P.S.R. ENGINEERING COLLEGE
SIVAKASI-626 140
(An Autonomous Institution, Affiliated to Anna University, Chennai)
Accredited by NBA and NAAC (A+ Grade)

COURSE MATERIALS

COURSE NAME:Programming for problem solving

COURSE CODE:191CSF1

FACULTY NAME: SAKTHILAKSHMI PRIYA.C,.AP/CSE

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

191CSF1	PROGRAMMING FOR PROBLEM SOLVING			L	T	P	C
				3	0	0	3
Programme:	**B.E./EEE/MECH/CIVIL/BIO-TECH/AIDS**	**Sem:**	**1**	**Category:**			**ESC**

Aim:	To provide an awareness to Computing and Programming.
Course Outcomes: The Students will be able to	
CO1:	Understand the basic terminologies of computer and various problem solving techniques.
CO2:	Recall the fundamentals needed for constructing a C program.
CO3:	Outline the various types of arrays and functions for string handling.
CO4:	Summarize the essence of functions and pointers in a C program.
CO5:	Illustrate the requirements of structures and unions.
CO6:	Make use of different functions for manipulating files.

INTRODUCTION	**9**
Generation and Classification of Computers- Basic Organization of a Computer - Number System - Binary - Decimal - Conversion - Problems. Software - Types, Development Steps. Algorithm - Pseudo code - Flow Chart. Problem formulation - Problem Solving.	
C PROGRAMMING BASICS	**9**
Introduction to Unix Operating System - Introduction to ‘ C’ programming - fundamentals - structure of a ‘C’ program - compilation and linking processes - Constants, Variables - Data Types - Expressions using operators in ‘C’ - Managing Input and Output operations - Decision Making and Branching - Looping statements - solving simple scientific and statistical problems.	
ARRAYS AND STRINGS	**9**
Arrays - Initialization - Declaration - One dimensional and Two dimensional arrays. String- String operations - String Arrays. Simple programs –Bubble Sort – Linear Search -Matrix Operations.	
FUNCTIONS AND POINTERS	**9**
Function - Definition of function - Declaration of function - Pass by value - Pass by reference - Recursion - Pointers - Definition - Initialization - Pointers arithmetic - Pointers and arrays- Example Problems.	
STRUCTURES AND FILES	**9**
Introduction - need for structure data type - structure definition - Structure declaration - Structure within a structure - Union - Programs using structures and Unions - File Manipulation - Storage classes - Pre-processor directives.	
Total Hours	**45**

Text Books:

1. Anita Goel and Ajay Mittal, “Computer Fundamentals and Programming in C”, Dorling Kindersley (India) Pvt. Ltd., Pearson Education in South Asia, 2017.
2. Balagurusamy E, “Programming in ANSI C”, Tata McGraw-Hill Education, 2016
3. ReemaThareja, “Computer Fundamentals and Programming in C”, 2e, Oxford University Press, 2016.

References:

1. Byron S Gottfried, “Programming with C”, Schaum’s Outlines, 3rdEdition, McGraw-Hill, 2017.
2. Dromey R.G., “How to Solve it by Computer”, Pearson Education, 4th Reprint, 2007.
3. KernighanB.W and Ritchie,D.M, “The C Programming language”, 2nd Edition, Pearson Education, 2006.

191CSF1: PROGRAMMING FOR PROBLEM SOLVING

UniT1:

INTRODUCTION

Generation and Classification of Computers- Basic Organization of a Computer – Number System - Binary - Decimal - Conversion - Problems. Software - Types, Development Steps. Algorithm - Pseudo code - Flow Chart. Problem formulation - Problem Solving.

Characteristics of Computers

The various characteristics of computers are as follows

1. **Speed**: Speed is the most important characteristics of computer .Computer having **more speed to perform jobs** instantaneously.
2. **Accuracy**: The computers are **perfect, accurate and precise**. Accuracy signifies the reliability of the hardware components of computers.
3. **Automatic**: A computer **works automatically, once programs are stored** and data are given to it, constant supervision is not required.
4. **Endurance**: A computer **works continuously** and **will not get tired** and will not suffer from lack of concentration.
5. **Versatility:** A computer can **be put to work in various fields.**
6. **Reduction of cost**: Though initial investment may be high, computer substantially reduces **the cost of transaction.**

Evolution of Computers

The Early development:

1.ABACUS:

ABACUS uses movable beads stung on wires above and below a cross bar and its operationare based on the idea of the place value notation. The beads of the counter represent digits.The value of the digit in each position is determined by adding the values of the beadspressed against the cross piece.

2.Pascal calculating machine:

It was the first real desktop calculating device that could add and subtract. It consists of a setof toothed wheels or gears with each wheel or gear having digits 0 to 9 engraved on it.Arithmetic operation could be performed by tunings these wheels.

3.Punched card Machine:

The presence and absence of the holes in the card represent the digits.Charles Babbage's Engine:
The machine took input from the punched card. The Analytical engine had a memory whichwill perform arithmetic operations.

Computer Generations

Generation in computer terminology is a change in technology a computer is/was beingused. Initially, the generation term was used **to distinguish between varying hardware technologies.**But nowadays, generation includes both hardware and software, which together make up an entire computer system.There are **totally five computer generations** known till date

Following are the main five *generations* of computers

Si.nO	Generation & Description
1	**First Generation** **The period of first generation : 1946-1959. Vaccum tube based.**
2	**2 Second Generation** **The period of second generation : 1959-1965. Transistor based.**
3	**Third Generation** **The period of third generation : 1965-1971. Integrated Circuit based.**
4	**Fourth Generation** **The period of fourth generation : 1971-1980. VLSI microprocessor based.**
5	**Fifth Generation** **The period of fifth generation : 1980-onwards.ULSI microprocessor based**

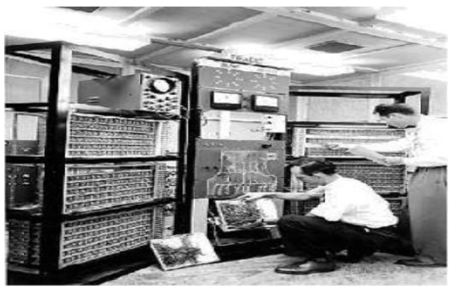
First Generation

i. The period of first generation was **1946-1959**.

ii. First generation of computer started with **using vacuum tubes** as the basic components for memory and circuitry for CPU (Central Processing Unit). These tubes like electric bulbs produced **a lot of heat** and were prone to frequent fusing of the installations, therefore, were **very expensive** and could be afforded only **by very large organisations.**

iii. In this generation mainly batch processing operating system were used. In this generation **Punched cards, Paper tape, Magnetic tape Input & Output device were used.**

iv. There were **Machine code** and **electric wired board languages** used.



The main features of First Generation are:
Vacuum tube technology
Unreliable
Supported Machine language only
Very costly
Generate lot of heat
Slow Input/Output device
Huge size
Need of A.C.
Non portable
Consumed lot of electricity

Some computer of this generation were:
ENIAC
EDVAC
UNIVAC
IBM-701
IBM-650

Second Generation

i. The period of second generation was **1959-1965.**
ii. This generation using the **transistor were** cheaper**, consumed less power**, **more compact in size, more reliable** and **faster** than the first generation machines made of vaccum tubes. In this generation, magnetic cores were
iii. used as primary memory and magnetic tape and **magnetic disks as** secondary storage devices.
iv. In this generation **assembly language** and **high level programming language like FORTRAN, COBO**L were used.
v. There were Batch processing and Multiprogramming Operating system used.

Fig: Second Generation
The **main features of Second Generation** are:

- Use of transistors
- Reliable
- Smaller size
- less heat
- less electricity
- Faster
- Still very costly
- Still very costly

- A.C. needed
- Support machine and assembly languages

Some computer of this generation were:

- IBM 1620
- IBM 7094
- CDC 1604
- CDC 3600
- UNIVAC 1108

Third Generation

i. The period of third generation **was 1965-1971.**

ii. The third generation of computer is marked by the use of **Integrated Circuits (IC's) in place of transistors.**

iii. **Single I.C has many transistors**, resistors and capacitors along with the associated circuitry. The I.C was invented by Jack Kilby . This development made **computers smaller in size, reliable** and **efficient.**

iv. In this generation Remote processing, Time-sharing, Real-time, Multi-programming Operating System were used.High level language (FORTRAN-II TO IV, COBOL, PASCAL PL/1,BASIC, ALGOL-68 etc.) were used during this generation.

Fig: Third Generation

The **main features of Third** Generation are:

- ✓ IC used
- ✓ More reliable
- ✓ More reliable
- ✓ Generate less heat
- ✓ Faster
- ✓ Lesser maintenance
- ✓ Still costly
- ✓ A.C needed
- ✓ lesser electricity
- ✓ high level language

- computer of this generation were:

- IBM-360 series
- Honeywell-6000 series
- PDP(Personal Data Processor
- TDC-316
- IBM-370/168

Fourth Generation

i. The period of Fourth Generation was **1971-1980.**

ii. The fourth generation of computers is marked by the use of **Very Large Scale Integrated (VLSI)**circuits**. VLSI circuits having about 5000 transistors and** other circuit elements and their Associated circuits on **a single chip** made it possible to have microcomputers of fourth generation. Fourth Generation computers became **more powerful, compact, reliable, and affordable**. As a result, it gave rise to personal computer (PC) revolution.

iii. In this generation Time sharing, Real time, Networks, Distributed Operating System were used.

iv. All the **Higher level languages like C and C++, DBASE** etc. were used in this generation.

