");
else if( (hours>12)  && (hours<=16) )
printf(" good  after noon \n");
else if( (hours>16)  && (hours<=19) )
printf(" good  evening \n");
else
 printf(" good night \n");
}
```

THE SWITCH STATEMENT

C has a built-in multi-way decision statement known as switch. The switch statement tests the value of given variable against a list of case values and when a match is found a block of statements associated with that case is executed.

The general syntax of switch statement is as follows:

```
switch(expression)

{
   case expression 1:
            stmt1;
            break;
   case expression 2:
            stmt2;
            break;
   case expression 3:
            stmt3;
            break;
   case expression 4:
            stmt4;
            break;
```

```
.
.
.
.
case expression n:
        stmt1;
default:
        stmt;
}
```

where expression 1, expression2, ..., expression m represent constant, integer-valued expressions. Generally, each of these expressions will be written either an integer constant or a character constant. Each individual statement following the case labels may be either simple or complex.

Both if and switch constructs allow the programmer to make a selection from a number of possible actions.

Rules for Switch Statements

- Values for 'case' must be integer or character constants
- The order of the 'case' statements is unimportant
- The default clause may occur first (convention places it last)
- You cannot use expressions or ranges

Example 7. *Read a character and print the following messages:*

Character	***Message***
F or f	*First*
S or s	*Second*
T or t	*Third*
Otherwise	*Wrong choice*

Solution :

```
#<include<stdio.h>
main()
{
char ch;
printf("Enter your choice ?");
scanf(" %c ",&ch);
switch(ch);
{
case 'f':
case 'F':
printf(" First \ n");
break;
case  's':
case 'S':
printf(" Second \n");
break;
case 't':
case 'T':
printf(" Third \n");
break;
```

```
default :
printf( " wrong choice \n");
}
}
```

Run

Enter your choice ? f

First

Run

Enter your choice ? s

Second

Run

Enter your choice ? t

Third

Run

Enter your choice ? r

Wrong choice

Run

Enter your choice ? p

Wrong Choice

Example 8. *Print the number is even or odd.*

Solution :

```
#include<stdio.h>
main()
{
int n,ch;
printf(" enter the number:");
scanf("%d",&n);
if(n%2==0)
   ch=1;
else
   ch=2;
switch(ch)
{
case 1:
        printf(" number is even\n");
        break;
case 2:
        printf(" number is odd\n");
        break;
}
}
```

7. Suppose the postal rates for mailing letters are as follow:
   Rs. 0.75 per 5 grams for the first 50 grams
   Rs. 0.50 per 5 grams for the next 100 grams
   Rs. 0.40 per 5 grams for the next 250 grams
   Rs. 0.25 per 5 kilograms for letters weighing more than 500 grams.
   Write a program, which prompts for the weight of a letter and prints the postage to be paid.
8. Given a list of marks ranging from 0 to 100, write a program to compute and print the number of students :
   (a) who have obtained more than 80 marks,
   (b) who have obtained more than 60 marks,
   (c) who have obtained more than 40 marks,
   (d) who have obtained 40 and less 40 marks.
9. A cloth showroom has announced the following seasonal discount on purchase of items:

Purchase Amount	Discount
0–200	5%
201–400	15%
401–600	20%
Above 600	25%

Write a program to compute net amount paid by the customer.

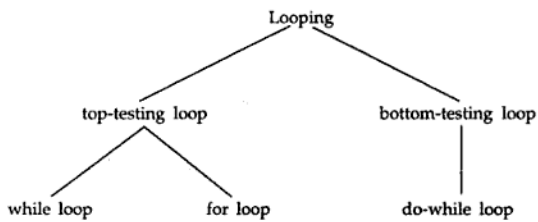
□□□

5 LOOPING

INTRODUCTION

Looping is repetitive execution of specific set of instructions. In large programs we have to perform same calculation for each set of data, solution using sequential approach is worth lengthy and less efficient. So using looping statements we can represent such problems efficiently and can decrease complexity of our program.

C language provides mainly two types of looping structure. Those are top-testing loop and bottom testing loop. In top-testing loop first condition is checked then execution of loop proceed and in second one firstly at once loop executed then condition checked. So bottom-testing loop executed at least once.



Looping Structure in 'C'

- ✔ 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 loop is similar, but the test occurs after the loop body is executed. This ensures that the loop body should 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 Loop

The general syntax for while loop is following:

```
while (expression)
{
   stmt1;
   ......;
   ......;        while block
}
```

Where expression is condition to perform calculation inside while block. Execution of while block continues till condition specified is true. If only single statement in while then block is optional.

Example 1. *To calculate factorial.*

$n! = n \times n\text{–}1!$

$5! = 5 \times 4! = 5 \times 4 \times 3! = 5 \times 4 \times 3 \times 2! = 5 \times 4 \times 3 \times 2 \times 1.$

Solution :

```
#include<stdio.h>
#include<conio.h>
void main()
{
   int a,num,fact=1;
   clrscr();
   printf("Enter a number : ");
   scanf("%d",&num);
   a=num;
   while(num>0)
   {
   fact=fact*num;
      num=num-1;
   }
   printf("Factorial of  %d is  = %d",a,fact);
}
```

Example 2. *To check whether a number is palindrome or not.*

Solution : A number is palindrome if reverse of a number is equal to given number *i.e.*, 121.

```
#include<stdio.h>
#include<conio.h>
void main()
{
   int a,b,c;
   clrscr();
   printf("Enter a number : ");
   scanf("%d",&n);
   a=n;
   while(n>0)
   {
   x=n%10;
      sum=sum*10+x;
```

```
    n=n/10;
    }
    if(sum==a)
       printf("Number is palindrome");
    else
       printf("Number isn't a palindrome");
    getch();
}
```

Example 3. *To check whether a number is Armstrong or not.*

Solution : A number is Armstrong if sum of cube of each digit of a number equal to a number is known as Armstrong number *i.e.*, 153.

$153 = 1^3 + 5^3 + 3^3 = 1 + 125 + 27 = 153$.

```
#include<stdio.h>
#include<conio.h>
void main()
{
   int a,sum=0,n,x;
   clrscr();
   printf("Enter a number : ");
   scanf("%d",&n);
   a=n;
   while(n>0)
   {
   x=n%10;
      sum=sum+x*x*x;
   n=n/10;
   }
   if(sum= =a)
      printf("Number is Armstrong");
   else
   printf("Number isn't a Armstrong");
   getch(); }
```

Example 4. *To count the digits and give the sum of the digits of a given number.*

Solution :

```
#include<stdio.h>
#include<conio.h>
void main()
{
   int c=0,sum=0,n,x;
   clrscr();
   printf("Enter a number : ");
   scanf("%d",&n);
   while(n>0)
   {
   x=n%10;
   sum=sum+x;
   n=n/10;
```

```
    c++;
    }
    printf("\n Sum of digits is %d is = ",sum);
       printf("\n The no. of digits is %d ",c);
    getch();
}
```

Example 5. *To convert an entered binary number into equivalent decimal number.*

Solution :

```
#include<stdio.h>
#include<conio.h>
#include<math.h>
void main()
{
   long int bin,x,num =0,i=0,
   printf("Enter the binary number :  ");
   scanf("%d",& bin);
   while(b>0)
   {
   x=bin%10;
   num=num +pow(2,i)+x;
   bin = bin/10;
   i++;
   }
   printf("Equivalent decimal no. is %d", num);
   getch();
}
```

Example 6. *To convert a decimal number into equivalent binary number.*

Solution :

```
#include<stdio.h>
#include<conio.h>
#include<math.h>
void main()
{
   long int num b=0,x,i=0,
   printf("Enter the decimal no. :  ");
   scanf("%d",& dec);
   while(dec>0)
   {
   x=dec%2;
   dec=dec+pow(10,i)+x;
   b=b/10;
   i++;
   }
   printf(" Equivalent Binary no. = %d",b);
   getch();
}
```

for Loop

The general syntax for while loop is following:

```
for(exp1;exp2;exp3)
{
   stmt1;
   stmt2;        } for block
   ..........
}
```

Where exp1 is an assignment to initialize some variable, exp2 is a relational expression and exp3 is an assignment to increment or decrement the value of some variable. Statement can be a simple or a compound statement. Where all three expressions are optional.

The execution of for loop proceeds as follows. exp1 is executed once before the start of loop. exp2 is then evaluated to determine if statement should be executed. If exp2 evaluates to a non-zero value, entry is made into the loop and statement has been executed. This completes one iteration of the loop.

To continue the loop execution, exp2 is evaluated again. If it returns a non-zero value, entry is again made into the loop to be execute statement. Thus each iteration starts with evaluation of exp2, proceeds to execution of statements (if exp2 returns a non zero value) and then ends with evaluation of exp3.

Example 7. *To generate a table of a number.*

Solution :

```
#include<stdio.h>
#include<conio.h>
void main()
{
   int num,I,t;
   clrscr();
   printf("enter the no. of which table is required ");
   scanf("%d",&num);
   for(I=1;I<=10;I++)
   {
   t=num*I:
   printf("%d*%d=%d\n",num,I,t);
   }
   getch();
}
```

Example 8. *To generate series 1 + 8 + 27 + 64 + 125 + ...*

Solution :

```
#include<stdio.h>
#include<conio.h>
void main()
{
   int i,n,a;
   clrscr();
   printf("enter last number");
   scanf("%d",&n);
   for(i=1;i<=n;i++)
```

```
    {
    a=i*i*i;
    printf("%d+",a);
    }
    printf("...");
    getch();
}
```

Example 9. *To calculate sum of N numbers.*

Solution :

```
#include<stdio.h>
#include<conio.h>
main()
{
   int n,sum=0,no,i;
   clrscr();
   printf("Enter the number:-");
   scanf("%d",&n);
   for(i=1;i<=n;i++)
   {
   printf("Enter the Number #%d:-",i);
   scanf("%d",&no);
   sum=sum+no;
   }
   printf("Sum of %d no is=%d",n,sum);
   getch();
}
```

Do-while Loop

The general syntax of do-while loop is as follows:

```
do
{
   stmt1;
   stmt2;          ] do-while block
   .........;      ]
} while(expression);
```

In this loop-condition check at bottom on exit of block so it is also known as exit-controlled loop. Thus the do-while block executed at least once.

```
do
{  printf("Enter 1 for yes, 0 for no :");
   scanf("%d",  &input_value);
}  while (input_value != 1 && input_value != 0)
```

Example 10. *To print a message "Hello World" 100 times.*

Solution :

```
#include <stdio.h>
main()
{
   int count;
   count=0;
   do
```

```
    {
        ++count;
        printf("Hello, World!\n");
    } while (count < 100)
}
```

Difference between while and do while:

- In 'do... while' loop a single statement or a block of statement are executed at least once and then the condition is evaluated.
- In a 'while' loop first condition is evaluated and then the statements are executed.

Nesting Loop

If a loop may contain another loop then loop is called nested loop.

```
for(exp1;exp2;exp3)
{
    stmt1;
    for(exp4;exp5;exp6)
    {
    stmt2;
    .........;
    }
    .........;
    stmtn;
}
```

Loop execution starts from top loop then if condition is true then first all iteration of inner loop have been executed. After that again execution go on top loop. So for each time each true condition of top loop whole inner loop executes.

Example 11. *To generate the pattern given below:*

1

121

12321

1234321

Solution :

```
#include<stdio.h>
#include<conio.h>
void main()
{
    int i,j;
    clrscr();
    printf("\n\n\n\n");
    for(i=1;i<=5;i++)
    {
    for(j=1;j<=5-i;j++)
    printf(" ");
    for(j=1;j<i;j++)
    printf("%d",j);
    for(;j>=1;--j)
    printf("%d",j);
```

```
    printf("\n");
    }
    getch();
}
```

Example 12. *To generate the pattern given below:*

```
*
**
***
****
*****
```

Solution :

```
#include<stdio.h>
#include<conio.h>
void main()
{
   int i,j,n;
   clrscr();
   printf("enter no. of lines :");
   scanf("%d",&n);
   printf("\n\n\n\n");
   for(i=1;i<=n;i++)
   {
   for(j=1;j<=n-i;j++)
   printf(" ");
   for(j=1;j<=i;j++)
   printf("*");
   printf("\n");
   }
   getch();
}
```

Example 13. *To generate the given series:* $1 - 4 + 9 - 16 + 25 - .....$

Solution :

```
#include<stdio.h>
#include<conio.h>
main()
{
   int i,n ;
   clrscr();
   printf("Enter the number:-");
   scanf("%d",&n);
   for(i=1;i<=n;i++)
   {
      if(i%2==0)
         printf("%d-",i);
      else
         printf("%d+",i);
   }
   printf("\b.......");
   getch();
}
```

Jump in Loops

The Break Statement

It is used for immediately exit from the loop in which it is included and control transfers to first statement after that loop or it is used for immature termination of a loop.

Example :

```
main()
{
stmt-1;
.
while(expr1)
{
stmt2;
.
.
if(expr2)
break;
stmt3
.
}
stmt-n;  /*this statement is first statement which is outside the while loop*/
}        /*and control will be transferred when the break statement encounters*/
```

in the above example when the control comes on **break statement,** the control will be transferred on the stmt-n.

Example :

```
main()
{
stmt1;
...
while(expr1)
{
   while(expr2)
   {
   ....
   if(expr3)
   break;
   stmt2;
   .....
   }
stmt3;
}
stmt4;
}
```

In that when control comes on break , then the control is transferred on the stmt3. So, the break statement will provide exit from a single loop.

Example 14. *Program to check whether a number is prime or not.*

Solution :

```
#include<stdio.h>
#include<conio.h>
```

```
void main()
{  int n,I;
   clrscr();
   printf("Enter number :");
   scanf("%d",&n);
   for(I=2;I<n/2;I++)
   {  if(n%I==0)
      break;
   }
   if(I<n/2)
   printf("Not Prime");
   else
   printf("Prime");
   getch();     }
```

Continue Statement

It allows skip statements in loop followed by continue statement while continuing next iteration.

Example :

```
while(expr1)
{
stmt2;
.
.
if(expr2)
continue;
stmt3; . . .
.}
stmtn;
}
```

In that when control comes on continue it transferred on while stmt.

Example 15. *To calculate sum and average non-negative n numbers.*

Solution :

```
main()
{
int k,n,x;
float sum=0 , avg=0;
printf("Enter number of numbers : ");
scanf("%d",&n);
for(k=0;k<n;k++)
{
printf("\nEnter number : ")
scanf("%d",&x);
if(n<0)
continue;
sum=sum+x;
}
avg=sum/n;
printf("\nSum= %f\n Average = %f",sum,avg);
getch();
}
```

Example 16. *To generate the pattern given below:*