Fig: Fourth Generation

The main features of Fourth Generation are:
 VLSI technology used
 Very cheap
 Portable and reliable
 Use of PC's
 Very small size
 Pipeline processing
 No A.C. needed
 Concept of internet was introduced
 Great developments in the fields of networks
 Computers became easily available

Some computer of this generation were:
 DEC 10
 STAR 1000
 CRAY-1(Super Computer)

Fifth Generation

i. The period of Fifth Generation **is 1980-till date.**
ii. In the fifth generation, the **VLSI technology became ULSI (Ultra Large Scale Integration) technology**, resulting in the production of microprocessor chips having ten million electronic components.
iii. This generation is based on parallel processing hardware and AI (**Artificial Intelligence)** software.
iv. AI is an emerging branch in computer science, which interprets means and method of making computers think like human beings.
v. All the Higher level languages **like C and C++, Java, .Net etc**. are used inthis generation.

AI includes:
 Robotics
 Neural networks
 Game Playing

.

Fig: Fifth Generation

The main features of Fifth Generation are:
- ✓ ULSI technology
- ✓ artificial intelligence
- ✓ Natural language processing
- ✓ Advancement in Parallel Processing
- ✓ friendly interfaces
- ✓ Availability of very powerful and compact

Some computer ofthis generation are:
- ✓ Desktop
- ✓ Laptop
- ✓ NoteBook
- ✓ UltraBook
- ✓ ChromeBook

1.5 Basic Computer Organization:

Introduction to digital computers

The architecture of the computers has not changed but the technology used to accomplish thoseoperations may vary from one computer to another computer. The basic computer organization remains the same for all computer systems. In 1950's, it took a room full of vacuum tubes and equipment to perform the tasks that are now replaced with a single chip, not bigger than a child's thumb nail. Advances in semiconductor technology have reduced the Size of the chip.

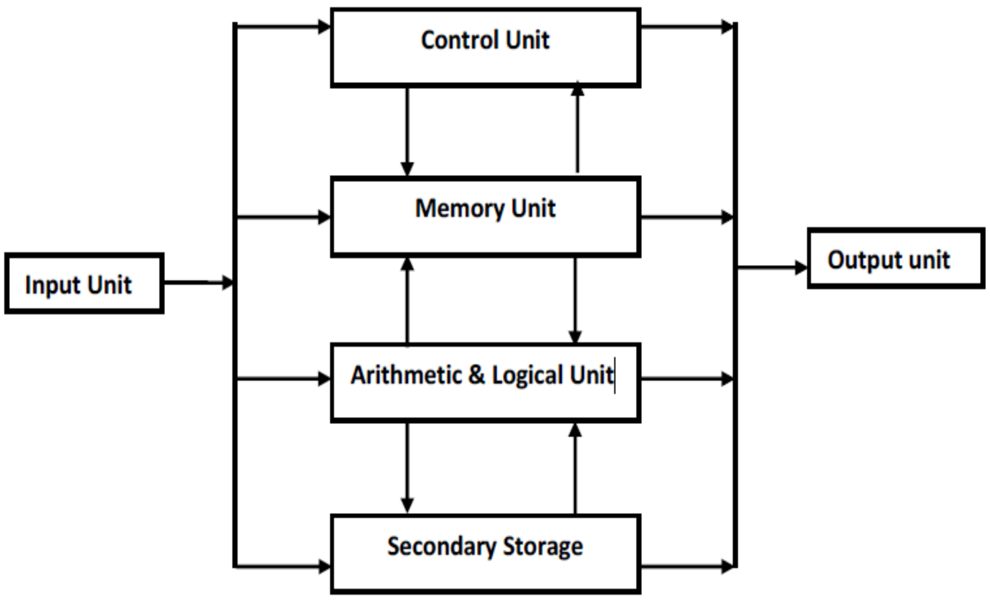


FIG**:block diagram of computer/architecture** diagram

The block diagram of the digital computer system has the following three units.
1. Input unit
2. Central processing unit
3. Output unit

1. Input unit

Computers need data and instructions in order to solve any problem. We need **to put the dataand instructions into the computers.** The input unit consists of one or more input devices.There are number of devices that perform the function of input devices. The keyboard of thecomputer is one of the most commonly used and standardized input device .Regardless of thetype of input device used in a computer system, all input devices perform the following**functions:**

1. Accept data and instruction from the outside world.
2. Convert it to a form that the computer can understand.
3. Supply the converted data to the computer system for further processing.

The input unit is used to send information or instructions or commands to the computer. The inputreceived from the input unit is immediately stored in main memory and the processed. Input may beprocessed. Input may be pressing a key in a keyboard or moving of mouse or movement of a joystick.

Following **are few of the important input devices** which are used in Computer Systems

- ✓ Keyboard
- ✓ Mouse
- ✓ Joy Stick
- ✓ Light pen
- ✓ Track Ball
- ✓ Scanner
- ✓ Graphic Tablet
- ✓ Microphone
- ✓ Magnetic Ink Card Reader (MICR)
- ✓ Optical Character Reader (OCR)
- ✓ Bar Code Reader
- ✓ Optical Mark Reader

Keyboard

Most common and very popular input device is keyboard. The keyboard helps **in inputting thedata to the computer.** The layout of the keyboard is like that of traditional typewriter, although thereare some additional keys provided for performing some additional functions.Keyboards are **of two sizes 84 keys or 101/102 keys, but now 104 keys or 108 keys keyboard isalso available for Windows and Internet.**

Fig: Keyboard

Mouse

Mouse is most **popular Pointing device**. It is a very **famous cursor-control device**. It is a small palm size box with a round ball at its base **which senses the movement of mouse** and sends**corresponding signals to CPU on pressing** the buttons. Generally it has two buttons called left and right button and scroll bar is present at the mid. Mouse can be used to control the position of cursor on screen, but it cannot be used to enter text into the computer.

.

Fig: Mouse

Joystick

Joystick is also **a pointing device** which is used to move cursor position on a monitor screen. It is a stick having a spherical ball at its both lower and upper ends. The lower spherical ball moves in a socket. The **Joystick can be moved in all four directions**. The function of joystick is similar to that of a mouse. It is mainly used in **Computer Aided Designing (CAD) and** playing computer games.

Fig: Joystick

Track Ball

Track ball is an **input device that is mostly used in notebook** or laptop computer, instead of amouse. This is a ball which is half inserted and **by moving fingers on ball**, pointer can be moved.Since the whole device is not moved, a track ball requires less space than a mouse. A track ball comes in various shapes like a ball, a button and a square.

Fig: Track Ball

Scanner

Scanner is an **input device which works more like a photocopy machine.** It is used whensome **information is available on a paper and it is to be transferred to the hard disc** of thecomputer for further manipulation.Scanner captures images from the source which are then converted into the digital form that can be stored on the disc. These images can be edited before they are printed.

Fig:scanner

2. Central processing unit

It is **the heart of the computer system**. All operations are carried out in this unit only. In most modern computers, a single IC does the following job

i) It performs all calculations
ii) It makes all decisions
iii) It controls all units of the computer

The CPU is sub-divided into the following sub-system:

- Control unit
- Arithmetic and logical unit (ALU)
- Memory unit
- Secondary storage devices.

(i) Control unit

The control **unit instructs the computer how to carry out program instructions**. It directs theflow of data between memory and arithmetic logical unit. It controls and coordinates the entirecomputer system.
The control **unit controls all other parts of the computer.**

The **input unit does not know when toreceive data and where to put the data in the storage unit after receiving it. The control unit gives thenecessary instructions to the input unit**,the control **unit instructs the input unit where to store the dataafter receiving** it from user.

In the same way, it controls the flow of data and instructions from thestorage unit to ALU. It also controls the flow of the results from the ALU to the storage unit. The controlunit also controls what should be sent to the output unit.

(ii) Arithmetic and logical unit

Arithmetic and logical **unit performs all the arithmetic and logical operations, arithmeticoperations like addition, subtraction, multiplication and logical operations such as comparisons.**

All calculations are performed in the arithmetic and logical unit (ALU) of the computer. ALU alsodoes comparisons and takes decision. Whenever any **calculation has to be done, the control unittransfers the required data from the storage unit to the ALU.**

(iii) Memory unit

Memory unit is the part of computer **which holds data for processing and** other information. Italso called **as main memory**. It is **a temporary storage (volatile).** The values stored in the processor withthe help of the many register like accumulator, base register. This registers store the value at the time ofcalculation.

(iv) Secondary storage devices

The secondary storage is used like an archive. It may store several programs, documents, database,etc. The program that you want to run on the computer is first transferred to the primary memorybefore it can run.

The secondary memory is slower than the primary memory. Some of the **commonlyused secondary memory devices are hard disk, compact disk (CD), digital video disk (DVD).** It is t**hepermanent storage; the data stored in this storage will not be erased when the power is lost. It is a nonvolatile**memory.

3. Output unit

These devices **used to get the response or result from the processor**. Output unit is the communication between the user and the computer. The output unit of a computer provides theinformation and results of a computation to the outside world. Commonly used **output devices are printer, plotter, and monitor.**

Following are few of the important output devices which are used in Computer Systems

- ✓ Monitors
- ✓ Graphic Plotter
- ✓ Printer

Monitors

Monitor **commonly called as Visual Display Unit (**VDU) is the main output device of acomputer. It forms images from tiny dots, called pixels, that are arranged in a rectangular form. The sharpness of the image depends upon the no. of the pixels.

There **are two kinds of viewing screen used** for monitors.

- ✓ Cathode-Ray Tube (CRT)
- ✓ Flat- Panel Display

Cathode-Ray Tube (CRT) Monitor

In the CRT display is made up of small picture elements called pixels for short. The smaller thepixels, the better the image clarity, or resolution. It takes more than one illuminated pixel to formwhole character, such as the letter e in the word help.

A finite number of character can be displayed on a screen at once. The screen can bedivided into a series of character boxes - fixed location on the screen where a standard charactercan be placed.The most screens are capable of **displaying 80 characters of** data horizontally and 25 lines

vertically. There are some disadvantage of CRT

 Large in Size

 High Power consumption

¶ **Printers**

Printer is the most important output device, which is used to **print information on paper.**The printers that print **the characters by striking against the ribbon and onto the paper** arecalled impact printers.

Characteristics of Impact Printers are following

- o **Very low consumable costs**
- o **Impact printers are very noisy**
- o **Useful for bulk printing due to low cost**
- o **There is physical contact with the paper to produce an image**

2 Types of computers/ Classification of computer

The computers come in many sizes and capabilities. It can be classified into the following based on hardware, utility, size and capacity.

1.Based on hardware design	1. Analog 2. Digital 3. Hybrid
2. Based on utility	1. General purpose computer 2. Special purpose computer
3.Based on size and capacity	1. Micro computer 2. Mini computer 3. Mainframe computer 4. Super computer
4.based on mode of use	1. palmtop pc's 2. laptop pc's 3. personal computer 4. work station 5. mainframe system 6. client and server

2.1 Based on hardware design

Based on hardware design, computers are classified as

1. Analog computer
2. Digital computer
3. Hybrid computer

Analog Computer

Analog **is Greek word which means similar. So** in analog computes, **any two quantities**are **measured by electrical voltage or current**. The analog computer operates **by measuringinstead of counting.** The analog computer works on **the supply of continuous electrical signals.**The display is also continuous and its output is in the form of graphs. Today analog computersare obsolete.

Digital computer

As the name implies, the digital computer handles quantities **represented as digits.** Indigital computer, **both numerical and non-numeric information's** are represented as string ofdigits.In digital computer, the input data is discrete in nature. It is represented by binarynotation in the form of 0's and 1's. Digital computer are much faster than analog computers andfar more accurate. Digital computer is largely used for business and scientific applications.

Hybrid computer

In hybrid computer, an attempt is made to combine the qualities **of both analog anddigital computer**. In a hybrid computer, the measuring functions are performed by the analogway while control and logic functions are digital in nature. Weather-monitoring system anddevices used in intensive care units of the hospitals are examples of hybrid computer.

2.2 Based on utility

Based on utility, the computers can be classified into

1. General purpose computers
2. Special purpose computers

General purposecomputers:

These are **designed and constructed to cater to almost all the needs of the society.** They can perform various operations. They are able to perform according to the programs created to meet different needs .These can be used for a variety of tasks **form financial accounting to mathematical calculations** form designing textile prints to controlling machinery. They are also flexible and can be used to work on business and scientific problems.

Special purpose computers

Special purpose computers can be **designed to perform specific functions**. In such devices, the instructions are permanently pre programmed. The instructions needed to perform the particular task are incorporated **into the internal memory of the computer.** Some of the special purpose computer are **aircraft control system, electronic voting machines etc.**

Based on size and capacity

Based on size and capacity, computers are classified into the following:

1. **Micro computer**
2. **Mini computer**
3. **Mainframe computer**
4. **Super computer**

Micro computers

In view of **its small size and the use of Microprocesso**r, this computer is called Micro computer.A microprocessor is a processor whose components namely Input, Output and CPU are on **a single integrated-circuit chip**. It is a **low-cost and** the word length of a microcomputer lies in the **range of 8 to 32 bits.**

Fig. Micro computer

They are normally single-microprocessor, single-user systems designed for performing basic operations like general **purpose calculations, industrial control, instrumentation, educational, training, small business applications,** home applications, **commercial equipment control, watches, fuel injection of a car, office automation, playing games** etc.

Mini computer

Minicomputer is **larger than the micro computers** and **is more powerful in terms** of processing power.Minicomputer is mainly multiprocessor systems where **many users simultaneously work** on the systems. Minicomputer possesses **greater storage capacity** and **larger memories** as compared to microcomputer.**Their word length is 32 bits.**

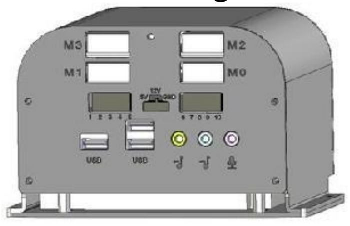


Fig.mini computer

Mainframe computers

Mainframe computer are **larger, faster and more expensive** than other general purpose computers. These are used to **handle huge volumes of data**. Their **word length may be 48 to 64 bit memory** capacity. These computers even possess and work with more than **one processor at the same time.**

Fig:Mainframe computer

Super computers

Super computers are **the most powerful** of all computers. They have a **high processing speed**.Their processing **speed lies in the range of 400 to 10,000 MIPS**. **Word length 64 to 96 bit**. It is specially designed to maximize the number of FLOPS. Their FLOPS rating is usually more than 1 gigaflop per second.

Fig:Super computer

2.4 Based on mode of use

However with **the rapidly developing technology,** such classifications are no more relevant.After a few months of introduction of a new computer in the marker, the new models of computers arerapidly introduced with **less cost** and **higher performance than the earliest.**

Palmtop pc's

The palmtop computers **accept handwritten inputs using an electronic pen**, which can be used to write on a palmtop's screen. These **have small disk storage** and can be **connected to a wireless network**. One has to train the system on the user's handwriting before it can be used. A palmtop computer has also facilities to be used as a mobile **phone, fax and E-mail machine.**

Fig:World first palmtop pc’s

Laptop pc’s

Laptop pc’s are portable computers with **less weight and small enough to** be kept on a lap. They use a keyboard, Flat screen liquid crystal display, and a Pentium or power pc processor. **Color displays are also available. It** is a battery powered personal computer generally smaller than a briefcase that can easily be transported and conveniently used in temporary space. **Laptops come with hard disk, CD ROM and floppy disk drives.**

3. Number systems

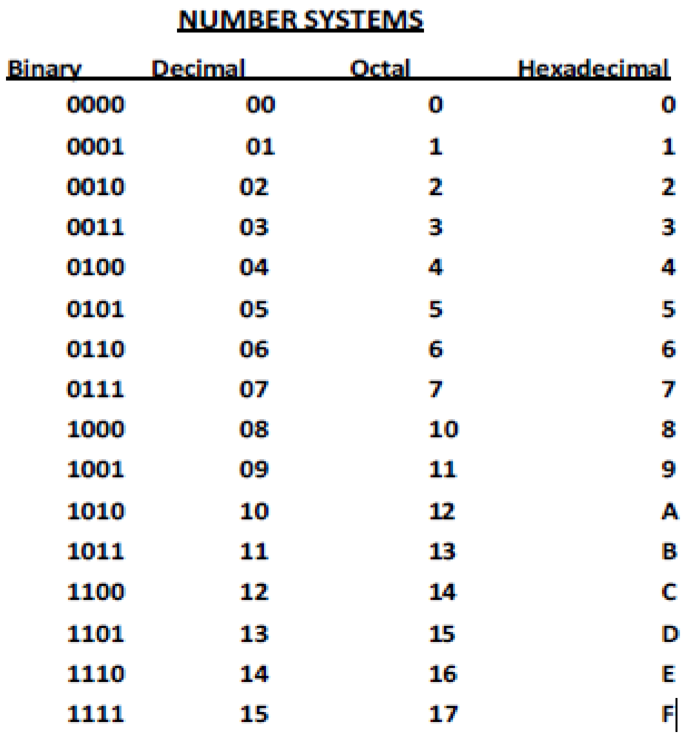
Introduction

A number is required for counting or to express the amount of some quantity. It consists of group of symbols called digits, which are arranged in a definite manner.. This gives rise to what we call a number system.

The most widely adopted system is

- **the decimal number system which has ten digits (0,1,2,3 …9),**
- **the octal system has eight digits (0,1,2,3…, 7),**
- **the hexadecimal system has sixteen digits (0,1,2,3…,9,A,B,C,D,E,F),**
- **the binary number system has only two (0,1), symbols.**

The number of digits in a system is **called ‘radix’ or ‘base’. So** that **decimal system may be called as radix-10** system**, the binary system as radix-2 system.**


NUMBER SYSTEMS

Binary	Decimal	Octal	Hexadecimal
0000	00	0	0
0001	01	1	1
0010	02	2	2
0011	03	3	3
0100	04	4	4
0101	05	5	5
0110	06	6	6
0111	07	7	7
1000	08	10	8
1001	09	11	9
1010	10	12	A
1011	11	13	B
1100	12	14	C
1101	13	15	D
1110	14	16	E
1111	15	17	F

3.1 Conversion of number system

Different number system conversions are followed.

1. Decimal to binary
2. Binary to decimal
3. Decimal to octal
4. Decimal to Hexadecimal
5. Octal to decimal
6. Octal to binary
7. Binary to octal
8. Hexadecimal to binary
9. Binary to hexadecimal
10. Hexadecimal to decimal

11. Octal to hexadecimal
12. Hexadecimal to Octal

1. Conversions of Decimal to Binary-

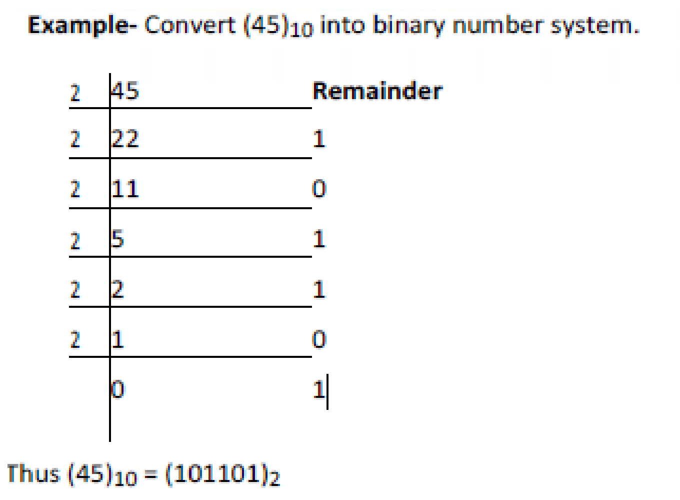
The method that is used for converting of decimals into binary is known as the remainder method. We use the following steps in getting the binary number-

(a) Divide the decimal number by 2.