```
        3
      3 2 3
    3 2 1 2 3
      3 2 3
        3
```

Solution :

```
#include<stdio.h>
#include<conio.h>
main()
{
   int i,j,k,l,n;
   clrscr();
   printf("enter the number of lines: ");
   scanf("%d",&n);
   for(i=n;i>=1;i--)
   {
   for(j=1;j<i;j++)
      printf(" ");
      for(k=n;k>=i;k--)
         printf(" %d ",k);
   for(l=k+1;l<n;l++)
         printf(" %d ",l+1);
      printf("\n");
   }
   for(i=1;i<=n;i++)
   {
   for(j=1;j<=i;j++)
      printf(" ");
      for(k=n;k>i;k--)
         printf(" %d ",k);
      for(l=k+1;l<n;l++)
         printf(" %d ",l+1);
         printf("\n");
   }
   getch();
}
```

Example 17. *To generate the pattern given below:*

```
1
12
123
1234
```

Solution :

```
#include<stdio.h>
#include<conio.h>
main()
{
```

```
    int i,j,k,n,l;
    clrscr();
    printf("Enter the number of lines : ");
    scanf("%d/n",&n);
    for(i=0;i<=n;i++)
  {
    for(j=1;j<=i;j++)
    {
      printf("%d",j);
    }
    printf("\n");
  }
}
```

Example 18. *To generate the pattern given below:*

1
12
123
1234
123
12
1

Solution :

```
#include<stdio.h>
#include<conio.h>
main()
{
   int i,j,k,n,l;
   clrscr();
   printf("Enter the number of lines : ");
   scanf("%d/n",&n);
   for(i=0;i<=n;i++)
   {
   for(j=1;j<=i;j++)
   {
   printf("%d",j);
   }
   printf("\n");
   }
   for(i=1;i<=n-1;i++)
   {
   for(j=1;j<=n-i;j++)
   {
   printf("%d",j);
   }
   printf("\n");
   }
   getch();
}
```

Example 19. *To generate the pattern given below:*

1
01
101
0101
10101

Solution :

```
#include<stdio.h>
#include<conio.h>
main()
{
   int i,j,n;
   printf(" enter number of lines : ");
   scanf("%d",&n);
   for(i=1;i<=n;i++)
   {
   for(j=i;j>=1;j--)
   printf("%d",j%2);
   printf("\n");
   }
   getch();
}
```

Example 20. *Write a program to find sum of even and odd numbers between 1 to 100.*

Solution :

```
#include<stdio.h>
#include<conio.h>
void main()
{
   int  I, m, sum,sum1;
   clrscr();
   sum = 0;
   sum1 = 0;
   for(I=1;  I<=100;I++)
   {
      m = I%2;
      if(m==0)
         sum = sum + I;
      else
         sum1 = sum1+I;
   }
   printf("The sum of even numbers between  1  and  100 is : %d", sum);
   printf("\n The sum of odd numbers  1  and  100 is : %d",sum1);
   getch();
}
```

Example 21. *Write a program to generate the pattern:*

```
54321
4321
321
21
1
```

Solution :

```
#include<stdio.h>
#include<conio.h>
void main ()
{
   int I,j;
   for(I =5; I>=1; I--)
   {
   for(j =I; j<=1; j--)
   {
   printf("%d", j);
   }
   printf("\n");
   }
   getch();
}
```

Example 22. *Write a program to generate the pattern:*

```
A
AB
ABC
ABCD
ABCDE
```

Solution :

```
#include<stdio.h>
#include<conio.h>
void main()
{
int I,j;
for(I=1; I<=5; I++)
{
for(j=1;j<=I;j++)
{
if(j==1)
printf("A");
else if(j==2)
printf("B");
else if(j==3)
printf("C");
else if(j==4)
printf("D");
```

```
else if(j==5)
printf("E");
}
printf("\n");
}
getch();
}
```

TEST YOURSELF

1. What is the difference between while and do-while statement ?
2. Can there be multiple increment expressions in increment part of the for statement ?
3. Write a program, to print the numbers, which are divisible by 7 from first 100 natural numbers.
4. Write a program to read a five digit number if it is even then add up the digits otherwise multiply them and print the result.
5. Write a program to calculate print Armstrong numbers between 1 to 500 and their sum.
6. Write a program to calculate print prime number between 1 to 100 and their sum.
7. Write a program to evaluate the power series :

   $e^x = 1 + x + x^2/2! + \ldots\ldots\ldots + x^n/n!, \quad 0 < x < 1.$
8. Write a program to find factorial up to n numbers

   i.e., n = 4

1!	1
2!	2
3!	6
4!	24
9. Write a program to find first n^{th} prime number.
10. Write a program to generate Pascal triangle.

□□□

6 ARRAYS IN C

There are times when we need to store a complete list of data items. You could do this by creating as many individual variables as would be needed for the job, but this is a hard and tedious process. That could be like that:

```
int n1 = 501;
int n2 = 22;
int n3 = 71;
```

It becomes increasingly more difficult to keep track of this as the number of variables increase. Arrays offer a solution to this problem.

ARRAY

An array is a collection of variables of the same type. Individual array elements are identified by an integer index. In the case of C you have to declare an **array** before you use it as you declare any type of variable.

Syntax for Array Declaration:

Data-type array_name[size]

declares an **array** of the specified type and with size elements.

Example:

int arr[15];

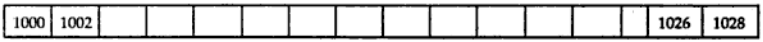
declares an array called **a** with fifteen elements.

Accessing Array Element

Individual element of an array can be accessed using an index or subscript number. This index specifies the element's position in the array and always an integer number. In C, array elements start from 0^{th} position and last position will be size-1. So, index set in C contains values from 0 to size-1. *i.e.,* a[5] is used to access fifth element.

An array in memory is represented as shown below:

int arr[15];



Assigning values to array elements is done by,

arr[10] = 25;

and assigning array elements to a variable is done by,

g = x[10];

An Initializing an Array

It is possible to initialize the elements of an array at the declaration time. The initial values must appear in the order in which they will be assigned to the individual array elements, separated by commas and enclosed in braces.

Example: The following definition

int arr [4] = {12, 21, 3, 52};

causes an array of integer of size 6, with the initial values of the elements 5,6,7 and 8,9,10 respectively.

int a[ ] = {5, 6, 7, 8, 9, 10}

causes an array of integer of size 6, with the initial values of the elements 5, 6, 7 and 8, 9, 10 respectively.

Here size is not specified, but the size will be assumed in basis of the list of values.

Rules for using Array:

- ✔ The number of objects in the array must be a constant and must be mentioned at the time of writing the program. That is, the size of the array is fixed.
- ✔ All the elements of the array are stored in a contiguous area in the memory.
- ✔ Any integer expression can be used for the subscript to refer to a particular element of the array.
- ✔ The index of the first element in an array is always 0.
- ✔ If an index value that refers to a non-existent object is used, then the results of the program will be unpredictable.
- ✔ Arrays obey the same scope rules, as do other objects in C.

Array Processing

C does not provide operators that can work with entire arrays *i.e.*, adding the elements of two arrays a1 and a2 into a third array total is not possible with the expression :

Total = a1 + a2 ;

This is invalid. Thus, addition, assignment, comparison operation etc. must be carried out on an element -by-element basis.

The processing of an array is generally carried out in three steps. These steps are given below:

1. Read the value in the elements of the array.
2. Perform the desired processing on the elements.
3. Print out the results of the processing.

Read Array Elements Value

It can be explained by the following example:

```
main()
{
int num[10];
int i;
for(i = 0; i<10 ; i++)
{
printf("\n Enter  %d element :  ", i+1);
scanf("%d",&num[i]);
}
}
```

In this example, using the for loop, the process of reading data of array num is accomplished.

Output Array Elements Value

Now to output values from array, we will again use for loop to access each cell.

```
for(count =0;count<6;count++)
{
   printf("\n %d value =",num[count]);
}
```

Processing Array Elements

Example 1. *Program to print the sum of 10 numbers using array.*

Solution :

```
#include<stdio.h>
main()
{
int arr[10],I,sum=0;
for(I=0;I<10;I++)
{
printf(" Enter the %d number \n", I+1);
scanf(" %d" ,& arr[I]);
}
for(I=0;I<10;I++)
sum=sum+arr[I];
printf(" sum = %d \n",sum);
}
```

Run: Enter 1 number : 10
Enter 2 number : 10
Enter 3 number : 10
Enter 4 number : 10
Enter 5 number : 20
Enter 6 number : 10
Enter 7 number : 10
Enter 8 number : 10
Enter 9 number : 20

Enter 10 number : 20
Sum = 130

Example 2. *A program to find average marks obtained by a class of 10 students in a test.*

Solution :

```
main()
{
float avg,sum=0;
int i;
int marks[10];
for(i=0;i<10;i++)
{
printf("\n Enter marks :\t");
scanf("%d",&marks[i]);
}
for(i=0;i<10;i++)
sum=sum+marks[I];
avg=sum/10.0;
printf("\n average marks : %f ",avg);
}
```

Run: 10 5 5 5 5.5 10 5 10 8
average marks : 6.00

Example 3. *To delete an element from an array.*

Solution :

```
#include<stdio.h>
#include<conio.h>
main()
{
int a[20],i,n,p;
clrscr();
printf("Please enter size of an array(<20) : ");
scanf("%d",&n);
for(i=0;i<n;i++)
{
printf("Enter the %dth element of an array : ",i);
scanf("%d",&a[i]);
}

printf("Enter the position for deleting an element: ");
scanf("%d",&p);
for(i=p;i<n;i++)
{
a[i]=a[i+1];
}
for(i=0;i<n-1;i++)
printf("%d\n",a[i]);
getch();
}
```

Example 4. *To reverse an array.*

Solution :

```
#include<conio.h>
#include<stdio.h>
main()
{
int a[10],i,temp,j;
clrscr();
for(i=0;i<10;i++)
scanf("%d",&a[i]);
for(i=0,j=9;i<5;i++,j--)
{
temp=a[i];
a[i]=a[j];
a[j]=temp;
}
for(i=0;i<10;i++)
{
printf("%d",a[i]);
}
getch();
}
```

Output

Example 5. *Programs to arrange data items in a particular order (Sorting).*

Solution : Straight Selection Sort :

```
#include<stdio.h>
main()
{
int I,,j,temp;
static int a[10];
for(I=0;I<10;I++)
{
printf("\n Enter  %d number  :", I);
scanf("%d",&a[I]);
}
for(I=0;I<9;I++)
   for(j=I+1;;j<10;;j++)
   if(a[I]>a[j])
   {
   temp=a[I];
   a[I]=a[j];
   a[j]=temp;
   }
for(I=0;I<10;I++)
printf("%d\n",a[I]);
}
```

Run: Enter 1 number : 12
Enter 2 number : 10
Enter 3 number : 11
Enter 4 number : 18
Enter 5 number : 20

Enter 6 number : 5
Enter 7 number : 10
Enter 8 number : 11
Enter 9 number : 22
Enter 10 number : 28

Output in Different Phases:

Phase-1	5	12	11	18	20	10	10	11	22	28
Phase-2	5	10	12	18	20	11	10	11	22	28
Phase-3	5	10	10	18	20	12	11	11	22	28
Phase-4	5	10	10	11	20	18	12	11	22	28
Phase-5	5	10	10	11	11	20	18	12	22	28
Phase-6	5	10	10	11	11	12	20	18	22	28
Phase-7	5	10	10	11	11	12	18	20	22	28
Phase-8	5	10	10	11	11	12	18	20	22	28
Phase-9	5	10	10	11	11	12	18	20	22	28

Position Selection Sort:

```
#include<stdio.h>
main()
{
   int I,,j,temp,k,s;
   static int a[10];
   for(I=0;I<10;I++)
   {
   printf("Enter  %d number ", I );
   scanf("%d",&a[I]);
   }

   for(I=0;I<9;I++)
   {
   s=a[I];
   k=I;
      for(j=I+1;j<10;j++)
      if(a[j]<s)
      {
      s=a[j];
      k=j;
      }
temp=a[I];
a[I]=a[k];
a[k]=temp;
}
```

```
    for(I=0;I<10;I++)
    printf("%d\n",a[I]);
}
```

Run: Enter, 1 number : 12
Enter 2 number : 10
Enter 3 number : 11
Enter 4 number : 18
Enter 5 number : 20
Enter 6 number : 5
Enter 7 number : 10
Enter 8 number : 11
Enter 9 number : 22
Enter 10 number : 28

Output in Different Phases:

Phase-1	5	10	11	18	20	12	10	11	22	28
Phase-2	5	10	11	18	20	12	10	11	22	28
Phase-3	5	10	10	18	20	12	11	11	22	28
Phase-4	5	10	10	11	20	12	18	11	22	28
Phase-5	5	10	10	11	11	12	18	20	22	28
Phase-6	5	10	10	11	11	12	18	20	22	28
Phase-7	5	10	10	11	11	12	18	20	22	28
Phase-8	5	10	10	11	11	12	18	20	22	28
Phase-9	5	10	10	11	11	12	18	20	22	28

Example 6. *Program to perform sorting using Bubble sort method.*

Solution :

```
#include<stdio.h>
main()
{
   int I,,j,temp;
   static int a[10];
   for(I=0;I<10;I++)
   {
   printf("Enter  %d number ", I );
   scanf("%d",&a[I]);
   }
      for(I=0;I<9;I++)
         for(j=9;j>=I+1;j--)
         if(a[j] < a[j-1])
```

```
            {
            temp=a[j];
            a[j]=a[j-1];
            a[j-1]=temp;
            }
        for(I=0;I<10;I++)
        printf("%d\n",a[I]);
}
```

Run: Enter 1 number : 89
Enter 2 number : 5
Enter 3 number : 62
Enter 4 number : 12
Enter 5 number : 10
Enter 6 number : 54
Enter 7 number : 1
Enter 8 number : 21
Enter 9 number : 25
Enter 10 number : 4

Output in Different Phases:

Phase-1	1	89	5	62	12	10	54	4	21	25
Phase-2	1	4	89	5	62	12	10	54	21	25
Phase-3	1	4	5	89	62	10	12	21	54	25
Phase-4	1	4	5	10	89	12	62	21	25	54
Phase-5	1	4	5	10	12	89	21	62	25	24
Phase-6	1	4	5	10	12	21	89	25	62	54
Phase-7	1	4	5	10	10	21	25	89	54	62
Phase-8	1	4	5	10	12	21	25	54	89	62
Phase-9	1	4	5	10	12	21	25	54	62	89

Searching Problems

Linear Search: In linear search the target element is compared with the array element one by one until the desired element is matched. If desired element is matched the search operation is successful . If during the searching process the entire array elements are compared and there is no match , it means that the target element is not present.