(b) Write the remainder (which is either 0 or 1) at the right most position.

(c) Repeat the process of dividing by 2 until the quotient is 0 and keep writing the remainder after each step of division.

(d) Write the remainders in reverse order.



Example- Convert $(45)_{10}$ into binary number system.

		Remainder
2	45	
2	22	1
2	11	0
2	5	1
2	2	1
2	1	0
	0	1

Thus $(45)_{10} = (101101)_2$

Conversions of Decimal Fractions to Binary Fractions-

For converting decimal fractions into binary fractions, we use multiplication. Instead of looking for a remainder we look for an integer. The following steps are used in getting the binary fractions-

(a) Multiply the decimal fraction by 2.

(b) If a non-zero integer is generated, record the non-zero integer otherwise record 0.

(c) Remove the non-zero integer and repeat the above steps till the fraction value becomes 0.

(d) Write down the number according to the occurrence.

Example- Find the binary equivalent of (0.75)10.

0.75 × 2 = 1.50 1
0.50 × 2 = 1.00 1

Thus (0.75)10= (0.11)2.

Moreover, we can write (45.75)10= (101101.11)2.

Example- Find the binary equivalent of (0.9)10.

0.9 × 2 = 1.8 1
0.8 × 2 = 1.6 1
0.6 × 2 = 1.2 1
0.2 × 2 = 0.4 0
0.4 × 2 = 0.8 0

0.8 × 2 = 1.6 1
0.6 × 2 = 1.2 1
0.2 × 2 = 0.4 0
0.4 × 2 = 0.8 0
0.8 × 2 = 1.6 1
0.6 × 2 = 1.2 1
0.2 × 2 = 0.4 0
0.4 × 2 = 0.8 0
0.8 × 2 = 1.6 1
0.6 × 2 = 1.2 1
0.2 × 2 = 0.4 0
0.4 × 2 = 0.8 0
0.8 × 2 = 1.6 1

Thus (0.9)10 = (0.111001100110011001)2.

2. Conversions of Binary to Decimal –

In converting binary to decimal, we use the following steps-

(a) Write the weight of each bit.

(b) Get the weighted value by multiplying the weighted position with the respective bit.

(c) Add all the weighted value to get the decimal number.

Example- Convert (101101)2 into decimal number system.

Thus (101101)2= $1 \times 2^5 + 0 \times 2^4 + 1 \times 2^3 + 1 \times 2^3 + 0 \times 2^1 + 1 \times 2^0$

= 32 + 0 + 8 + 4 + 0 + 1

= 45

Conversions of Binary Fractions to Decimal Fractions –

The conversions of binary fractions to the decimal fractions is similar to conversion of binary numbers to decimal numbers. Here, instead of a decimal point we have a binary point. The exponential expressions (or weight of the bits) of each fractional placeholder is 2^{-1}, 2^{-2}………

Example- Convert (101101.11)2 into decimal number system

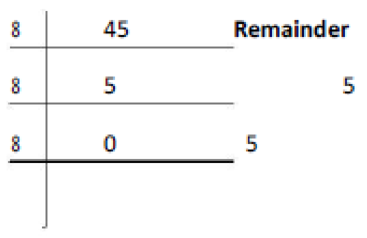
$=1 \times 2^5 + 0 \times 2^4 + 1 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 + 1 \times 2^{-1} + 1 \times 2^{-2}$

Thus (101101.11)2= 32 + 0 + 8 + 4 + 0 + 1 + 0.5 + 0.25 = 45.75

3. Conversion of Decimal to Octal-

In converting decimal to octal, we follow the same process of converting decimal to binary. Instead of dividing the number by 2, we divide the number by 8.

Example- Convert (45)10 into octal number system.



		Remainder
8	45	
8	5	5
8	0	5

Thus $(45)_{10}$= $(55)_8$.

Conversions of Decimal Fractions to Octal Fractions –

We follow the same steps of conversions of decimal fractions to binary fractions. Here we multiply the fraction by 8 instead of 2.

Example- Find the octal equivalent of (0.75)10.

0.75 × 8 = 6.00 6

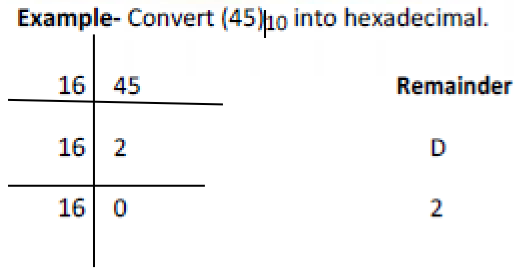
Thus (0.75)10= (0.6)8.

And (45.75)10= (55.6)8.

4. Conversion of Decimal to Hexadecimal –

We divide by 16 instead of 2 or 8. If the remainder is in between 10 to 16, then the number is represented by A to F respectively.

Example- Convert $(45)_{10}$ into hexadecimal.



		Remainder
16	45	
16	2	D
16	0	2

Thus $(45)_{10}$= $(2D)_{16}$.

Conversions of Decimal Fractions to Hexadecimal Fractions –

Here we multiply the fraction by 16 instead of 2 or 8. If the non-zero integer is in between 10 to 16, then the number is represented by A to F respectively.

Example- Find the hexadecimal equivalent of (0.75)10.

0.75 × 16 = 12.00 C (12 = C) Thus (0.75)10=(0.C)16.

And (45.75)10= (2D.C)16.

5. Conversions of Octal to Decimal-

We follow the same steps of conversion of binary to decimal. The only difference is that here weight of n^{th} bit is 8^{n-1} instead of 2^{n-1}.

Example- Convert (55)8 into decimal number system.

= $5 \times 8^1 + 5 \times 8^0$

= Thus (55)8 = 40 + 5.

= 45

Conversions of Octal Fractions to Decimal Fractions-

The weight of the bit of the fraction placeholder is 8^{-1}, 8^{-2}………. We follow the same steps of conversion of binary fractions to decimal fractions.

Example- Convert (55.6)8 into decimal number system.

$=5 \times 8^{1} + 5 \times 8^{0} + 6 \times 8^{-1}$

Thus (55.6)8= 40 + 5 + 0.75 = 45.75

6. Conversions of Octal to Binary-

We use the following steps in converting octal to binary-

(a) Convert each octal digit into 3-bit binary equivalent.

(b) Combine the 3-bit section by removing the spaces to get the binary number.

Example- Convert (55)8 into binary.

Octal Number	5	5
Binary Number	101	101

Thus $(55)_8 = (101101)_2$.

Example- Convert $(456)_8$ into binary.

Octal Number	4	5	6
Binary Number	100	101	110

Thus $(456)_8 = (100101110)_2$.

Conversions of Octal Fractions to Binary Fractions-

We follow the same steps of conversion of octal to binary.

Example- Convert (55.6)8 into binary.

Octal Number	5	5	6
Binary Number	101	101	110

Thus $(55.6)_8 = (101101.11)_2$.

7.Conversions of Binary to Octal-

We use the following steps in converting binary to octal-

(a) Break the number into 3-bit sections starting from LSB to MSB.

(b) If we do not have sufficient bits in grouping of 3-bits, we add zeros to the left of MSB so that all the groups have proper 3-bit number.

(c) Write the 3-bit binary number to its octal equivalent.

Example- Convert (101101)2 into octal.

Example- Convert $(101101)_2$ into octal.

Binary Number	101	101
Octal Number	5	5

Thus $(101101)_2 = (55)_8$.

Example- Convert $(1101101)_2$ into octal.

Binary Number	001	101	101
Octal Number	1	5	5

Thus $(1101101)_2 = (155)_8$.

Conversions of Binary Fractions to Octal Fractions-

We use the following steps in converting binary fractions to octal fractions-

(d) Break the fraction into 3-bit sections starting from MSB to LSB.

(e) In order to get a complete grouping of 3 bits, we add trailing zeros in LSB.

(f) Write the 3-bit binary number to its octal equivalent.

Example- Convert (101101.11)2 into octal.

Example- Convert $(101101.11)_2$ into octal.

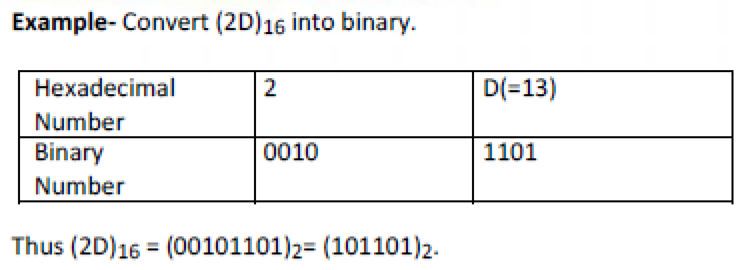
Binary Number	101	101	110
Octal Number	5	5	6

Thus $(101101)_2 = (55.6)_8$.

8. Conversions of Hexadecimal to Binary-We use the following steps-

(a) Convert each hexadecimal digit to its 4-bit binary equivalent.