```
#include<stdio.h>
main()
{
   int  a[10],i,j,v,flag=0,pos;
```

```
    clrscr();
    for(i=0;i<10;i++)
    scanf("%d",&a[i]);
    printf("Enter the element which u want to search:-");
    scanf("%d",&v);
    for(i=0;i<10;i++)
    {

    if(a[i]==v)
    {
    flag=1;
    pos=i+1;
    break;
    }
  }
  if(flag==1)
  printf("Found at position %d",pos);
  else
  printf("Sorry, not found");
  getch();
  }
```

Run:

5 16 12 15 14 10 11 13 14 11

enter element to be search: 5

found at position 1

Or

enter element to be search: 45

Sorry, not found

Binary Search

- In binary search the array must be sorted.
- Binary search is faster searching method than linear search. It is based on divide and conquer rule.

A[ ] ->	11	12	13	14	15	16	17	18	19	22

Low high mid = (low + high)/2

0 9 4

is num[mid] equals target , if yes then output matched

is num[mid] greater than target, if yes then high = mid – 1

is num[mid] less than target, if yes then low = mid + 1

i.e., target element = 12

I pass –> a[mid]>tar so high= 4–1= 3

II pass –> a[mid] == tar then success

```
#include<stdio.h>
main()
```

```
{
int a[10],i,j,v,flag=0,pos,low=0,high=9,mid;
clrscr();
printf("Please enter sorted array \n");
for(i=0;i<10;i++)
scanf("%d",&a[i]);
printf("Enter the element which u want to search:-");
scanf("%d",&v);
while(low<high)
{
mid=(low+high)/2;
   if(a[mid]==v)
   {
      flag=1;
      pos=mid+1;
      break;
   }
   else if(a[mid]>v)
         high=mid-1;
   else
         low=mid+1;
   }
   if(flag==1)
printf("Found at position %d",pos);
else
printf("Sorry, not found");
getch();
}
```

Two Dimensional Arrays

It is possible to have an array of more than one dimension. Two-dimensional array (2-D array) is an array of number of 1-dimensional arrays. A two dimensional array is also called a matrix or table. In C language 2-D arrays are stored as Row-Major-Order. Row-Major-Order means that 1st Row will get storage for their location first and so on.

Two-dimensional arrays declared as follows:

datatype array_name [row_size][col_size];

A_{00}	A_{01}	A_{02}
A_{10}	A_{11}	A_{22}
A_{20}	A_{21}	A_{02}

A 2-D array of 3 × 3

Initializing 2-D Array

Two dimensional array can be initialized at the time of declaration.

Example: int a[3][3] = {1, 2, 3, 4, 5, 6, 7, 8, 9};

Or

int a[3][3] = { {1, 2, 3}, {4, 5, 6}, {7, 8, 9} };

Accessing 2-D Array Element

In two dimensional array accessing element requires two index variables, where one for referencing first dimension in an array while other for second dimension.

i.e., a[3][4] is used to access element of 3rd row or 4th column.

Processing 2-D Array

The processing of an array is generally carried out in three steps. These steps are given below:

1. Read the value in the elements of the 2-D array.
2. Perform the desired processing on the elements.
3. Print out the results of the processing.

Example 7. *Program to read a matrix of size 3 × 3 and display it.*

Solution :

```
#include<stdio.h>
#include<conio.h>
main()
{
int a[3][3],b[3][3],i,j;
for(i=0;i<3;i++)
{
for(j=0;j<3;j++)
{
printf("\n Enter the matrix element :");
scanf("%d",&a[i][j]);
}
}
printf("\n Entered Matrix  is : \n");
for(i=0;i<3;i++)
{
for(j=0;j<3;j++)
printf("%6d",a[i][j]);
printf("\n");
}
}
```

Run:

```
Enter matrix element : 1
Enter matrix element : 2
Enter matrix element : 3
Enter matrix element : 1
Enter matrix element : 2
Enter matrix element : 3
Enter matrix element : 1
Enter matrix element : 2
Enter matrix element : 3
```

Entered Matrix is :

1	2	3
1	2	3
1	2	3

Example 8. *Program to read a matrix of size 3 × 3 and calculate sum of diagonal element it.*

Solution :

```
#include<stdio.h>
#include<conio.h>
main()
{
int  a[3][3],b[3][3],i,j,sum=0;
for(i=0;i<3;i++)
{
for(j=0;j<3;j++)
{
printf("\n Enter the matrix element :");
scanf("%d",&a[i][j]);
if(i==j)
sum=sum+a[i][j];
}
}
printf("\n Entered Matrix  is : \n");
for(i=0;i<3;i++)
{
for(j=0;j<3;j++)
printf("%6d",a[i][j]);
printf("\n");
}
printf("Sum of diagonal element's is : %d",sum)
}
```

Run:

```
Enter matrix element : 1
Enter matrix element : 2
Enter matrix element : 3
Enter matrix element : 1
Enter matrix element : 2
Enter matrix element : 3
Enter matrix element : 1
Enter matrix element : 2
Enter matrix element : 3
Entered Matrix is :
1       2       3
1       2       3
1       2       3
Sum of diagonal element's is : 6
```

Some Other Example Of Two Dimensional Array

Example 9. *Program to perform Matrix Addition.*

Solution :

```
#include<stdio.h>
#include<conio.h>
main()
{
int a[3][3],b[3][3],m,n,p,q;
static int i,j,k,c[3][3];
printf("Enter row and col for 1st Matrix :");
scanf("%d%d",&m,&n);
printf("Enter row and col for 2nd Matrix :");
scanf("%d%d",&p,&q);
if(m==p && n==q)
{
printf("Enter element of 1st Matrix :\n ");
for(i=0;i<m;i++)
{
for(j=0;j<n;j++)
{
printf("Enter the numbers:");
scanf("%d",&a[i][j]);
}
}
printf("Enter element of 2nd Matrix :\n ");
for(i=0;i<p;i++)
{
for(j=0;j<q;j++)
{
printf("Enter the numbers:");
scanf("%d",&b[i][j]);
}
}
for(i=0;i<m;i++)
for(j=0;j<q;j++)
{
c[i][j]=a[i][j]+b[i][j];
}
printf("1st Matrix is :\n ");
for(i=0;i<m;i++)
{
for(j=0;j<n;j++)
printf("%6d",a[i][j]);
printf("\n");
}
printf("2nd Matrix is :\n ");
for(i=0;i<p;i++)
{
for(j=0;j<q;j++)
printf("%6d",b[i][j]);
printf("\n");
}
printf("Result Matrix Is : \n");
```

```
for(i=0;i<m;i++)
{
for(j=0;j<q;j++)
printf("%6d",c[i][j]);
printf("\n");
}
}
else
printf("\n Entered Matrix isn't compatiable for Matrix Addition.");
getch();
}
```

Run: Enter row and col for 1st Matrix : 2 2
Enter row and col for 1st Matrix : 2 2
Enter the elements of 1st Matrix :
Enter numbers: 1
Enter numbers: 2
Enter numbers: 3
Enter numbers: 4
Enter the elements of 2nd Matrix :
Enter numbers: 1
Enter numbers: 2
Enter numbers: 1
Enter numbers: 2
Result matrix is:
2 4
4 6
Or
Enter row and col for 1st Matrix : 2 3
Enter row and col for 1st Matrix : 2 2
Entered Matrix isn't compatiable for Matrix Addition.

Example 10. *Program to perform Matrix Multiplication.*

Solution :

```
#include<stdio.h>
#include<conio.h>
main()
{
int a[3][3],b[3][3],m,n,p,q;
static int i,j,k,c[3][3];
printf("Enter row and col for 1st Matrix :");
scanf("%d%d",&m,&n);
printf("Enter row and col for 2nd Matrix :");
scanf("%d%d",&p,&q);
if(n==p)
{
```

```
printf("Enter element of 1st Matrix :\n ");
for(i=0;i<m;i++)
{
for(j=0;j<n;j++)
{
printf("Enter the numbers:");
scanf("%d",&a[i][j]);
}
}
printf("Enter element of 2nd Matrix :\n ");
for(i=0;i<p;i++)
{
for(j=0;j<q;j++)
{
printf("Enter the numbers:");
scanf("%d",&b[i][j]);
}
}
for(i=0;i<m;i++)
for(j=0;j<q;j++]
for(k=0;k<n;k++)
{
c[i][j]=c[i][j]+a[i][k]*b[k][j];
}
printf("1st Matrix is :\n ");
for(i=0;i<m;i++)
{
for(j=0;j<n;j++)
printf("%6d",a[i][j]);
printf("\n");
}
printf("2nd Matrix is :\n ");
for(i=0;i<p;i++)
{
for(j=0;j<q;j++)
printf("%6d",b[i][j]);
printf("\n");
}
printf("Result Matrix is : \n");
for(i=0;i<m;i++)
{
for(j=0;j<q;j++)
printf("%6d",c[i][j]);
printf("\n");
}
}
else
printf("\nEntered Matrix isn't compatiable for Matrix Multiplication.");
getch();
}
```

Run: Enter row and col for 1st Matrix : 2 2
Enter row and col for 1st Matrix : 2 2

Enter the elements of 1st Matrix :
Enter numbers: 1
Enter numbers: 2
Enter numbers: 3
Enter numbers: 4
Enter the elements of 2nd Matrix :
Enter numbers: 1
Enter numbers: 2
Enter numbers: 1
Enter numbers: 2
Result matrix is:
3 6
7 14
Or
Enter row and col for 1st Matrix : 2 3
Enter row and col for 1st Matrix : 2 2
Entered Matrix isn't compatiable for Matrix Multiplication.

Example 11. *To print lower triangular Matrix.*

Solution :

```
#include<stdio.h>
#include<conio.h>
main()
{
int a[3][3],m,n,p,q;
static int i,j,k;
printf("Enter row and column for  Matrix :");
scanf("%d%d",&m,&n);
printf("Enter element of Matrix :\n ");
for(i=0;i<m;i++)
{
for(j=0;j<n;j++)
{
printf("Enter the numbers:");
scanf("%d",&a[i][j]);
}
}
for(i=0;i<m;i++)
for(j=0;j<n;j++)
{
   if(j>i)
   a[i][j]=0;
}
printf("Matrix is :\n ");
for(i=0;i<m;i++)
{
```

```
    char a[5];
    int i;
    for(i=0; i<5; ++i) scanf("%c",&a[i]);
    for(i=4;i>=0;--i) printf("%c",a[i]);
}
```

Notice that the only difference is the declared type of the array and the **%c** used to specify that the data is to be interpreted as a character in **scanf** and **printf.** The trouble with character arrays is that to use them as if they were text strings you have to remember how many characters they hold. In other words, if you declare a character array 25 elements long and store **SUNIL** in it you need to remember that after element 4 the array is empty. For that C uses the simple convention that the end of a string of characters is marked by a **null** character. A **null** character is, as you might expect, the character with ASCII code 0. If you want to store the **null** character in a character variable you can use the notation **\0** - but most of the time you don't have to actually use the **null** character. The reason is that C will automatically add a **null** character and store each character in a separate element when you use a string constant.

A string constant is indicated by double quotes as opposed to a character constant, which is indicated by a single quote. *For example*:

"Akash"

is a string constant, but

'Akash'

is a character constant. The difference between these two superficially similar types of text is confusing at first and the source of many errors. All you have to remember is that "Akash" consists of six characters, **Akash** followed by **\0.** If you are familiar with other languages you might thought that you could assign string constants to character arrays and work as if a string was a built-in data type. In C however the fundamental data type is the array and strings are very much grafted on. For example, if you try something like:

```
char name[40];
name="Sunil"
```

It will not work. However, you can print strings using **printf** and read them into character arrays using **scanf.**

For example:

```
main()
{
   char name[25] ="Sunil";
   printf("%s",name);
   scanf("%s",name);
   printf("%s",name);
}
```

This program reads in the text that you type, terminating it with a null and stores it in the character array **name.** It then prints the character array treating it as a string, *i.e.*, stopping when it hits the first null string. Notice the use of the **"%s"** format descriptor in **scanf** and **printf** to specify that what is being printed is a string.

The problem with the scanf function is that it terminates its input on the first white space it finds. (A white space includes blanks, tabs, carriage returns, form feeds, and new lines.) Therefore, if the following line of text is typed

Example 14. *Program to copy one string into another.*

Solution :

```
#include<stdio.h>
#include<conio.h>
main()
{
   char a[30],b[30];
   int i;
   clrscr();
   printf("\n enter 1st string");
   gets(a);
   for(i=0;a[i]!='\0';i++)
   b[i]=a[i];
   b[i]='\0';
   printf("Copied string :");
   puts(b);
   getch();
}
```

Example 15. *Program to reverse a string and check for Palindrome.*

Solution :

```
#include<stdio.h>
#include<conio.h>
#include<string.h>
main()
{
   char a[15],b[15];
   int l,i,j;
   printf("Enter a string : ");
   gets(a);
   l=strlen(a);
   for(i=0,j=i-1;i<l/2;i++,j--)
   {
   if(a[i]!=a[j])
   {
   printf("Not  Palindrome");
   break;
   }
   }
   if(i==l/2)
   printf("Palindrome");
   getch();
}
```

Example 16. *To extract the portion of given string.*

Solution :

```
#include<stdio.h>
#include<conio.h>
#include<string.h>
```

```
main
{
char a[15],b[15];
int l,i,j,c,m,n;
printf("Enter a string : ");
gets(a);
printf("Enter Position and number of characters to be Extracted : ");
scanf("%d%d",&m,&n);
l=strlen(a);
if(m<l)
{
for(i=0,j=m;i<n;i++,j++)
{
b[i]=a[j];
}
b[i]='\0';
}puts(b);
getch();
}
```

Some Library Function For String Handling

Strlen(char *str) : Finds length of a string. The function returns a number.

Example: **strlen("Regional")**

The function returns 8.

strcat(char *destination,char *source) concatenates a copy of source to destination.

Example: strcat("Regional","College")

The function will concate string "College" with string "Regional" and hence the output will be "RegionalCollege".

strcmp(char *str1,char *str2) compares two strings and returns a number.

If a number is positive it means that the str1 is greater than str2 .

If a number is negative it means that the str1 is less than str2.

If a number is zero it means that the str1 and str2 are identical.

Example:

```
Int r;
...
...
r=strcmp("Regional","Poornima"); returns positive value to variable r.
r=strcmp("Regional","Regional"); returns zero to variable r.
r=strcmp("Poornima","Regional"); returns negative value to variable r.
strcpy(char *str1,char *str2) copies the contents of str2  to str1.
Strrev(char *str) reverse content of str.
```

Example:

```
Char a[]="AKANKSHA";
strrev(a);
puts(a);
```

The output will be "AHSKNAKA".

7 FUNCTION

MODULAR PROGRAMMING

Programs for real life applications are generally very large and complex. A team of programmers in such situation, to handle the complexity to the program and to allow teamwork, normally develops such programs; it becomes necessary to subdivide the programs into smaller and relatively independent subprograms these subprograms are then integrated into a single program for the application. This programming approach is referred to as modular programming. Modular programming is a term used to describe dividing a program into areas of code or modules that perform specific task. Following are the important.

Characteristics of Modular Programming

- ✔ The programmer designs the program in levels, where a level consists of one or more modules, the first level is complete main program and modules at successive levels consist of sub modules reference in the prior level.
- ✔ The main program controls the order in which the modules are executed by the computer. This describes in the solution to a program. The procedures being written in the order of machine execution.
- ✔ Modules should be structured within themselves by incorporating the construct of sequencing, selection and repetition. Every program can and should be written using only these constructs.

Advantages

Modular programming offers the following advantages over monolithic in line programming where the entire logic is packed together.

- ✔ By dividing the solution to a problem into many parts it is possible to obtain a better insight into the solution of the problem as a whole.
- ✔ It is often far easier and quicker to document specific tasks related to functions than document a complete unstructured programs.
- ✔ Programming maintenance is simpler for individual modules than for an entire program.

In order to be able to use a module in an application, the programmer should only know what that module does. He need not bother about how that task is performed.

Top-down Approach

In the top-down design the overall task of the program is to break down a large programs into several smaller logics and then they are interlaced by calling these logics whenever necessary.

The top-down approach starts from a high level abstract solution of the given problem and work down toward more detailed specific solution.

Top-down programming refers to style of programming where an application is constructed starting with a high level description of what it is supposed to do, and breaking the specification down into simpler pieces, until a level has been reached that corresponds to the primitives of the programming language to be used.

In this, the over all task is just defined in terms of generalized subtask, which are subsequently further defined and so on. This process is continued downward (↓) until the sub tasks are defined in a form suitable for execution by the computer.

The top down approach starts from a high level abstract solution of the given problem and works down towards more detailed specific solution. The high level abstract solution of the problem may involve some complex operations. Each of these complex operations may be viewed as a module and developed separately. This approach is illustrated below.

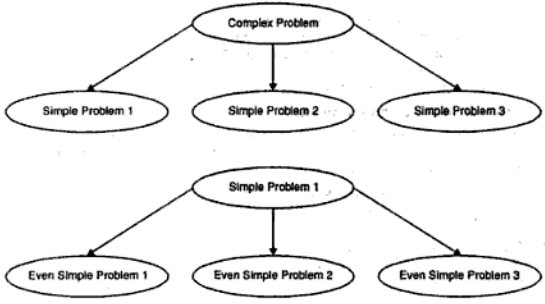


Figure 7.1.

In addition to program coding, the top down approach should be used in testing and debugging. Thus firstly, one main, module is written, it is tested and debugged. But the main module sends some information to the modules at the second level, which have not been written yet. Thus, dummy modules at the second level are needed. If errors are found in the main module, they are corrected immediately. Then the modules at the second level are written. If one or more modules at the second level send some information to a third level it will be necessary to develop dummy modules at the third level, and so on.

Advantages

- ✔ It shows the sequential division of the problem from top to bottom, which is a generalized and easy method.
- ✔ Modules can be developed parallel.
- ✔ Easy, quick and minimum error prone development programs.

What structures or patterns are allowed when using structured programming?

There are only there types (*a*) sequence (*b*) decision (*c*) loop.

Sequence Structures

In the sequence structure there must be a definite starting and ending point. After starting the sequence, programming statements are executed one after another until all the statements in the program continued to another sequence.

Decision Structure

The decision structure allows the computer to branch, depending upon certain condition normally; there are only two possible branches. As with all structure there is a starting point and an ending point for discussion structure. It is important to note that regardless of which branch is taken, the ending point is the same.

Loop Structure

The final structure is the loop. Actually there are two commonly used structures for loops.

One is the do until structure, and the other one is the do while structure. In the do structure the loop is done until a certain condition is met. For the do while structure, the loop is done while a certain exists.

FUNCTION

C functions are the equivalent of what in other languages would be called subroutines or procedures. Functions can be classified into two categories. These two categories are:

(*i*) Standard library function

(*ii*) User defined functions.

Standard Library Functions

Every C compiler is accompanied by a set of library of functions. In order to make C programs portable from one compiler to another and from one system to another the names of most of the functions in the library are internationally agreed upon. Library functions are further categorized and according to that stored in different files known as header files. In order to use library functions include specified header file and then call library function with specified return type and arguments.

Example 1. *Find the square root of any number.*

Solution :

```
#include<stdio.h>
#include<math.h>
main()
{
int n;
printf(" Enter the number ");
scanf("%d",&n);
printf(" The square root of %d  is %f ",n sqrt(n)  );
/*sqrt() library function /*
}
```

User Defined Functions

If there is no appropriate library function for your specific requirement, we can define our own function. We can even build our own library of functions that include

- ✔ Writing a function is also known as **defining the function.**
- ✔ The code that is placed inside the function is known as the **function definition.**
- ✔ Using the function is also known as **calling the function.**

In its most basic form, a function definition has the following form.

```
Function-name ()
{
   declarations;
   statements;
}
```

Once the function has been defined, it can be called as many times as we require. In its basic form, the syntax of a function call is as follows:

```
function-name();
```

What happens when a function is called?

Any function that is called by another function is known as a called function. A function that calls another function is known as calling function.

When a function is called, control is transferred from the calling function to the called function. Execution starts from the first executable statement in the called function. Once the called function finishes execution, control is handed back to the calling function. Execution resumes from the statement following the function call.

The user defined functions are of three types:

1. Function with no argument and no return value

Functions which have no argument and no return value are written as:

```
main()          → calling function
{
   func();      → function call
}
   void func() → called function (body of the function)
{
   statements
}
```

in the above example the function func( ) is called by the main( ) and the code of the function (body of the function) is written after main( ) function. As the function func( ) has no argument, main( ) can not send any data to func( ) and since it has no return statement, hence function cannot return any value to main( ).

```
/* find the value square of any number */
#include<stdio.h>
main()
{
fun();   /* user defined function with no argument */
}
fun()
{
int n,s;
```

```
printf("Enter the number ");
scanf("%d", &n)
s=s*n;
printf("The square on number % d is %d\n", n, s);
}
```

2. Function with parameter but no return value

Functions have parameters, hence the calling function can send data to the called function but it cannot return any value to calling function as it has no return statement. This function can be written as:

```
main ()
{
   func ( a, b); → actual parameter
}
   void func ( c, d )  → formal parameter
{
   statements ;
}
```

Here a and b are actual parameters which are used for sending the value c and d are the formal parameters, which take values from the actual parameters. It is necessary to declare the data type of the formal parameters before defining the function.

Example 2. *Find the sum of the two numbers.*

Solution :

```
#include<stdio.h>
main()
   {
   int  a,b;
   printf(" Enter the first number ");
   scanf(" %d" ,&a);
   printf(" Enter the second number ");
   scanf(" %d" ,&b);
   sum(a,b); /* actual parameter */
   }
   void sum(c, d) /* formal parameter */
   int c,d;
   {
   int s=0;
   s=c+d;
   printf(" The sum of these two numbers = %d", s);
   }
```

3. Function with argument and return value

Such function has parameters, the calling function can send data to the called function and it can also return any value to the calling function with the use of the return statement. This function can be written as

```
main()
{
func(a, b);
}
```

```
ret-type func(a, b)
int x,y ;
{
return(expression);
}
```

Here return statement returns the value of the expression to the calling function.

Example:

```
/* Program to find the sum of two numbers */
#include<stdio.h>
main()
{
   int a,b;
   print("Enter the first number : ");
   scanf(" %d" ,&a);
   print("Enter the second  number : ")
   scanf("%d",&b);
   printf(" The sum of two numbers =%d", ,sum(a,b));
}
int sum(c ,d)
int c,d;
{
   int s=0
   s=c+d;
   return(s); /* return statement */
}
```

Passing Information to a Function

C provides a mechanism through which we can communicate information to a function known as parameter passing mechanism.

A piece of information that is communicated to a function is called a parameter or an argument.

Function Prototype

If a function returns a value that is not of integer type, then it must be declared before it is called. The syntax for declaring a function is given below :

Ret-type function-name (arg2 – type , ..., argn-type) ;

Where ret-type is the return type of the function and arg2-type is the data type of the first argument and argument-type is the data type of the n^{th} argument.

The declaration of a function is also known as a function prototype. The definition of a function is also its declaration. The function prototype must agree with the definition and uses of the function. Function prototypes enable the compiler to detect errors in the number of argument and their types.

The parameter is in the function definition is known as a formal parameter. The parameter provided to function at the time of function call is known as actual parameter. A function must have same number of actual and formal parameters. The types of corresponding actual and formal parameters must also agree.

Example 3. *WAP to find out max of n numbers.*

Solution :

```
#include<stdio.h>
#include<conio.h>

main()
{
int a,p;
int man(int );
printf("Enter a number");
scanf("%d",&a);
p=man(a);
printf("%d",p);
getch();
}
int man(int n)
{
int max1=0,num,i;
for(i=0;i<n;i++)
{
printf("Enter number");
scanf("%d",&num);
if(num>max1)
{
max1=num;
}
}
return max1;
/* printf("%d",max1);*/
}
```

Example 4. *To find factorial of given number using function.*

Solution :

```
#include<stdio.h>
#include<conio.h>
longinf fact(int);
void main()
{
int n;
long int f ;
printf(" Enter a number");
scanf(" %d",&n);
f=fact(n);
printf(" Factorial is =%lf ",f);
getch();
}

long int fact(int x);
{
int i;
long int k=1;
```

```
scanf(" %d ", &n);
rev(n);
}
rev(k)
int k;

{
int rem ;
printf(" Now the number is  ");
while (k>0)
{
rem=k%10;
printf("% d",rem);
k/10;
}
printf("\n");
}
```

RECURSION

A recursive function is one, which calls itself. The process is used for repetitive computation in which each action is started in terms of previous result.

In order to solve a problem recursively two conditions must be satisfied :

(1) Problem must be written in recursive form.

(2) The problem statement must include a terminating condition.

In recursive functions, we must have an if statement somewhere to force the function to return without the recursive call being executed. Otherwise, the function will never return.

Recursive functions can be effectively used to solve problems where the solution is expressed in terms of successively applying the same solution to subsets of the problem. Recursive functions are useful in evaluating certain types of mathematical function. You may also encounter certain dynamic data structures such as linked lists or binary trees. Recursion is a very useful way of creating and accessing these structures.

Example 8. *To find factorial of number upto n using recursion.*

Solution :

```
main()
{
int n,a1;
printf("Enter number :");
scanf("%d",&n);
a1=fact(n);
printf("fact= %d",a1);
getch();
}
int fact(int a)
{
if(a==1)
return 1;
```

```
else
return a*fact(a-1);
}
```

Example 9. *Program to generate Fibonacci series using recursion.*

Solution :

```
#include<stdio.h>
long fibonacci(long)
void main()
{
long result number;
printf(" Enter an integer number ");
scanf("%d",&number);
result = Fibonacci (number);
printf(" Fibonacci (%d)=%d",number ,result);
getch();
}
long Fibonacci (long n)
{
if( n= =0 :: n= = 1)
return n;
else
return Fibonacci (n-1)+ Fibonacci (n-2);
}
```

Example 10. *Print the sum of odd digits of any number using recursion.*

Solution :

```
#include<stdio.h>
main()
{
int n,value;
printf(" Enter  the number :");
scanf(" %d",&n);
value=sum(n);
printf(" The sum of digit =%d\n",value);
}
sum(k)
int k;
{
int rem, s=0;
if(k>0)
{
rem=k%10;
s+=rem;
sum(k/10);
}
return(s);
}
```

Storage Class in 'C'

A variable in C can be characterized by two different ways:

(*i*) Data type

(*ii*) Storage class

C language supports four storage classes:

(*i*) Automatic

(*ii*) External

(*iii*) Static

(*iv*) Register

Storage classes provide following information about variable:

(*i*) Scope of variable

(*ii*) Lifetime of the variable

(*iii*) Default initial value of variable

(*iv*) Actual location of variable

Scope or Visibility

Any variable that has been defined inside a function is "visible" only inside body of the function. This means that the variable can be accessed only inside the body of the function.

Variables declared inside a function body are also known as local or automatic variables. The visibility of an automatic variable is the function in which it is defined. A formal parameter of a function is also treated as a local variable.

C provides another kind of variable called a global variable. These variables are defined outside any function. A global variable is visible from the point of definition to the end of the file being compiled. A global variable is accessible to all the functions that are defined after the global variable.

Lifetime

The time for which a data object remains in existence during the execution of a program is called its lifetime.

Declaring Variables of Specified Storage Classes

A variable is declared by writing keyword followed by variable name:

Storage Class	Keyword
Automatic	auto
External	extern
Static	static
Register	register

Automatic (auto)

These variables have scope within the function block in which it is defined. Due to this property they are also known as local and internal variables. Variable is automatically created and destroyed when the function is called and exited. Hence, it is called automatic. If it is initialized then default value is set to that value otherwise some garbage value.

```
main()
{
```

```
auto int  n=25;
{
   {
      auto int  n=10;
      printf("\n Value of auto variable j in inner block = %d",n);
   }
printf("\n Value of auto variable j in outer block  %d\n",n);
}
}
```

Output: Value of auto variable j in inner block = 25
Value of auto variable j in outer block = 10

External (Extern)

These variables have scope within the entire program. They are also known as global variables. They are declared outside the function and default value is zero. They are stored in memory. They live till the end of the program. Any function can use these variables and change their value.

Static

These variables can have global and local scope according to place of declaration. Their default value is zero. Storage allocated to these variables is allocated in RAM.