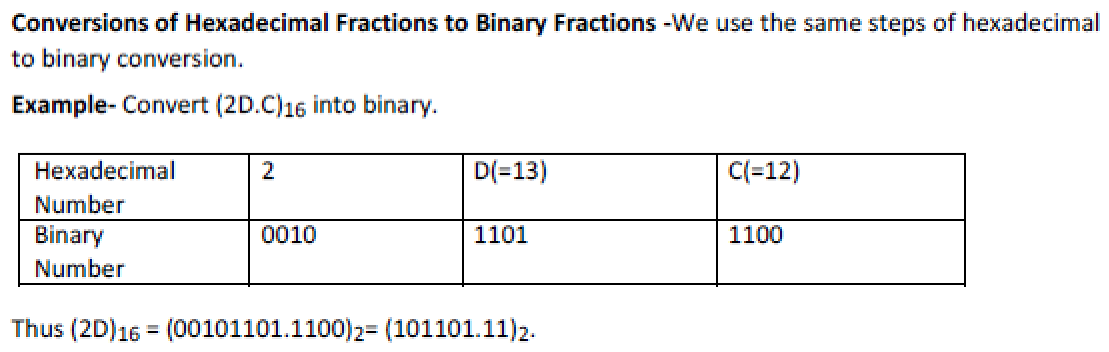
(b) Combine all the binary numbers



Example- Convert $(2D)_{16}$ into binary.

Hexadecimal Number	2	D(=13)
Binary Number	0010	1101

Thus $(2D)_{16} = (00101101)_2 = (101101)_2$.



Conversions of Hexadecimal Fractions to Binary Fractions -We use the same steps of hexadecimal to binary conversion.

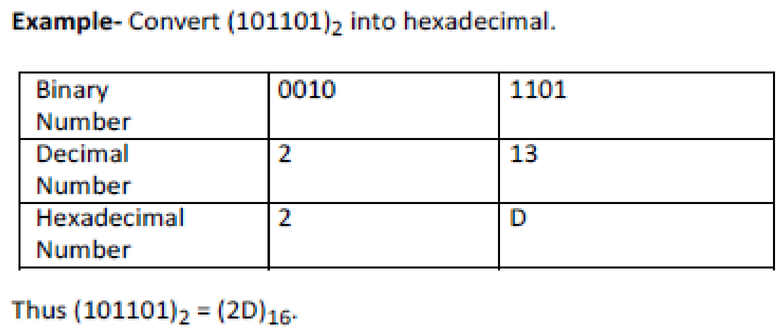
Example- Convert $(2D.C)_{16}$ into binary.

Hexadecimal Number	2	D(=13)	C(=12)
Binary Number	0010	1101	1100

Thus $(2D)_{16} = (00101101.1100)_2 = (101101.11)_2$.

9. Conversions of Binary to Hexadecimal- We convert binary to hexadecimal in the similar manner as we have converted binary to octal. The only difference is that here, we form the group of 4 – bits.

Example- Convert (101101)2 into hexadecimal.



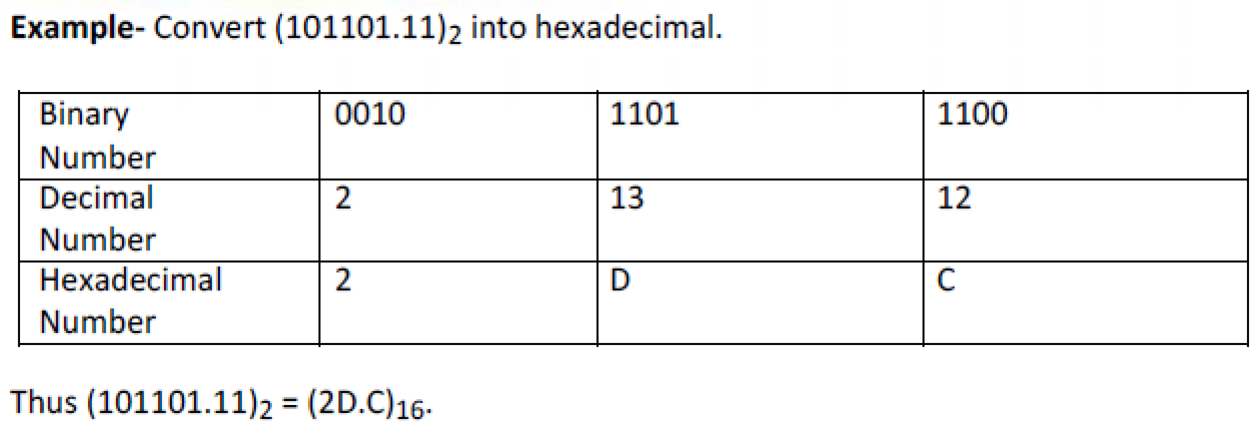
Example- Convert $(101101)_2$ into hexadecimal.

Binary Number	0010	1101
Decimal Number	2	13
Hexadecimal Number	2	D

Thus $(101101)_2 = (2D)_{16}$.

Conversions of Binary Fractions to Hexadecimal Fractions –

We convert binary fractions to hexadecimal fractions in the similar manner as we have converted binary fractions to octal fractions. The only difference is that here we form the group of 4 – bits



Example- Convert $(101101.11)_2$ into hexadecimal.

Binary Number	0010	1101	1100
Decimal Number	2	13	12
Hexadecimal Number	2	D	C

Thus $(101101.11)_2 = (2D.C)_{16}$.

10. Conversions of Hexadecimal to Decimal-

We do the conversion of hexadecimal to decimal as we have done the conversion of binary to decimal. Here weight of nth bit is 16^{n-1} instead of 2^{n-1}

Example- Convert $(2D)_{16}$ into decimal.

Hexadecimal Number	2	D(=13)
Wt. of each Bit	16^1	16^0
Weighted Value	2 × 16	13×16^0
Solved Multiplication	32	13

Thus $(2D)_{16}$= 32 + 13 = 45.

Conversions of Hexadecimal Fractions to Decimal Fractions-

We do the conversion of hexadecimal fractions to decimal fractions in the similar manner as we have done the conversion of binary fractions to decimal fractions. Here weight of bit is 16^{-1}, 16^{-2}…….

Example- Convert (2D.C)16 into decimal.

$=2 \times 16^1 + 13 \times 16^0 + 13 \times 16^{-1}$

$=32 + 13 + 0.75 = 45.75$

11.Conversions of Octal to Hexadecimal-

The conversion involves the following steps-

(a) Convert each octal digit to 3 –bit binary form.

(b) Combine all the 3-bit binary numbers.

(c) Group them in 4-bit binary form by starting from MSB to LSB.

(d) Convert these 4-bit blocks into their hexadecimal symbols.

Example- Convert (55)8 into hexadecimal.

Octal Number	5	5
Binary Number	101	101

Combining the 3-bit binary block, we have 101101.

Grouping them in 4 bit binary form-

Binary Number	0010	1101
Hexadecimal Symbol	2	D

Thus $(55)_8 = (2D)_{16}$.

Conversions of Octal Fractions to Hexadecimal Fractions

- The method of conversion is based on the same procedure that we have discussed in conversions of octal to hexadecimal.

Example- Convert (55.6)8 into hexadecimal.

Octal Number	5	5	6
Binary Number	101	101	110

Combining the 3-bit binary block, we have 101101.110.

Grouping them in 4 bit binary form-

Binary Number	0010	1101	1100
Hexadecimal Symbol	2	D	C

Thus $(55)_8 = (2D.C)_{16}$.

12.Conversions of Hexadecimal to Octal-

We convert each hexadecimal digit in binary. Combine all the binary numbers. Again group them into 3-bit form. Convert the 3-bit block in octal.

Example- Convert (2D)16 into octal.

Hexadecimal Number	2	D(=13)
Binary Number	0010	1101

Combining the binary number, we get 00101101=101101
Grouping the binary number in 3-bit

Binary Number	101	101
Octal Number	5	5

Thus $(2D)_{16} = (55)_8$.

Conversions of Hexadecimal Fractions to Octal Fractions –

We follow the same steps of hexadecimal to octal conversion.

Example- Convert (2D.C)16 into octal.

Hexadecimal Number	2	D(=13)	C(=12)
Binary Number	0010	1101	1100

Combining the binary number, we get 00101101.1100=101101.11
Grouping the binary number in 3-bit

Binary Number	101	101	110
Octal Number	5	5	6

Thus $(2D.C)_{16} = (55.6)_8$.

Software

- is a generic term for organized collections of computer data and instructions, often broken into two major categories two major categories:
  1. system software
  2. application software

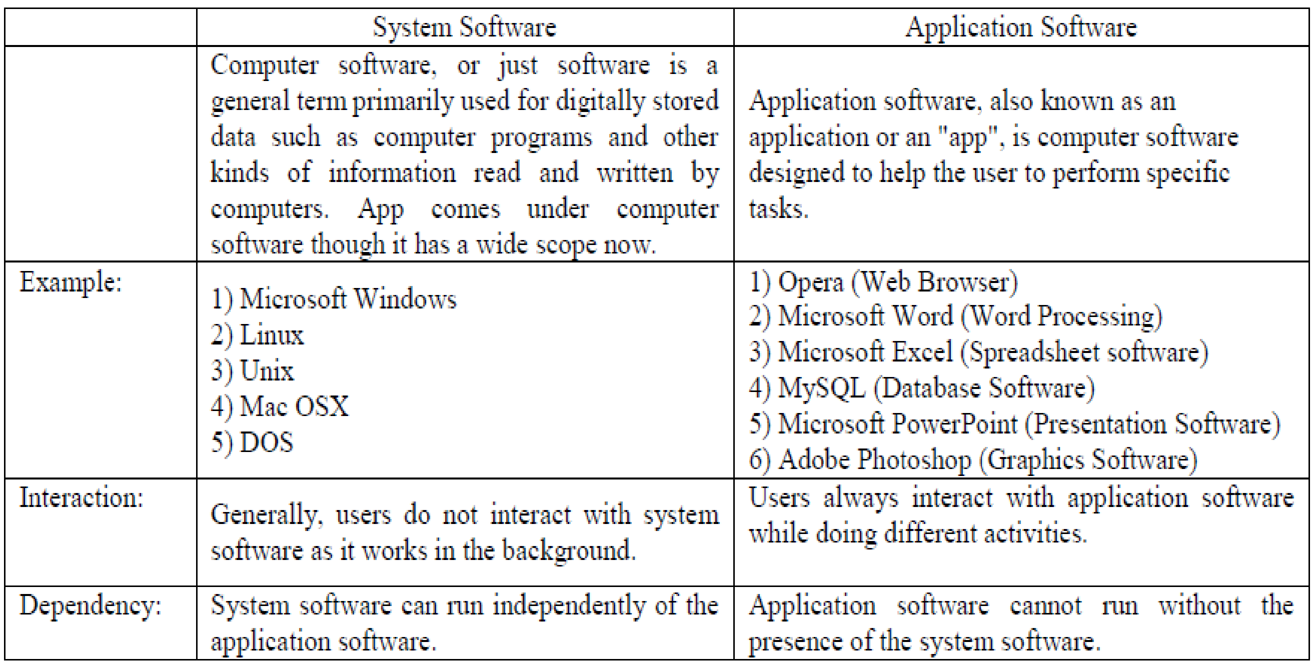
system Software

- Known as **Operating System**
- is responsible for **controlling, integrating, and managing** the individual hardware components of a computer system
- Windows is an example of OS.
- example of System Software:
- 1) Microsoft Windows
- 2) Linux
- 3) Unix
- 4) Mac OSX
- 5) DOS
-