```
main()
{
static int a[10] ,k;
clrscr();
for(k=0;k<10;k++)
printf("a[%d] = %d\n",k, a[k]);
getch();
}
```

Run: A[0]=0
A[1]=0
A[2]=0
A[3]=0
A[4]=0
A[5]=0
A[6]=0
A[7]=0
A[8]=0
A[9]=0

Static variables retain their values even after a function call is over.

Example:

```
main()
{
int x;
int f1();
x=f1();
printf("%d",x);
x=f1();
printf("%d",x);
}
```

```
int f1()
{
static s;
s=s+10;
return s;
}
```

Run: X=10

X=20 

Why the value x is 20 not 10 because in the function f1() then s is a static variable and hence, it will retain its old value even after the function call is over.

Register

These variables have scope within the function block in which it is defined. Their default value is garbage. They are stored in CPU registers, so they provide fast access. If machine registers are not available in that case the register variables are automatically assumed as auto variables.

Advantage of Storage Classes

By using suitable storage class for a variable, the programmer can economize memory requirements by restricting the lifetime of the variable. It will also help to maintain a modular, well structured program by allowing access to variables only to those files and functions need it.

Local and Global Variable

The variables which are defined within a body of the function or block is local to that function or block only and called local variable.

Example:

```
Var(a,b)
int a,b;
{
int c,d;
---------
}
```

Here a and b are the local variables which are defined within the body of the function var( ). Local variable can be used for only that function, in which they are defined. The same variable name may be used in different functions and these variables are local to that function only.

Example:

```
Var1(x, y)
int a,b;
{
int c,d;
c=5;
d=10;
---------
  --------
}

Var 2( m,n)
```

```
int m,n;
{
int a,b;
a=15;
b=20;
--------
--- ----
}
```

Here value of a = 5 , b = 10 is local function var1 and a = 15, b = 20 is local to the function var2( ).

Global Variable

The variables which are defined outside the main( ) function is called global variable. The global variable has the same and name throughout the program. It is useful to declare the variable global when the variable has constant values throughout the program.

Example:

```
int a, b=6;
main()
{
a=8;
-------
-------
func();
}
func()
{
int mult;
mult=a * b;
--------
--------
}
```

Here a and b are the global variable.

□□□

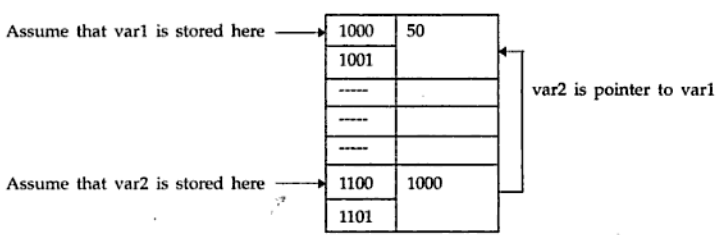
8 POINTER IN C

The computer access its own memory not by using variable names but by using a memory map with each location of memory uniquely defined by a number, called the address of that memory location. For example, assume that float variable ft have been defined. Also assume that float is stored in four bytes. If ft is stored in the bytes with addresses 201, 202, 203 and 204, then the address of ft is 201. The address of variable ft will be 201 because 201 is least significant byte.

What is Pointer?

A pointer is a variable that stores the location of memory. In more fundamental terms, a pointer stores the address of a variable. In more picturesque terms, a pointer points to a variable.

C allows us to store the address of one variable in another variable. For example, assume that var1 and var2 are two variables. var1 is an integer variable. Let's assume that the variable var1 is stored at locations 1000 and 1001 and the variables var2 is stored in location 1100 and 1101. The value of var1 is 50 and the value of var2 is 1000. This is illustrated below.



When variables var1 and var2 are such that the value of var2 is the address of var1, then var2 is said to be pointing to var1. **Var2 is called a pointer.**

C allows us to define objects that are capable of storing the address of other objects.

Defining a Pointer

The general syntax for defining a pointer is:

Data_type *ptr_name

Where data_type specifies the data type of the object that the pointer ptr_name will point to.

Example:

int *p; → defines p as a pointer to an integer. Adding an asterisk in front of a variable's name declares it to be a pointer to the declared type. Notice that the asterisk applies only to the single variable name that it is in front of, so:

int *p, q;

long *pl; → defines p1 as a pointer to an long.

char *pch; → defines pld as a pointer to an character.

float *pf; → defines pf as a pointer to an character.

So, p is capable of storing the address of any integer variable, pl is capable of storing the address of any float variable and pch is capable of storing the address of any character variable, and so on.

In C, it is necessary to specify the data type of object that a pointer will point to. It is not possible to have a pointer that can hold the address of an integer at one time and then the address of a float at some other point of time.

void *pv;

A pointer to void can hold the address of any variable.

Once you have declared a pointer variable you can begin using it like any other variable, but in practice you also need to know the meaning of two new operators: & and *. The & operator returns the address of a variable. You can remember this easily because & is the ampersand character and it gets you the address. For example:

int *p, q;

declares p, a pointer to int, and q an int and the instruction:

p=&q; → stores the address of q in p and now p is pointing to q.

Example:

```
int i, j;
int *x;
float f=75.22;
i=30;
j=59;
......
......
x=&i;    /* x is now pointing to i  */
......
......
x=&j;    /* x is now pointing to j */

......
......
x=&f     /*This is invalid because a pointer to int can't
           hold the address of a float variable*/
```

The Dereferencing/Indirection Operator

The second operator * is a little more difficult to understand. If you place * in front of a pointer variable then the result is the value stored in the variable pointed at. That is, p stores the address, or pointer, to another variable and *p is the value stored in the variable that p points at.

The * operator is called the de-referencing operator and it helps not to confuse it with multiplication or with its use in declaring a pointer.

This multiple use of an operator is called operator overload.

Confused? Well most C programmers are confused when they first meet pointers. There seems to be just too much to take in on first acquaintance. However there are only three basic ideas:

1. To declare a pointer add an * in front of its name.
2. To obtain the address of a variable use * in front of its name.
3. To obtain the value of a variable use * in front of a pointer's name.

```
int * m,n,p;
n=10;
m=&n;
p=*m;
```

→ p will have value 10, because * before m will give the value stored in the variable pointed at m.

Example:

```
#include<stdio.h>
#include<conio.h>
main()
{
   int *ptr;
   int a=100,b;
   ptr=&a;
   b=*ptr   /* here in front of ptr ,* means dereferencing */
   printf("a=%5db=%5d\n",a,  b);
   *ptr=*ptr+10; /* here the value of the variable to whom ptr is
                                                    pointing is modified */
   printf("a=%5db=%5d\n",a,b);
}
```

Run: a=100 b=100
a=110 b=100

Passing Pointers as Parameters

Why passing pointers as function parameter is required?

```
Void interchange(int a,int b)
{
int temp;
temp=a;
a=b;
b=temp;
}
```

However, if you try this function you will find that it doesn't work. You can use it—but it just will not change the values stored in a and b back in the calling program, because in interchange function that all parameters in C are passed by value. That is, when you use interchange (a,b) function the values in a and b are passed into the function interchange via the parameters and any changes that are made to the parameters do not alter a and b back in the main program.

The solution to this very common problem is to pass not the values stored in the variables, but the addresses of the variables. The function can then use pointers to get at the values in the variables in the main program and modify them.

So function should be like that:

```
void interchange(int *a,int *b)
{
int temp;
temp=*a;
*a=*b;
*b=temp;
}
```

The following uses interchange to exchange the values of two variables:

```
#include<stdio.h>
void interch(int *, int *);
main()
{
int a, b;
printf(" Enter two numbers ");
scanf(" %d ", &a,&b);
printf(" Before interchange : a %d\n b=%d\n",a,b);
swap(&a, &b ); /* Notice the & */
printf(" After interchange :a=%d\n b=%d\n", a,b);
}
```

Arrays and Pointers

In C there is a very close connection between pointers and arrays. Recall that the value of the name of an array is the address of its 0th element. An array name is a constant pointer.

Suppose a is a ten element array of integers. The statement

```
int a[10] ;
```

Creates an array of 10 consecutive integer variables. Moreover the name a is now a constant pointer to the 0th element of the array. When you write an expression such as a[i] this is converted into a pointer expression that gives the value of the appropriate element. To be more precise, a[i] is exactly equivalent to *(a + i) *i.e.*, the value pointed at by a + i . In the same way *(a+ 1) is the same as a[1] and so on.

This can be proved the following example:

```
#include<stdio.h>
main()
{
int a[]={10,20,30,40,50};
int *ptr,i;
ptr=a; /* The content of a is assigned to ptr and it is the address
                                                          of a[0] */
for(i=1;i<=5;i++)
{
printf(" %d\t%d\n",*ptr,a[i]);
ptr++;
}
}
```

Output:

10 10
20 20
30 30
40 40
50 50

Operation on Pointer/Address Arithmetic

C provides several arithmetic operations that can be applied on pointers the following are a list of such arithmetic operations. In all cases, it is assumed that a pointer p is pointing to an array.

- It is possible to increment a pointer a p. The Expression p++ sets p to point to the next element in the array.

  Example:

```
.......
int a[]={10,20,30,40};
int *p;
p=a;
printf("%d\n" ,*ptr);
p++;
printf("%d",*ptr);
.......
```

  Run:

  10
  20

- It is possible to add an integer to a pointer. The expression p+ I, refers to the address of the element that is I elements beyond the element is currently pointing to. The expression p+ =I, increments p to point I elements beyond where it currently does.

  Example:

```
.......
int a[]={10,20,30,40,50,60,70,80};
int *p;
p=a;
printf("%d\n" ,*ptr);
p=p+2;
printf("%d",*ptr);
........
```

  Run:

  10
  30

- An integer may be subtracted from pointer .The expression p – j refers to the address of the j^{th} object prior to the one p is currently pointing to. This is true regardless of the kind of object p points to.

Example:

```
.......
int a[]={10,20,30,40,50,60,70,80};
int *p;
p=a;
printf("%d" ,*ptr);
p=p+4;
printf("%d",*ptr);
p=p-1;
printf("%d",*ptr);
........
```

Run:

10 50 40

- Two pointers may be compared provided both are pointing to members of the same array. Suppose p and q point to members of the same array, then their values can be compared. All relational operators (<, >, <=, >=, = = and !=) may be used. The pointer pointing to an element with a higher subscript value will always be greater than a pointer to an element with a lower subscript value.
- A pointer can also be compared with 0. This is useful when testing whether a pointer is pointing to a valid object or not. It is guaranteed that no object will have an address of 0. A pointer with a value of 0 is also referred to as a null pointer. The symbol NULL is also defined in stdio.h to the value 0.
- It is possible to subtract two pointers. However the operation is only when both pointers point to members of the same array. Suppose, p and q point to elements of the same array, and p < q, then q–p indicates the number of elements between location p and q (inclusive of the element that q is pointing to).
- The operation of addition, multiplication or division of two pointers is not defined.
- A pointer can be initialized like any other variable. The values assigned are generally zero or an expression containing the address of a predefined data item of appropriate type.

Dynamic Memory Allocation

All the programs that we have seen so far "created" objects at compile time. In the case of local objects, the compiler took care of automatically creating and destroying these objects at the appropriate times.

Sometimes it is desirable to create objects at run time. That is we do not inform the compiler that an object of a specified type and name needs to be created. Instead, we create an object of the appropriate type ourselves at run time. This is done by allocating memory for the object that we need to create. Since the memory is allocated for objects " dynamically" (*i.e.*, at run time),creation of such objects is known as dynamic memory allocation.

The C standard library provides the function malloc( ) to create objects dynamically. The function takes one arguments—the size in bytes of the objects to be created. It either returns the pointer to the creates object if it was able to allocate memory or return NULL if not.

The following example illustrates the use of malloc ( ).

```
int * create int()
{
```

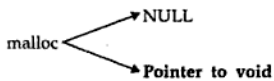
```
int *p;
p=(int *) malloc(10);
*p=0;
return p;
}
```

There are several things to be noted in the above function.

The function create_int( ) has been declared as returning a pointer to int.

The malloc function has been called with a single argument –10. The malloc function allocates ten bytes of memory and returns a pointer to this block. Malloc( ) does not know the type of information that will be stored in this block of memory (recall what we said earlier, the contents of the memory are subject to various interpretations). Since malloc ( ) does not know the data type of the object, it cannot return a pointer of a specific type such as int or float etc. It therefore returns a pointer to void. A pointer to void can point to any object.

We then convert the pointer to void to a pointer to int through type casting and save the value returned by malloc in the pointer p. The pointer p is now pointing to the newly created object.



Malloc returns **NULL,** if malloc failed to allocate memory otherwise returns **pointer to void,** if successful.

Function related to dynamic allocation memory:

(*i*) malloc (allocates memory and return its address)

(*ii*) calloc (allocates memory and returns its address)

(*iii*) free (deallocates memory and puts it into free pool)

(*iv*) realloc (reallocates memory when the allocated memory is found to be not sufficient)

Example 1. *Program to read N numbers in an array and sort it into ascending order.*

Solution:

```
#include<stdio.h>
#include<conio.h>
#include<process.h>
main()
{
   int *a,n,i,j;
   int *arr(int);
   printf("Enter size of array : "); scanf("%d",&n);
   a=arr(n);
   for(i=0;i<n;i++) {printf("Enter value : "); scanf("%d",&a[i]);}
   for(i=0;i<n;i++)
   {
      for(j=i+1;j<n;j++)
      if(a[i]>a[j])
      {  temp=a[i],a[i]=a[j],a[j]=temp;   }

   }
   for(i=0;i<n;i++)
```

```
    printf("%5d",a[i]);
}
/*function definition of int *arr(int)*/
int  *arr(int size)
{
    int *p;
    p=(int *) malloc(sizeof(int) * size);
    if (p==NULL)
    {
    printf("malloc failed to allocate memory \n");
    exit(1);
    }
    return p;
}
```

***This above program can create an array of any size and will make our program more flexible.**

Pointer to Pointer

A pointer to a pointer variable whose value is the address of another pointer. The pointer to pointer is defined as follows :

```
datatype ** variable_name;
int ** z;
```

The two asterisks indicate that z is a pointer to a pointer.

Example:

```
Main()
{
int a,*ptr,**ptr_t_ptr;
a=500;
ptr=&a;
ptr_t_ptr=&ptr;  /* address of ptr is assigned to ptr_t_ptr */
printf("%d\n",**ptr_t_ptr); /* hrer ** in front of ptr_t_ptr will
                                        give 500 for output */
}
```

Pointer to Function

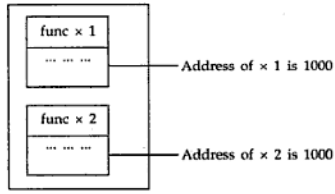
```
data_type (*pf) ([argument])
void (*pf)(int);
```

In the above declaration the pf is pointer to a function. A pointer to a function is a variable whose value is the address of a function. In the above statement , pf is declared as pointer to function returning void and accepting a int value. The parenthesis surrounding *pf is required because the * operator has lower precedence than the function call operator( ).