Applicaton Software

- is used to accomplish **specific tasks other than just running the computer** system.
- It may consist of:
  - a single program, such as **an image viewer**;
  - a small collection of programs (often called a software package) that work closely together to accomplish a task, such as **a spreadsheet or text processing system;**
  - a larger collection (often called a software suite) of related but independent programs and packages that have a common user interface or shared data format, such as Microsoft Office,

Comparison Application Software and System Software



	System Software	Application Software
	Computer software, or just software is a general term primarily used for digitally stored data such as computer programs and other kinds of information read and written by computers. App comes under computer software though it has a wide scope now.	Application software, also known as an application or an "app", is computer software designed to help the user to perform specific tasks.
Example:	1) Microsoft Windows 2) Linux 3) Unix 4) Mac OSX 5) DOS	1) Opera (Web Browser) 2) Microsoft Word (Word Processing) 3) Microsoft Excel (Spreadsheet software) 4) MySQL (Database Software) 5) Microsoft PowerPoint (Presentation Software) 6) Adobe Photoshop (Graphics Software)
Interaction:	Generally, users do not interact with system software as it works in the background.	Users always interact with application software while doing different activities.
Dependency:	System software can run independently of the application software.	Application software cannot run without the presence of the system software.

SOFTWARE DEVELOPMENT PROCESS

- The software development process consists of four major steps. Each of these steps is detailed below.
- Step 1:Reqiurement Analysis
- Step 2:Planning
- Step 3: Designing
- Step 4: Implementing
- Step 5: Testing
- Step 6: Deployment and Maintenance

Requirement Analysis

- The **business analyst** interacts with **stakeholders to** develop the business requirements document.
- They also write use cases and share this information with the project team.
- The aim of the requirements analysis is for
    - **quality assurance,**
    - **technical feasibility, and**
    - **to identify potential risks**
- to address in order for the software to succeed.

Planning

- customer or stakeholder has requested a project,
- Alignment: How does this project connect to your company's larger mission and goals?
- Resource availability and allocation:
- Do you have the people and tools you need to take this on?
- Project scheduling: How does this project fit within your company's goals and other tasks?
- Cost estimation:
- How much is it going to cost?
- Check feasiblity(legal, budget,time.,cost)
- Scope of current release and project plan document

Designing:

- During the design phase, lead developers and technical architects create the initial high-level design plan for the software and system.
- This includes the delivery of requirements used to create the Design Document Specification (DDS).,
- security processes, as well as hardware and system requirements.
-

Implementing

- Implementation is the part of the process where software engineers actually program the code for the project.
- Developing and coding
- In this phase, the database admin creates and imports the necessary data into the database. Programming languages are defined by requirements. Developers create the interface as per the coding guidelines and conduct unit testing.

- This is an important phase for developers. They need to be open-minded and flexible if any changes are introduced by the business analyst.

Testing

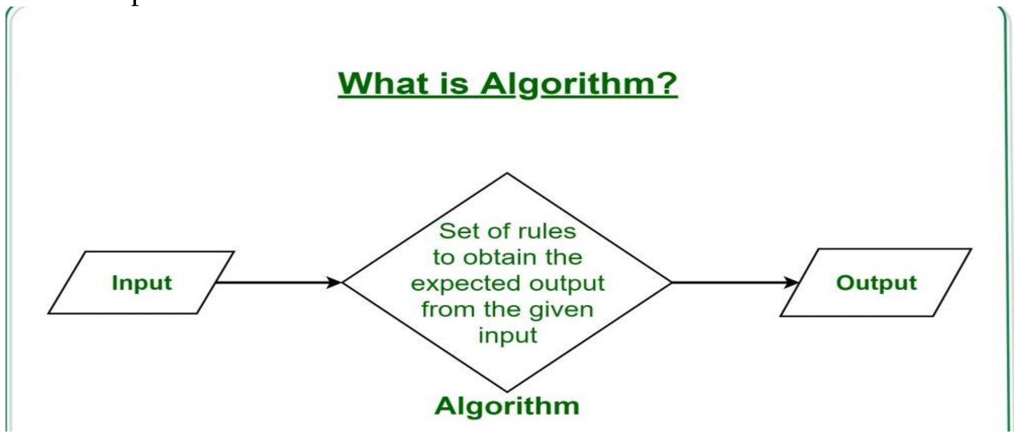
- Software testing can be stated as the process of validating and verifying that a software program/application/product:
- meets the requirements that guided its design and development;
- works as expected; and
- can be implemented with the same characteristics.

Deployment and Maintenance

- In a post-production, live software environment, the system is in maintenance mode.
- No matter the number of users, the sophistication of the software and rigorous QA testing, issues will occur.
- That's the nature of software with managing data, integration, and security, and real world usage.
- Access to knowledgeable, reliable support resources is essential, as is routine maintenance and staying up to date on upgrades.

Algorithm

It is defined as a **sequence of instructions** designed in such a way that if the instructions are executed in the specified sequence, the desired results will be obtained. The algorithm **should precise and unambiguous** in practice and the results should obtain **after a finite number of steps.** An algorithm is defined if any problem whose solution can be expressed in a list of executable instruction.


What is Algorithm?
Input
Set of rules to obtain the expected output from the given input
Output
Algorithm

Characteristics of algorithms

In order to qualify algorithms is a sequence of instructions, they must possess the following characteristics:

i) In the algorithms, each and every instruction should be precise.
ii) Each and every instruction should be unambiguous
iii) Ensure that the algorithm will ultimately terminate.
iv) The algorithm should be written in sequence.
v) It looks like normal English
vi) The desired result should be obtained only after the algorithm terminates

Qualities of a good algorithm

There are so many methods or logics available to solve the problem individually. All of those methods logics may not be good, for given problem; there may be so many algorithms not of all equality. The following are the primary factors that are often used to judge the quality of the algorithm..

s.no	Factors	Quality of good algorithm
1	Time	To execute a program, the computer system takes some amount of time. The lesser is the time required, the better is the algorithm.
2	Memory	Execute the program with in limited memory
3	Accuracy	Multiple algorithms may provide suitable or correct solutions to a given problem, some of these may provide more accurate results than others, such algorithms may be suitable.
4	Sequence	The procedure of an algorithm must form in a sequence and some of the instructions of an algorithm may be repeated in number of times or until a particular condition is met.
5	Generalized	The designed algorithm must solve a single isolated problem and more often algorithms are desired to handle a range of input data to meet this criteria, so the algorithms must be generalized.

Representation of algorithms

The algorithms can be represented in several ways. Generally the programmers follows one of the following ways to represent an algorithms:

i) Normal English
ii) Flowchart
iii) Pseudo code
iv) Decision table
v) Program

1	Normal English	The algorithm can be easily represented in step by step sequential order in normal English, such algorithms are easy to understand, write and read.
2	Flow chart	The flowchart is a pictorial representation of algorithms, the sequential steps in an algorithm can be represented as a flowchart using the standard symbols.
3	Pseudo code	The flowcharts are only one of the possible format decision tools. Whereas pseudo code is also a formal design tools and is very well with the rules of structured design and programming.
4	Decision table	A decision table helps a lot in designing a specific segment of design. It provides another way to look at a complex, nested selection to help clarify the conditions to be tested and how those conditions should be nested to arrive at the proper actions.
5	Program	The algorithms can be implemented as a program using any high level language that becomes a program.

A sample algorithm

Algorithm

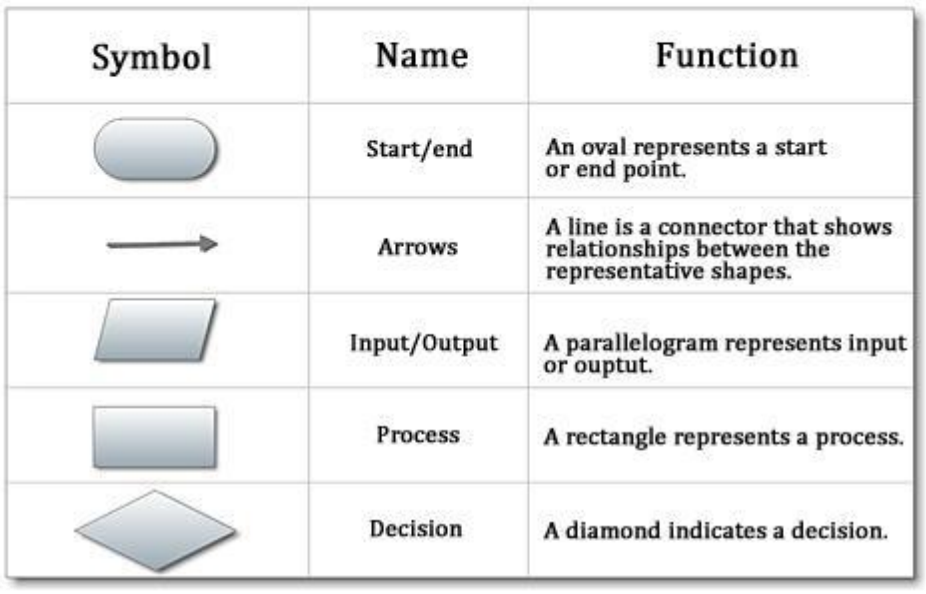
Step 1: start the program
Step 2: read number of students, name of the student, marks.
Step 3: calculate the total marks obtained by each student.
Step 4: check student mark is greater than 70 or not. If greater than 70 then “first class”.
Step 5: check student mark is greater than 60 and less than 70or not. If so then, “second class”.
Step 6: check student mark is greater than 50 and less than 60 or not. If so then, “third class”.
Step 7: print the result of the student with their names and marks.
Step 8: stop the program.

Flowchart

A **flowchart** is a type of diagram that represents an algorithm, workflow or process, showing the steps as boxes of various kinds, and their order by connecting them with arrows. This diagrammatic representation illustrates a solution to a given problem. Flowcharts are used in analyzing, designing, documenting or managing a process or program in
various fields.

Basic Flowchart Symbols

A flowchart is the visualization of a system or process consisting of five basic objects:



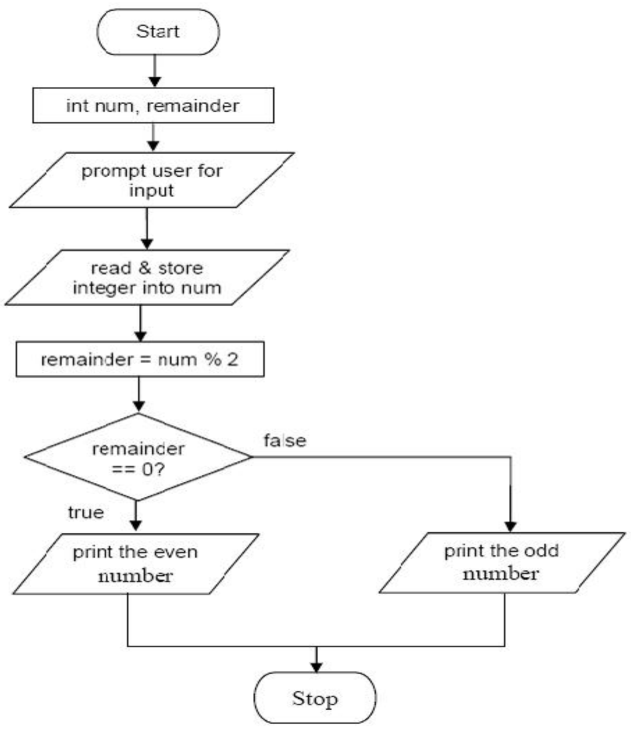
Symbol	Name	Function
⬭	Start/end	An oval represents a start or end point.
→	Arrows	A line is a connector that shows relationships between the representative shapes.
▱	Input/Output	A parallelogram represents input or ouptut.
▭	Process	A rectangle represents a process.
◇	Decision	A diamond indicates a decision.

2. ALGORITHM & FLOWCHART TO FIND THE GIVEN NUMBER IS EVEN OR ODD.

Algorithm:

Step 1: Include Header files.
Step 2: Read the value of a number.
Step 3: Check the given number by using if (n%2==0)
Step 4: if its remainder is zero, Print it is even number.
Step 5: If its remainder is one, Print it is odd number.
Step 6: Exit

Flowchart:

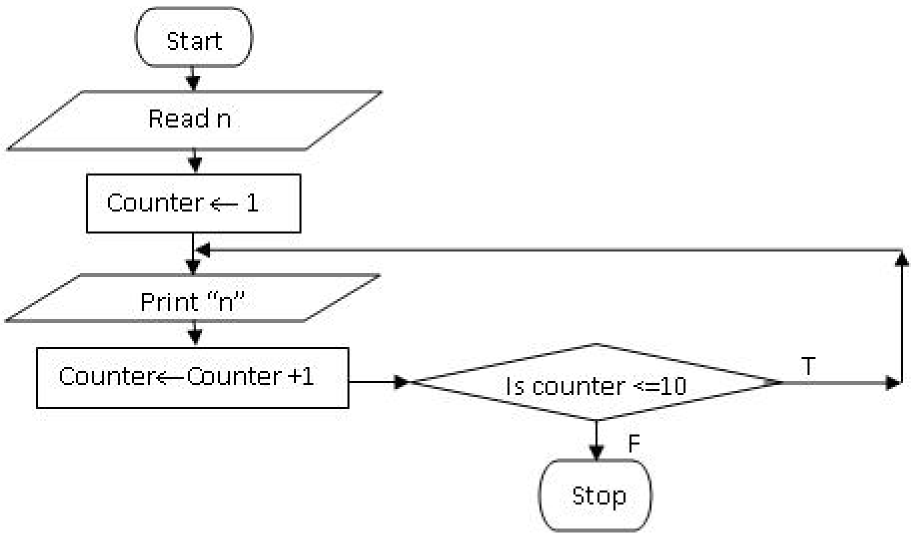


3. ALGORITHM AND FLOWCHART TO PRINT THE NUMBERS FROM 1 to 10.

Algorithm:

Step 1: Include Header files.
Step 2: Read the value of n.
Step 3: Initialize the value of counter by 1 using counter variable.
Step 4: Print the values upto the value of n by checking with if statement if(n<10).
Step 5: Exit.

Flowchart:



Advantages of Flowchart:

- Flowcharts are a **better way of communicating the logic** of the system.
- With the help of flowcharts programs can be **easily analyzed.**
- It provides **better documentation**.
- **Easy to trace errors in the** software.
- **Easy to understand.**
- It helps **to provide correct logic.**

Disadvantages of Flowchart

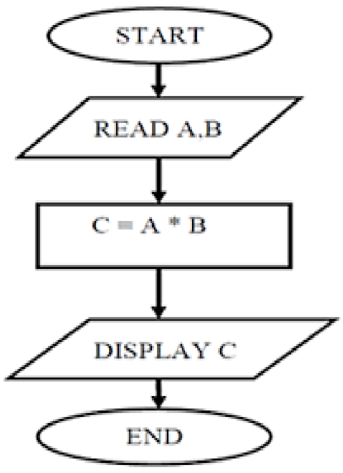
- It is **difficult t**o draw flowcharts for large and complex programs.
- **There is no standard** to determine the amount of detail.
- Some developer thinks that it **is waste of time.**
- If changes are done in software, then the flowchart must be **redrawn**

Types of flow chart

- Sequence Control structure
- Selection control structure
- Iterative control structure

Sequence Control structure

In sequence control structure, the steps are executed in linear order, one after other

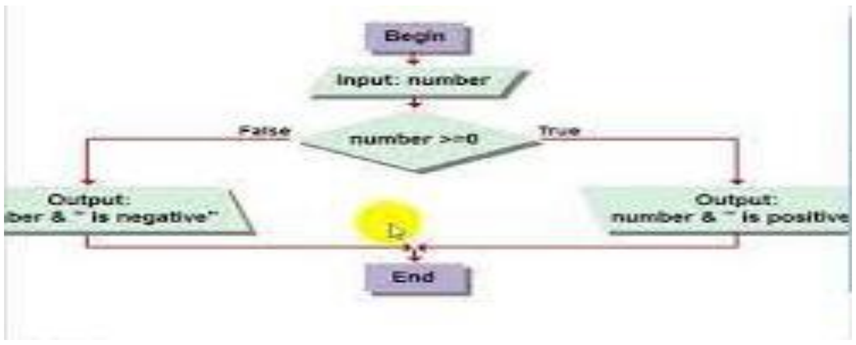


Selection control structure

Selection control structure

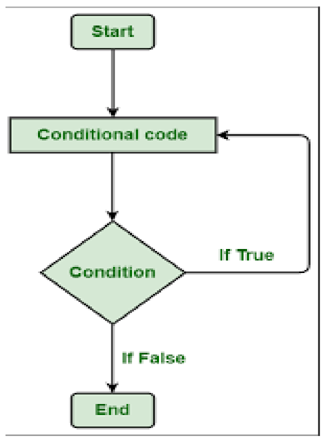
- In selection control structure, the steps to be executed next is based on a decision taken.

- If the condition is true , one path is followed
- If the condition is false , another path is followed.
- Ex. Flowchart to positive or negative number



Iterative control structure

- **To run program multiple times.**

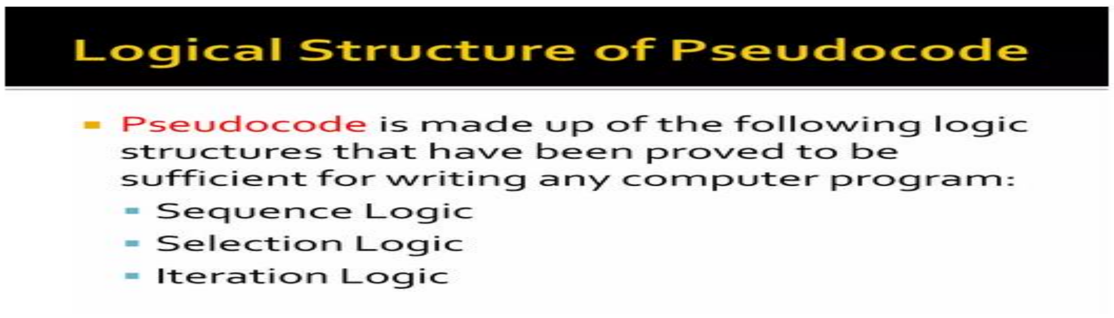


PSEUDOCODE

- Is an informal way of programming description that does not require any strict programming language syntax or underlying technology considerations.
- It is used for creating an outline or a rough draft of a program.
- Pseudo code summarizes a program's flow, but excludes underlying details.
- **Pseudocodeis not an actual programming language**.
- So it **cannot be compiled into an executable program.**
- It uses short terms or **simple English language** syntaxes to write code for programs before it is actually converted into a specific programming language.
- This is done to identify top level flow errors, and understand the programming data flows that the final program is going to use.