```
void *pf(int);
```

In the above statement the *pf is not surrounded by parenthesis , in this case the pf is not considered as pointer to function but as a function name , which returns int value and accepting no arguments.

In 'C' Language a function is smallest unit of code whose address can be taken.



As we know that every function in the computer memory is stored on different addresses and hence it is possible to take their addresses and kept in pointer to function.

Example:

```
#include<stdio.h>
main()
{
void(*ptrf)(); /* ptrf is pointer to function */
void func1();
void func2();
void func3();
ptrf=func1;    /* here the address of func1 is assigned to ptrf */
(*ptrf)();     /* function func1 is called due dereferencing */
ptrf=func2;
(*ptrf)();
ptrf=func3;
(*ptrf)();
}

void func1()
{
   printf("Hello, I am Sunil Chauhan \n");
}
void func2()
{
   printf("You can contact me for your problems\n");
   printf("A-27, Vidyut Nagar, Ajmer Road, Jaipur\n");
}
void func3()
{
   printf("My Phone Number is 94149-76044\n");
   printf("or you can send mail to sunil865@rediffmail\n");
}
```

Run:

Hello, I am Sunil Chauhan.
You can contact me for your problems.
A-27, Vidyut Nagar, Ajmer Road, Jaipur
My Phone Number is 94149-76044.
or you can send mail to sunil865@rediffmail.

Advantages of Pointer

- ✔ Pointer is also closely associated with arrays and therefore provide an alternate way to access individual array elements.
- ✔ Pointers are often passed to a function as argument. We refer to this use of pointer as passing argument by reference or by address or by location.
- ✔ A convention array definition results in a hred block of memory being reserved at the banging of program execution whereas this does not occur if the array is represented in terms of a pointer variable. Therefore the use of a pointer variable to represent a array require some type of initial memory assignment before the array elements are passed this is known as dynamic memory management allocation.
- ✔ We can pass functions to other functions using pointer.
- ✔ A pointer enables us to access a variable that is defined outside the function.
- ✔ Pointers are more efficient in handling the data tables.
- ✔ Pointers reduce the execution speed of program.

Disadvantages of Pointer

- ✔ Usage of pointers makes a program complex.
- ✔ Using pointer in program takes more space because only one data item can be used through two variables one is reference and by it's original name.

□□□

9 STRUCTURE AND UNION

STRUCTURE

A structure is a collection of variables under a single name. These variables can be of different types, and each has a name, which is used to select it from the structure. A structure is a convenient way of grouping several pieces of related information together.

Defining a Structure

To define a structure, we use the keyword structure. The collection of data items that needed to be parts of the structure are then declared within braces.

```
struct [structure_tag]
{
data_type_1 var1,var2….;
data_type_2 var3,var4….;
......
......
} [struct_var,....];
```

Example:

```
struct student
{
   char name[20];
   float marks;
};
```

A structure declaration starts with the key word **struct** and a list of data items that need to be part of the structure. Each data item in structure is called a member of the structure.

The structure definition presented above creates a new structure data type called student. In order to use the structure, we first need to define variables of this type. This is similar to defining variables of type int. While int is a built in data type, student is a type that we have created. Such types are also referred to as user defined types.

i.e., struct student engg_student;

This creates a variable called engg_student, which is of the structure type student.

Accessing Structure Element

The members of a structure are processed individually for structure manipulation. A structure member can be accessed using period operator (.) with structure variable name.

```
printf("Age : %d\n ",s.age);
printf("Address : %s\n ",s.address);
printf("City : %s\n ",s.city);
getch();
}
```

Array of Structure

struct student s[10];

As we have declared array of int or array of float, in the same way we can declare the array of structure. The bytes required by array will **10 * the size of all the members** in the structure.

```
for(i=0;i<10;i++)
{
   printf("Name ");
   puts(s[I].name);
   printf("Marks");
   scanf("%d",&s[i].marks);
}
```

Example 2. *To print roll number, name and father name of the student who secured highest total mark.*

Solution :

```
#include<stdio.h>
struct student
{
   int rollno;
   char name[20];
   char fname[20];
   int tmarks;
}
main()
{
   struct student[];
   int i,k,j,s,max;
   for(i=0;i<5;i++)
   {
   scanf("%d",&&,[i]rollno);
   gets(s,[i].name);
   gets(s,[i].fname);
   scanf("%d",&&,[i].tmarks);
   }
   max=0;
   for(i=0;i<5;i++)
   {
   if(s,[i].tmarks>max)
   max=s,[i].tmarks;
   }
   j=i;
```

```
    for(k=j;k<=j;k++)
    {
    printf("%d",s,[k]rollno);
    printf(s,[k].name);
    printf("%s",s,[k].fname);
    printf("%d",s,[k].tmarks);
    }
}
```

Pointer To Structure

```
struct student
   {
   char name[20];
   int marks;
   };
struct student stu;
struct student ptrstu; /* ptrstu is pointer to structure */
......
......
ptrstu=&stu;           /* ptrstu is pointing to stu */
strcpy(ptrstu->name,"Sunil  Chauhan"); /* using pointer variable if one
                                          want to assign values to variable
                                          then instead of a . operator you
                                          have to use -> operator.*/

ptrstu->marks=900;
```

Nested Structure

```
Struct date
{
   int  dd;
   int mm;
   int yy;
};

Struct  student
{
   int stud_id;
   char name[30];
   struct
date dob;
   struct  date doj;
   char sex;
};
```

The above structure definition is an example of a nested structure definition.

Self-Referential Structure

It is also possible to define a structure one of whose members is a pointer to an object of the structure type itself.

```
int month;
int year;
};
void print_date(struct date);
main()
{
struct date d = {15,9,2004};
print_date(d);
}

void print_date(struct date d1)
{
printf(" \n Date (dd/mm/yy) : %d/%d/%d \n", d1.day, d1.month,d1.year);
}
```

3. Passing address of structure

```
Struct item
{
int item_id;
char item_name[10];
float item_price;
int item_Quantity;
};

main()
{
struct item i1;
get_item(&i1);
printf("\nItem id = %d",i1.item_id);
printf("\nItem name = %s",i1.item_name);
printf("\nItem price = %f",i1.item_price);
printf("\nItem quantity = %d",i1.item_quantity);
getch();
}

void get(struct item *I)
{
printf("\n Enter Item id :");
scanf("%d",&i1->item_id);
printf("\n Enter Item name : ");
gets(i1->item_name);
printf("\n Enter Item price: ");
scanf(" %f",&i1->item_price);
printf("\n Enter Item quantity : ");
scanf(" %d",&i1->item_quantity);
}
```

Unions

Unions are concept borrowed from structures and therefore follow the same syntax as structures. In structures, each member has its own storage location, whereas all the members of a union use the same location. This implies that, although a union may contain many members of different types, it can handle only one member at a time.

10 FILE HANDLING IN C

INTRODUCTION

A file is a named collection of data. This data is usually stored in a secondary storage device like a hard disk or a floppy disk.

The name that can be given to a file is usually operating system dependent. Some operating systems allow names of unlimited length and any number of extensions while some other have limitations on the length of file names. The space that is occupied by a file to store data is normally managed by the operating system's underlying file system.

Most programming languages provide an interface through which it is possible to manipulate information that is stored in files. C provides this interface through its standard library of functions. A very important concept in C is the **stream.**

In C, the **stream** is a common, logical interface to the various devices that comprise the computer. In its most common form, a **stream** is a logical interface to a **file.** As C defines the term "file", it can refer to a disk file, the screen, the keyboard, a port, a file on tape and so on. Although files differ in form and capabilities, all **streams** are the same. The **stream** provides a consistent interface and to the programmer one hardware device will look much like another.

A **stream** is linked to a file using an **open operation.** A **stream** is disassociated from a file using a **close operation.** The current location, also referred to as the current position, is the location in a file where the next file access will occur. There are two types of **stream: text** (used with ASCII characters some character translation takes place, may not be one-to-one correspondence between stream and what's in the file) and **binary** (used with any type of data, no character translation, one-to-one between stream and file).

Difference between Text and Binary File

Text File	Binary File
1. In a text file information is stored using the machine's code for representing characters *e.g.,* ASCII or EBCDIC.	1. A binary file stores data in binary form.
2. It can be viewed and manipulated through an operating system utility *i.e.,* Editor.	2. It can be viewed and manipulated through a program.
3. It is a variable record length files.	3. It is a fixed record length files.

Read data from file

Once a file has been opened, depending upon its mode, you may read and/or write bytes to or from it using these two functions.

```
int fgetc(FILE *fp);
int fputc(int ch, FILE *fp);
```

The **fgetc()** function reads the next byte from the file and returns it as an integer and if error occurs returns **EOF**. The **fgetc()** function also returns **EOF** when the end of file is reached. Your routine can assign **fget()**'s return value to a **char** you don't have to assign it to an integer.

The **fput()** function writes the bytes contained in ch to the file associated with fp as an unsigned **char**. Although ch is defined as an **int**, you may call it using simply a **char**. The **fput()** function returns the character written if successful or **EOF** if an error occurs.

Text File Functions

When working with **text files**, C provides four functions which make file operations easier. The first two are called **fputs()** and **fgets()**, which write or read a string from a file, respectively. Their prototypes are :

```
int fputs(char *str, FILE *fp);
char *fgets(char *str, int num, FILE *fp);
```

The **fputs()** function writes the string pointed to by str to the file associated with fp. It returns **EOF** if an error occurs and a non-negative value if successful. The null that terminates str is not written and it does not automatically append a carriage return/linefeed sequence.

The **fget()** function reads characters from the file associated with fp into a string pointed to by str until num-1 characters have been read, a new line character is encountered, or the end of the file is reached. The string is null-terminated and the new line character is retained. The function returns str if successful and a **null pointer** if an error occurs.

The other two file handling functions to be covered are **fprintf()** and **fscanf()**. These functions operate exactly like **printf()** and **scanf()** except that they work with files. Their prototypes are :

```
int fprintf(FILE *fp, char *control-string, list of variables);
int fscanf(FILE *fp, char *control-string, list of variables);
```

Instead of directing their input/output operations to the console, these functions operate on the file specified by fp. Otherwise their operations are the same as their console-based relatives. The advantages to **fprintf()** and **fscanf()** is that they make it very easy to write a wide variety of data to a file using a text format.

FILE INPUT/OUTPUT

To work with files, the library routines must be included into your programs. This is done by the statement,

```
#include<stdio.h>
```

as the first statement of your program.

Using Files

- **Declare a variable of type file**

  To use files in C programs, you must declare a file variable to use. This variable must be of type **FILE**, and be declared as a pointer type.

  File is a predefined type. You declare a variable of this type as

  ```
  FILE *in_file;
  ```

  this declares infile to be a pointer to a file.

- **Associate the variable with a file using fopen()**

  Before using the variable, it is associated with a specific file by using the fopen() function, which accepts the pathname for the file and the access mode (like reading or writing).

  ```
  in_file = fopen("myfile.dat", "r");
  ```

  In this example, the file **myfile.dat** in the current directory is opened for **read** access.

- **Process the data in the file**

  Use the appropriate file routines to process the data.

- **When finished processing the file, close it**

  Use the fclose() function to close the file.

  ```
  fclose(in_file);
  ```

The following illustrates the fopen function, and adds testing to see if the file was opened successfully.

```
#include <stdio.h>
#include<process.h>
main()
{
   FILE *input_file;
   Input_file = fopen("Regional.dat", "w");
   if( input_file == NULL)
   {
   printf("** Regional.dat could not be opened.\n");
   printf("returning to dos.\n");
   exit(1);
   }
}
```

Example 1. *Program to write records in a data file(Mark04 . dat).*

Solution :

```
#include<stdio.h>
#include<process.h>
#include<conio.h>
main()
{
      FILE *fp;
      int rollno, marks;
   fp=fopen("Marks04.dat","w");
   if(fp==NULL)
```

```
    {
       printf("\n**Marks04.dat couldn't opened\n");
    getch();
    exit(1)'
    }
    while(1)
    }
    printf("Rollno:");
    scanf("%d",&rollno)
    if(rollno==-999)
    {
    break;
    }
    print("Marks");
    scanf("%d",&marks);
    fprint(fp,"%d%d",rollno,marks);
    }
fclose(fp);
getch();
}
```

Example 2. *Program to read records in a data file(Mark04 . dat).*

Solution :

```
#include<stdio.h>
#include<process.h>
#include<conio.h>
main()
{
      FILE *fp;
      int rollno, marks, r;
   fp=fopen("Marks04.dat","w");
   if(fp==NULL)
   {
      printf("\n**Marks04.dat couldn't opened\n");
   getch();
   exit(1)'
   }
   printf("Roll number\tMarks");
   printf("---------------");
   r=fscanf(fp,"%d%d:,&rollno,&marks);
   while(r)
   {
   printf("%d\t%s\n",rollno,marks);
   r=fscanf(fp,"%d%d",&rollno,&marks);
   }

   fclose(fp)
}
```

Example 3. *Program to read from file(marks04 . dat) and write records in file(first.dat,second.dat, failed.dat) as per the percentage.*

Solution :

```
#include<stdio.h>
#include<process.h>
#include<conio.h>
main()
{
   FILE *fp, *first, *second, *failed;
   int rollno, marks, per;
   fp=fopen("Marks04.dat","w");
   first=fopen("first.dat","w");
second=fopen("second.dat","w");
failed=fopen("failed,dat","w");
   if(fp==NULL || first==NULL || second==NULL || failed==Null)
   {
      printf("\n**Failed to open files\n");
   getch();
   exit(1);
   }
   r=fscanf(fp,"%d%d",&rollno,&marks);
   while(r)
   {
      per=(marks/500.0)*100;
      if(per>=60)
         fprint("first."%d%d",rollno,marks);
      else if(per>=48)
         fprint("second"%d%d",rollno,marks);
      else if(per<36)
         fprint("failed."%d%d",rollno,marks);

      r=fscanf(fp,"%d%d",&rollno,&marks);
   }
   fclose(fp);
   getch();
}
```

Character Input and Output with Files

This is done using equivalents of getchar and putchar which are called getc and putc. Each takes an extra argument, which identifies the file pointer to be used for input or output.

EOF, The End of File Marker

EOF is a character which indicates the end of a file. It is returned by read commands of the getc and scanf families when they try to read beyond the end of a file.

Example 4. *Program to read name of students in a file (Deepshika.dat) and copy the data from deepshika.dat to the file (Regional.dat).*

Solution :

```
#include<stdio.h>
#include<coinio.h>
```

```
for(n=1;n<=100;n++)
{
if(n%2==0)
   fprintf(fp1,"%d",n);
else
   fprintf(fp2,"%d",n);
}
fclose(fp1);
fclose(fp2);
}
getch();
}
```

□□□

11 PREPROCESSOR

INTRODUCTION

Preprocessor is a program that processes our source program before it is passed to the compiler, our program passed through several processors before it is ready to execute.

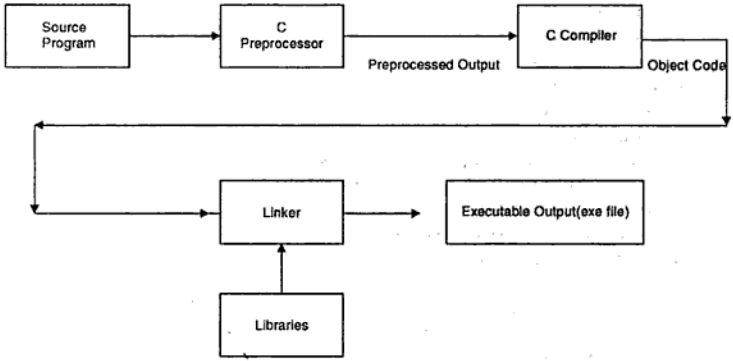


Figure 11.1.

The preprocessor offers several features called preprocessor directives. Each of these preprocessor directive begin with a # symbol. The directives often placed at the beginning of a program. Some preprocessor directives are:

(*a*) #Define directive

(*b*) #Include directive

(*c*) #If directive

(*d*) #Pragma

(*e*) Conditional compilation directive

When the C source program is submitted for compilation, the first phase of compilation is taken over by the preprocessor. The preprocessor goes through the C program to perform an operation known as "preprocessing". It then generates an intermediate output known as the

preprocessed output. The preprocessor output is submitted to the compiler as input and then the compiler translates and creates object file.

Functions of a C Preprocessor

The C preprocessor performs various useful functions that are handily when developing large application. These functions are listed below.

Inclusion of Named Files

The preprocessor provides the facility of including the contents of another file in the source program. That is currently being complied. This is done through the #include directive. A file can be included within double quote(" ") or angular brackets(< >)

i.e.,

```
#include"user.c"
```

The content of user c will be included with source program before compilation. In this way user can create own header files.

```
#include<stdio.h>
#include<alloc.h>
#include<process.h>
#include<string.h>
#include<ctype.h>
```

Macro Substitution

The macro substitution refers to the text replacement facility that is offered by the C preprocessor. The macro substitution facility is provided by the #define preprocessor directive.

The syntax of the #define directive is given below:

```
# define name          replacement Text
```

All subsequent occurrences of *name* is replaced by the replacement text.

Example :

```
#define MAX   10

for(k=0;k<MAX;k++)
{
printf("%d\n" ,k);
}
```

Example :

```
#define square (x)   x*x
....
R=square(5);  /* the square( 5) will be substituted with the replacement
                                                              text*/
              /*at compile time, hence R= 5*5 */
.......
```

Conditional Compilation

The C preprocessor provides certain directives that control which part of the code are seen by the compiler and which aren't, this process of selectively compiling parts of code is known as conditional compilation.

□□□

So, $(0100101.00011)_2$.

(iii) $(101)_8 = (?)_{10}$

$(101) = 1 * 82 + 0 * 81 + 1 * 80 = 64 + 0 + 1 = (65)_{10}$.

(iv) $(110)_{16} = (?)_{10}$

$(110) = 1 * 162 + 1 * 161 + 0 * 160$

$= 256 + 16 + 0 = 272$

$(110)_{16} = (272)_{10}$.

(v) $(111)_{16} = (?)_8$

		1	1	1
4 bit position–	0001	0001	0001	0001
3 bit position–	000	100	010	001
		0	4	2	1

$(111)_{16} = (421)_8$.

(vi) $(1111)_2 = (?)_{16}$

binary number 1111

Grouping the above binary number into 4-bits , we have hexadecimal equivalent

1111

+

15

$(1111)_2 = (f)_{16}$.

(vii) $(111)_8 = (?)_{16}$

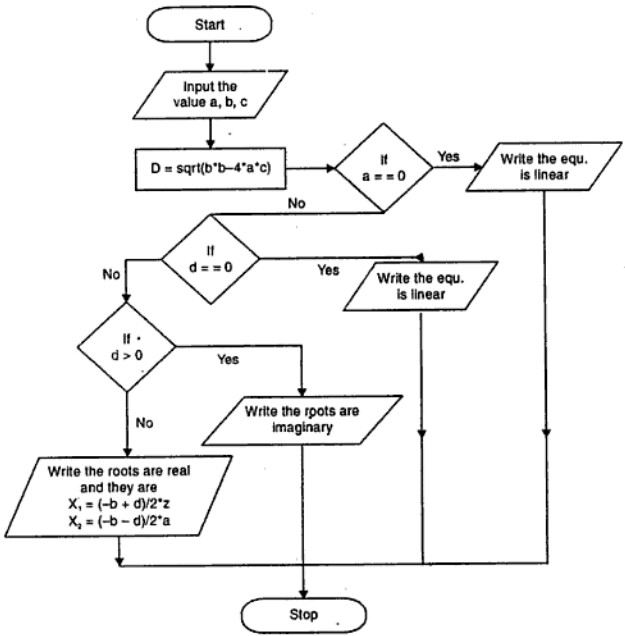
octal number–	1	1	1
	+		
3 bit binary	001	001	001
4 bit position	0000	0100	1001
	0	4	9

$(111)_8 = (49)_{16}$.

Q. 4. Draw the flow chart and write a C program to find out roots of a given quadratic equation.

$$Ax^2 + bx + c = 0$$

Solution:



Program:

```
#include<stdio.h>
#include<conio.h>
void main()
{
//* programme to find root of any quadratic equation ax²+bx+c =0
  int D,a,b,c,x1,x2
  printf("Enter the equation's variable value");
  printf("Enter a");
  scanf("%d", &a);
  printf("Enter b");
  scanf("%d", & b);
  printf("Enter c");
  scanf("%d", &c);
//*Calculate value of D
```

```
D=sqrt(b*b)-4*a*c;
if(a==0)
printf("Equation is linear.\n");
else
{
if(d==0)
{
x1=x2=-b/2*a;
printf("Roots are real and equal and they are %d, &%d", x1, x2);
}
else if(d>0)
{
x1=(-b+D)/2a;
x2=(-b-D)/2a;
printf(" The roots are real  and they are %d, &%d", x1, x2);
}
else
printf("Roots are imaginary);
}
getch();
}
```

Q. 5. Write a C program to count occurrences of a given word in a given text.

Solution:

```
#include<stdio.h>
#include<conio.h>
main()
{
int I,n =0;
char a[100];
clrscr();
printf("/n Enter the text:");
gets(a);
printf("/n NO. of word:");
for(I=0; a[I] !='\0';I++)
{
if(a[I]= =' ')
n++;
}
printf("%d", n+1);
getch();}
```

Q. 6. Write a C program to sort the given array of elements in increasing order.

Solution:

```
#include<stdio.h>
#include<conio.h>
main()
{
int i,j,temp;
static int a[10];
for(i=0;i<10;i++)
{
printf(" Enter %d array element : " ,i+1);
scanf("%d",&a[i]);
```

```
}
for(i=0;i<9;i++)
for(j=9;j>=i+1;j--)
if(a[j]<a[j-1])
{
temp=a[j];
a[j]=a[j-1];
a[j-1]=temp;
}
printf("Sorted array in increasing order :\n");
for(i=0;i<10;i++)
printf("%d\n",a[i]);
getch();
}
```

Q. 7. (a) Write C program for the following inbuilt functions:

(*i*) Strcat

(*ii*) Strcpy

Solution: (*a*)

```
#include<stdio.h>
#include<conio.h>
#include<string.h>
main()
{
  int a[101],I,j,temp,n;
  char str1[10],str2[10],str3[20];
  clrscr();
  printf("\n Enter the string 1:");
  scanf("%s",&str 1);
  printf("\n Enter the string 2:");
  scanf("%s",&str2);
  strcat(str 1, str2);
  printf("\n concatenated string is:%s\n",str1);
  strcpy(str3, str1)
  getch();
  printf("\n copied string is: %", str3);
  getch();
```

(b) What will be the output of the following program segment?

```
main()
{
  static int b[]={10, 20, 30, 40, 50};
  int I, * k;
  k=& b[4] -4;
  for(I=0;I<=4; I++)
  {
    printf(" %d ", * k);
    k++;
  }
}
```

Solution: (*b*) output 10 20 30 40 50.

Q. 8. Write short notes on any two of the following :

(*i*) Operating systems

(*ii*) Computer languages

(*iii*) Compiler and interpreter

(*iv*) Data types of C language.

Solution : (*i*) See in Chapter 9 in Part A.

(*ii*) See in Chapter 6 in Part A.

(*iii*) Given below in next Q.

(*iv*) See in Chapter 2 in Part B.

Q. 9. Differentiate the following (any three):

(*i*) Process and processor.

(*ii*) Compiler and interpreter.

(*iii*) System program and application program.

(*iv*) Time sharing operating system and real time operating systems.

Solution: (*i*) Process and processor

Process	Processor
Process is a program in execution, process is more than the programme code. It is represented by the value of the programme counter and the content of the processor of the register. Its process stores temporary data and data section containing global variable.	A unit of a computer system which intemperate the instruction and execute them processor is the unit which control all the process, all operations perform by it.

(*ii*) Compiler and interpreter

Compiler	Interpreter
1. Simple and easy to write.	1. Complex and difficult to write.
2. Does not require much memory space to store itself.	2. Requires large memory space to store it self.
3. Translates the program line by line into machine language at once.	3. Translates the entire program into machine language at once.
4. Checks for the syntax errors line by line and execute each line (statement) immediately when statement becomes errors free.	4. Checks the syntax errors in the program and when the whole program becomes errors free then it is translated into machine code and execute.
5. Takes less time in translation.	5. Takes more time in translation.
6. Takes more time to execute the program.	6. Takes less time to execute the program.

7. Fast for debugging (detection and correction of syntax and logical errors.)	7. Slow for debugging.
8. The object code produced by an interpreter is not saved for future reference.	8. The object code produced by the compiler is permanently saved for future reference.

(*iii*) System program and application program

Application Programs	System Program
1. These are general purpose programs used for governing of computer.	1. These are programs designed for certain specific usages.
2. They are designed to optimize the functioning and efficiency of a computer.	2. Their use varies from financial usage to non-financial and scientific operations.
3. They can operate independently.	3. They need system software for their smooth functioning.
4. They are very complex programs and usually provided by the manu-facture.	4. They are relatively easier programs and can be developed in house.
5. Computer can not work without system programs.	5. Computer does not need an application program of its operation.

(*iv*) Time sharing operating system and real time operating system

Time Share Operating System	Real Time Operating System
1. Time-sharing operating system is used to maximize utilization of CPU time.	1. On line processing systems with severe time limitations are called real time operating system.
2. Time delay is short.	2. Time delay is negligible.
3. Degree of sophistication is less.	3. Highly sophisticated and very expensive systems.
4. Used for varied problems. Provides resource sharing for small users.	4. Used in applications that require an immediate response from the computer.

Q. 1. (a) **Convert the following:**

(*i*) $(101011)_2 = (?)_8$

(*ii*) $(703.5)_{10} = (?)_{16}$

(*iii*) $(89AE)_{16} = (?)_{10}$.

Solution: (*a*) (*i*) 101 011 (Group of 3 bits)

5 3 (Decimal equivalent)

Result: $(53)_8$.

(*ii*) For integer part – 703

$703/16 = 43$ + with a remainder of 15

$703/16 = 43$ + with a remainder of 11

$43/16 = 2$ + with a remainder of 11

$= (2BF)_{16}$

for decimal part

$(0.5)_{10} = (?)_{16}$

$(0.5)_{10} = (?)$

$0.5 * 16 = 8.0$

$(0.5)_{10} = (0.8)_{16}$

thus $(703.5)_{10} = (2BF.8)_{16}$.

(*iii*) Hexadecimal number 89AE

$$(89AE)_{16} = 8 * 16^3 + 9 * 16^2 + A * 16^1 + E * 16$$
$$= 8 * 16^3 + 9 * 16^2 + 10 * 16^1 + 14 * 16^0$$
$$= 8 * 4096 + 9 * 256 + 10 * 16 + 14 * 1$$
$$= (35246)_{10}.$$

(b) **Perform the following:**

(*i*) **101001 – 010111**

(*ii*) $(736)_8 - (677)_8$

(*iii*) $(101)_2 + (73)_8$

(*iv*) $(110111)_2 + (AF)_{16}$.

Solution: (*b*) (*i*) $-101001 = 41$

$010111 = -23$

$010010 = 18.$

(*ii*) $738 = (480)_{10}$

$677 = (447)_{10}$

$41 = 33.$

(*iii*) $(101)_2 - (?)_8$

$(101)_2 = (5)_{10}$

$(73)_8 = (59)_{10}$

$5 + 59 = (64)_{10} = (100)_8$

(*iv*) $(110111)_2 + (AF)_{16}$

First we make same base

$(110111)_2 = (?)_{16}$		$(110111)_2 = (55)_{10}$
0011	0111	$(AF)_{16} = (175)_{10}$
2	7	total $(230)_{10} = (E6)_{10}$

$(37)_{16} + (AF)_{16}$

$(37)_{16} + (AF)_{16} = (E6)_{16}$

Q. 2. State the difference between:

(*i*) Assembler and Compiler

(*ii*) ASCII and EBCDIC

(*iii*) Fixed point number and floating point number

(*iv*) Impact printer and Non-Impact printer

(*v*) ROM and PROM

(*vi*) Sequential access and Random access device

(*vii*) High Level Language and Machine Level Language.

Solution : (*i*) See in Chapter 6 in Part A.

(*ii*) See in Chapter 4 in Part A.

(*iii*) See in Chapter 4 in Part A.

(*iv*)

Basis	Impact Printer	Non-impact Printer
Type of energy used	Force	Electrical, chemical or thermal energy
No. of copies at a time	1 or more	1 only
Use of ribbon	Yes	No
Use of ink toner	No	Yes
Type of paper	Any type	Specific type of paper is preferred
Quality of print	Average to good	Good to excellent

(*v*) See in Chapter 8 in Part A.

(*vi*) See in Chapter 8 in Part A.

(*vii*) See in Chapter 6 in Part A.

Q. 3. Convert the following :

(*i*) $(363)_{10} = (?)_2$

(*ii*) $(ABCD)_{16} = (?)_8$

(*iii*) $(2F59)_{16} = (?)_{10}$

(*iv*) $(1011.1101)_2 = (?)_{16}$

Solution:

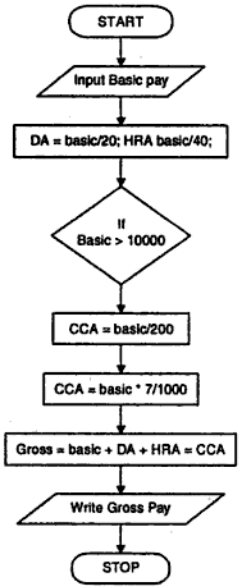


Figure 1.

```
#include<stdio.h>
#include<conio.h>
main()
{
float basic, DA=0, HRA=0, CCA=0, gross=0;
clrscr();
printf("\n Enter the Basic Pay : ");
scanf("%d",&basic);
DA=basic/20;
HRA=basic/40;
if(basic>10000)
CCA=basic/200;
Else
CCA=basic*7/1000;
gross=basic+DA+HRA+CCA;
printf("\n The Gross pay is : %2f Rs. ",gross);
}
```

```
int I,x;
char name[25],grade[5];
clrscr();
f1=fopen(sname.doc,"w");
f2=fopen(sgrade.doc,"w");
printf("Enter number of student");
scanf("%d",&n)
for(I=0;I<n;I++)
{
printf("Enter student %d name : ",I+1);
fscanf(stdin,"%s",name);
fprintf(f1,"%s\n",name);
printf("\t Enter Grade");
fscanf(stdin,"%s",grade);
fprintf(f2,"%s\n",grade);
}
fclose(f1);
fclose(f2);
f1=fopen(sname.doc,"r");
f2=fopen(sgrade.doc,"r");
f3=fopen(sdat.doc,"w");

for(I=0;I<n;I++)
{
fscanf(f1,"%s\n",name);
printf("Name : %s      ",name);
fprintf(f3,"  Name = %s",name);
fscanf(f2,"%s\n",grade);
printf("Grade = %s",grade);
fprintf(f3,"Grade = %s\n",grade);
printf("\n");
}
fclose(f1)
fclose(f2);
fclose(f3);
getch();
}
```

Q. 9. Add marks of 3 different subjects in three arrays , output division of the student in 4th array.

Division	Condition
1	Above 70
2	Above 70
3	Above 70
4	Above 70
5	Remaining

Solution:

```
#include<stdio.h>
#include<conio.h>
void main()
{
int mark[3],I,sum=o,per;
clrscr();
printf("Enter mark in Hindi");
```

```
scanf("%d", &mark[0]);
printf("Enter mark in Mathematics ");
scanf("%d", &mark[1]);
printf("Enter mark in English");
scanf("%d",&mark [2]);
for(I=0;I<=2;I++)
sum=sum+mark[I];
per=sum/3;
printf("sum=%d",sum);
if(per>=70)
printf("first div");
if(per>=60& &per<70)
printf("second div");
if(Per>=50 && per<60)
printf("third div");
if(per>=33&&per<50)
printf("fourth div");
if(per<33)
printf("fifth div");
getch();
}
```

Q. 10. Draw the flowchart and write a C program to find out largest number in given list of 10 numbers.

Solution:

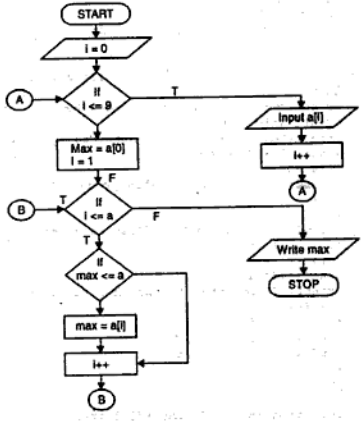


Figure 2.

Program:

```
void main()
{
int a[10], I=0,max;
clrscr();
printf("Enter 10 numbers : \n");
for(I=0;I<=9;I++)
scanf("%d",&a[I]);
b=a[0];
for(I=0;I<=9;I++)
{
if(ma::<a[I])
{
max=a[I];
}
}
printf("Largest number is : %d ",max);
getch();
}
```

Q. 11. Write a program to find the product of two matrices, this function should accept two matrices as parameters and return product.

Solution:

```
//product of two matrices
#include<stdio.h>
#include<conio.h>
void prod(int[ ] [ ],int[ ][ ],int);
void main ();
{
int a, b ,c ,d ,x[10][10],y[10][10],z[10[10],I,j,k,n =179;
clrscr();
  printf("Enter the first matrices type (1 .line 2 .row )");
  scanf("%d%d",&a,&b);
  printf("Enter the another matrices type (1 .line 2 .row)");
  scanf("%d%d",&c,&d);
  if(a!=d)
  printf("This is not a valid product");
  else
  {
  printf("Value enter for first matrices ");
  for(I=1;I,=a;I++)
  for(j=1;j<=b;j++)
  scanf("%d",&x(i)(j);
  printf("Value enter for another matrices ");
  for(I=1;j<=c;I==)
  for(j=1;j<=d; j==)
  scanf("%d",&y(i)(j);
  prod(x(10(10),y(10),a);
  }
  getch()
  }
  void prod(int x[][],int y[][],inta)
  {
  for(I=1;I<a;I++)
```

```
{
for(j=1;j<=a;j++)
{
z[I][j]=0;
for(k=1;k<=a;k++)
{
z[I][j]=z[I][j]+x[I][k]*y[k][j];
}
}
}
for(I=1;I<=a;I++)
{
printf("%c",n);
for(j=1;j<=a;j++)
{
printf("%3d",z[I][j]);
}
printf("%c\n",n);
}
}
```

Q. 12. Write a C program to read an integer from a file 'integer.dat' and write all the prime numbers in another file 'prime.dat'.

Solution:

```
#include<stdio.h>
#include<conio.h>
#include<iostream.h>
void main()
{
   FILE *f1, *f2;
int n,I,num,j,n1;
clrscr();
f1=fopen("integer.doc","w");
printf("Enter number of integers : ");
scanf("%d",&n);
printf("\n Enter integers : ");
for(I=0;I<n;I++)
{
fscanf(stdin,"%d",&num);
fprintf(f1,"%d",num);
}
fclose(f1);

f1=fopen("integer.doc","r");
f2=fopen("prime.doc","w");
for(I=0;I<n;I++)
{
fscanf(f1,"%d",num);
for(j=2;j<num;j++)
{
  if(num%j==0)
  break;
```

```
}
if(j==num)
fprintf(f2,"%d",num);
}
getch();
}
```

Q. 13. Write a C program to define a structure "employee" with the following members

-emp-name

-emp-city

-emp-phone number

Accept the data for 10 such structure from the user and then print emp-name, emp-city and emp-phone number of the employee whose name is "Ram".

Solution:

```
#include<stdio.h>
#include<conio.h>
struct employee
{
  char emp_name[50];
  char emp_city[20];
  long int emp_phone_number;
}e[10];
main()
{
struct e;
int I,n;
char name[50];
clrscr();
printf("How many employee:");
scanf("%d",&n);
for(I=0;I<n;I++)
{
printf("\N enter the name of employee %d :",I+1);
scanf("%s",&e[I].emp_name);
printf("\N enter the city & telephone no. of employee %d :",I+1);
scanf("%s %ld",&e[I].emp_city,&e[I].emp_phone_number);
}
printf("\N enter employee name whose data to be shown:");
scanf("%s",name);
for(I=0;I<n;I++)
{
if(strcmp(e[I].emp_name)==0)
break;
}
it(I==n)
printf("Data not found");
else
```

```
{
printf("\n employee name :%s",e[I].emp_name );
printf("\n \nCity : %s",e[I].emp_city);
printf("\n\nTelephone number : ld",e[I].emp_phone_number);
}
getch();
}
```

Q. 14. Differentiate between the structure and union of C.

Solution: Every variable of structure has own address on memory that exist anywhere in memory, but variable of union address in a sequential way.

Q. 1. (a) Convert the following:

(*i*) $(735)_{10} = (?)_8 = (?)_2$

(*ii*) $(69.5)_{10} = (?)_{16} = (?)_8$.

Solution: (*i*)

Division	**Quotient**	**Remainder**
735/8	91	7 ↑
91/8	11	3 |
11/8	1	3 |
1/8	0	1 |

$(735)_{10} = (1337)_8$

In octal	1	3	3	7
In binary	001	011	011	111

So, $(735)_{10} = (1337)_8 = (001011011111)_2$.

(*ii*) $(69.5)_{10} = (?)_{16}$

for integer part.

Division	**Quotient**	**Remainder**	**for decimal part**
69/16	4	5 ↑	$= 0.5 \times 16 = 8.0$
4/16	0	4 |	

$(69.5)_{10} = (45.8)_{16}$

$(69.5)_{10} = (?)_8$

Division	**Quotient**	**Remainder**	**for decimal part**
69/8	8	5 ↑	$= 0.5 \times 8 = 4.0$
8/8	1	0 |	
1/8	0	1 |	

$(69.5)_{10} = (105.4)_8$.

(b) Perform the following:

(*i*) 101101101 + 011111011

(*ii*) $(73A)_{16} + (256)_{10}$

(*iii*) $(273)_8 + (111000111)_2$

(*iv*) $(87A)_{16} + (7DC)_{16}$.

Solution: (*i*)

$$\begin{array}{r} 101101101 \\ +\ 011111011 \\ \hline 1001101000 \end{array}$$

(*ii*) First make the base same

$$(73A)_{16} = (?)_{10}$$
$$= 7 \times 16^2 + 3 \times 16^1 + 10 \times 16^0$$
$$= 1792 + 48 + 10$$
$$= (1850)_{10}$$

$$\begin{array}{r} 1850 \\ +\ 259 \\ \hline 2109 \end{array}$$

$$= (2109)_{10}$$

(*iii*) $(273)_8 + (111000111)_2$

$(111000111)_2 = (707)_8$

$(707)_8 + (273)_8 = (1202)_8$

(*iv*) $(87A)_{16} + (7DC)_{16}$

$$\begin{array}{r} 87A \\ 7DC \\ \hline 1056 \end{array} \qquad \begin{array}{r} (100001111010)_2 \\ +\ (011111011100)_2 \\ \hline (1000001010110)_2 \end{array}$$

Q. 2. State the difference between:

(*i*) Laser printer Vs. Daisy chain printer;

(*ii*) Optical Mouse Vs. Mechanical Mouse;

(*iii*) Multi-user Operating System Vs. Batch operating system;

(*iv*) Volatile Memory Vs. Nonvolatile Memory

Solution: (*i*) See in Chapter 7 in Part A.

(*ii*) See in Chapter 7 in Part A.

(*iii*) See in Chapter 9 in Part A.

(*iv*) See in Chapter 8 in Part A.

Q. 3. (a) What do you understand with precedence of operators? How is it considered in expression evaluation?

(b) Write an algorithm to calculate distance between two points (X_1, Y_1) and (X_2, Y_2). What library file is required; if it is coded in 'C' language?

Solution: (*a*) See in Chapter 2 in Part B.

(*b*) caldiff(x_1, x_2, y_1, y_2, d)

Step1 Set $a = x_2 - x_1$ and $b = y_2 - y_1$

Step2 Set d = sqrt(a * a – b * b)

Step3 Exit

Or

```
void main()
{
int x1,x2,y1,y2,a,b,diff;
```

```
scanf("%c:,&ch);
switch(ch)
{
case 'a':
big = ch;
break;
case 'b':
case 'c':
big = ch;
break;
default:
big='z'
}
```

Solution: Enter option: a

then big = a

if b or c

then big = b or c

if ch = other character then big = z.

Q. 6. (a) Write a program to concatenate two strings using pointers. Do not use function.

Solution:

```
//concatenation of two strings
#include<stdio.h>
#include<conio.h>
void main()
{
   char str1[10],str2[10],*x,*y;
      int j;
   clrscr();
      printf("Enter a string :-");
      scanf("%s",str1);
      x=str1;
      printf("Enter a another string :-");
      scanf("%s",str2);
      y=str2;
   while(*x!='\0')
   {
   x++;
   }
   while(*y!='\0')
   {
   x=y;
   y++;
   x++;
   }
   *x='\0';
   printf("Concatenated string is %s",str1);
getch();
}
```

(b) What is structured programming?

Solution: See in Chapter 9 in Part B.

Q. 7. Write a program which will read a line and count all occurrences of 'college' in that line. Draw a flowchart for this problem.

Solution:

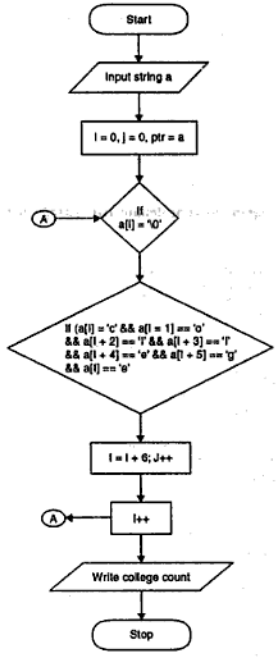


Figure 1.

Q. 9. Write an algorithm to calculate distance between two points (x_1, y_1) and (x_2, y_2).

Solution:

```
//difference between two point's co-ordinate
#include<stdio.h>
#include<conio.h>
void main()
{
int  x1,x2,y1,y2,a,b,c,d,diff;
clrscr();
  printf("Enter the 1st point's coordinate");
  scanf("%d%d",&x1,&y1);
  printf("Enter the 2nd point's coordinate");
  scanf("%d%d",&x2,&y2);
  a=x2-x1;
  b=y2-y1;
  c=a*a;
  d=b*b;
  diff=sqrt(c+d);//sqrt is a function to square root
  printf("Difference between the point is %d",diff);
  getch();
}
```

About the Book

The objective of this book is to make you a good programmer. We assume that you have not done any elementary course in C Programming. This book will help you in understanding how programs are developed and make you confident enough for writing your own small programs and help you in scoring good marks in examinations. In this book a number of useful programs are included in each chapter to demonstrate not only the syntax but the way in which the programs should be developed.

This book has been written in two parts. Part A is designed to fulfill the need of the subject **"Computer Fundamentals"** and Part B is designed to fulfill the need of the subject **"C Programming"** for the first year students of all branches of engineering of the University of Rajasthan. Here, the emphasis is on case of understanding of essential concepts.

Authors' Profile

Mr. Sunil Chauhan is currently the Head of Computer Engineering department in Regional College for Education, Research and Technology College, Jaipur. He holds the degree of M.Tech(CS) and MBA from Rajasthan University. He has long experience of teaching postgraduate, graduate, 'B' Level and Engineering students. He is teaching 'C' Programming Language for the last fifteen years and his areas of interest include Software Engineering, System Analysis and Design, Management Information System and Operating Systems. Mr. Chauhan has also developed various real life applications and worked in the software industry.

Ms. Kratika Gupta is currently working as Lecturer, Regional College for Education, Research and Technology College, Jaipur. She holds the degree of B.E.(CE) from Rajasthan University. She is teaching for the last three years. She is currently pursuing her M.Tech(CE).

Mr. Akash Saxena has done M.Sc(CS) & M.Tech(CS) in Computer Science from Udaipur University. He has been teaching for the last five years and has developed various real life projects and worked in the software industry.

Premier12

ISBN 81-7008-854-2

LAXMI PUBLICATIONS (P) LTD