Advantage of Pseudo code

- Acts as a **bridge between the program and the algorithm or flowchart.**
- Also **works as a rough documentation**.
- developer can be **understood easily**.
- The main goal of a pseudo code is **to explain what exactly each line of a program should do**.



Logical Structure of Pseudocode

- Pseudocode is made up of the following logic structures that have been proved to be sufficient for writing any computer program:
  - Sequence Logic
  - Selection Logic
  - Iteration Logic

Sequence Logic:

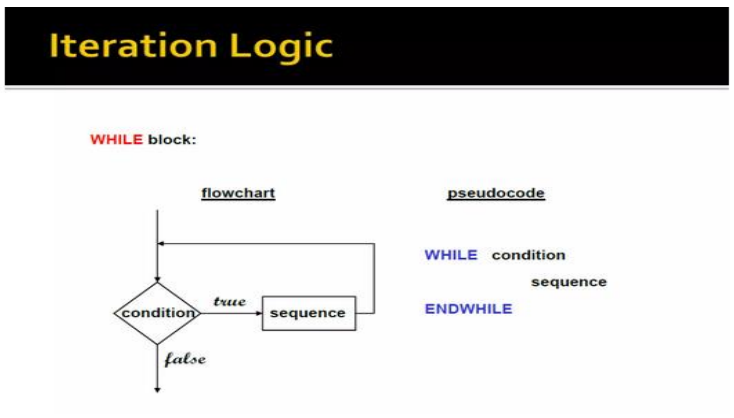
It is used to perform instructions in a sequence that is one after another. pseudo code instruction are written in an order in which they are to be performed.

```
BEGIN
NUMBER s1, s2, sum
OUTPUT("Input number1:")
INPUT s1
OUTPUT("Input number2:")
INPUT s2
sum=s1+s2
OUTPUT sum
END
```

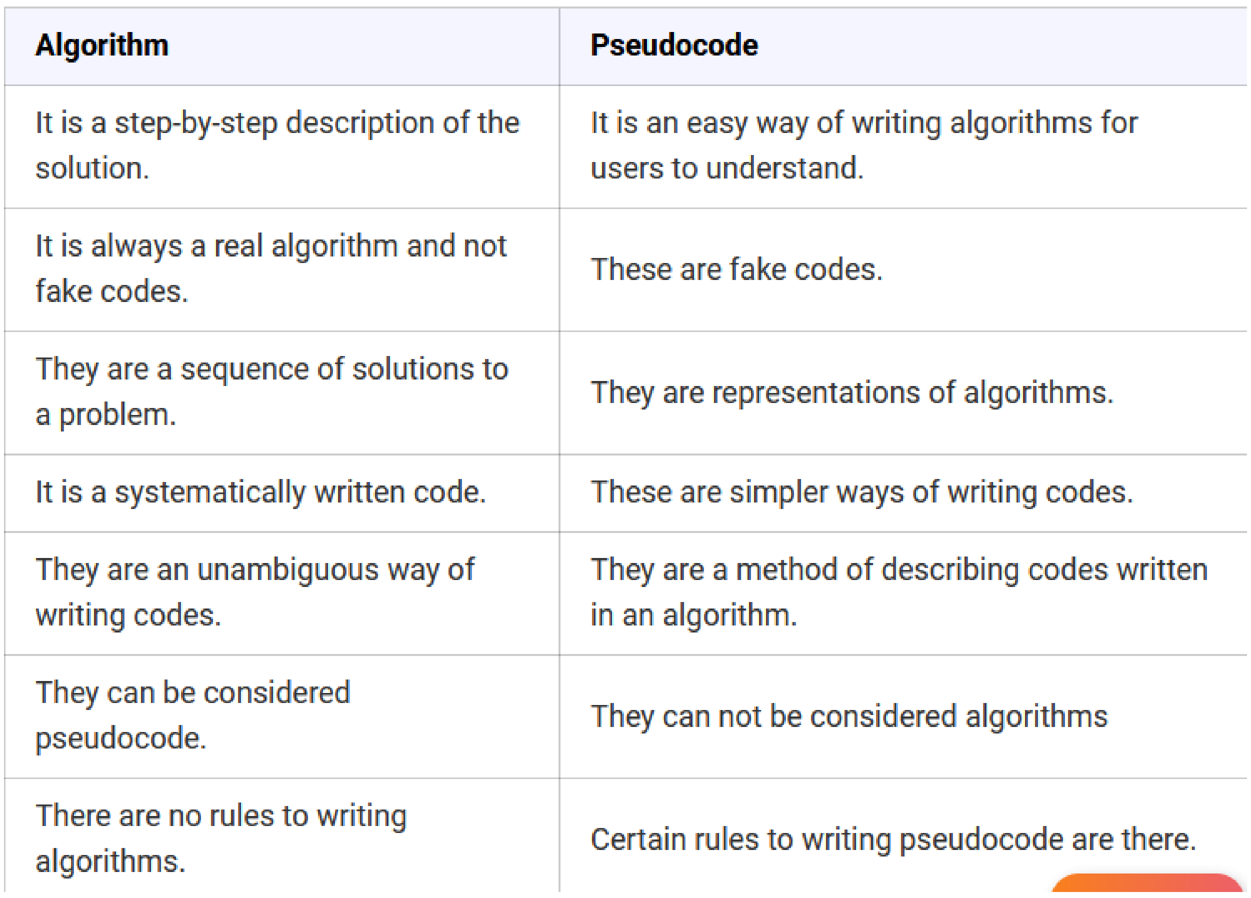
Selection logic:

```
BEGIN
```

```
 NUMBER num
 OUTPUT "Enter a Number"
OKU num
IF num>0  THEN
OUTPUT "Entered number is positive"
ELSE IF num<0 THEN
OUTPUT "Entered number is negative"
ELSE
OUTPUT "Entered number is zero"
ENDIF
 END
```



```
WHILE   condition
                sequence
ENDWHILE
```



Algorithm	Pseudocode
It is a step-by-step description of the solution.	It is an easy way of writing algorithms for users to understand.
It is always a real algorithm and not fake codes.	These are fake codes.
They are a sequence of solutions to a problem.	They are representations of algorithms.
It is a systematically written code.	These are simpler ways of writing codes.
They are an unambiguous way of writing codes.	They are a method of describing codes written in an algorithm.
They can be considered pseudocode.	They can not be considered algorithms
There are no rules to writing algorithms.	Certain rules to writing pseudocode are there.

Problem formulation & Problem Solving

Effective problem formulation is fundamental success of all analysis, particularly in command and control assessment because the problems are often ill defined and complex, involving many dimensions and rich contents.   Problem formulation involves decomposition of the analytic problems into appropriate dimensions such as structures, functions and mission areas.

Problem formulation is an interactive process that evolves over the course of study. It is essential even for small studies as where time is short, It will save time later help ensure quality.

The problem formulation phase should identify the context of the study and aspects of the problem related issues. There is no universal acceptance approach to problem formulation.

However, practices exist that can be applied. First find out what the question is then find out what the real question is.

Problem Solving:

When we start reading these and wants to learn how to solve a problem by using computers, it is first of all important to understand what the problem is. We need to read all the problem statements a number of times to ensure that is understands what is asked before attempting to solve t problem.

Method of problem solving:

1. Recognize and understand the problems
2. Accumulate facts
3. Select appropriate theory
4. Make necessary assumptions
5. Solve the problems
6. Verify the results

Performing step 5 solves the problem may involve a computer. The 5 steps in using a computer as a problem solving tool

1. Develop an algorithm and flowchart
2. Write a program in computer language
3. Enter the program in to computer
4. Test and debug the program
5. Run the program, input data, and get the results from computer.

C PROGRAMMING BASICS UNIT-II	**9**
Introduction to Unix Operating System - Introduction to ‘ C’ programming - fundamentals - structure of a ‘C’ program - compilation and linking processes - Constants, Variables - Data Types - Expressions using operators in ‘C’ - Managing Input and Output operations - Decision Making and Branching - Looping statements - solving simple scientific and statistical problems.	

Introduction to C

C – Language History

 C language is a structure oriented programming language, was developed at Bell Laboratories in1972 by Dennis Ritchie

 C language features were derived from earlier language called “B” (Basic CombinedProgramming Language – BCPL)

 C language was invented for implementing UNIX operating system

 In 1978, Dennis Ritchie and Brian Kernighan published the first edition “The C ProgrammingLanguage” and commonly known as K&R C

 In 1983, the American National Standards Institute (ANSI) established a committee to provide amodern, comprehensive definition of C. The resulting definition, the ANSI standard, or “ANSI C”,was completed late 1988.

Features of ’C’:

 C is a general purpose language

 C is a structural Language

 C is middle level language ie., it supports both the low and high level language features

 C is flexible and more powerful language

 C programs are fast and efficient

 C is most suitable for writing system software as well as application softwares

 Machine independent and portable

 C has the ability to extend itself, we can continuously add our own functions to the existing library functions

 C is the robust language

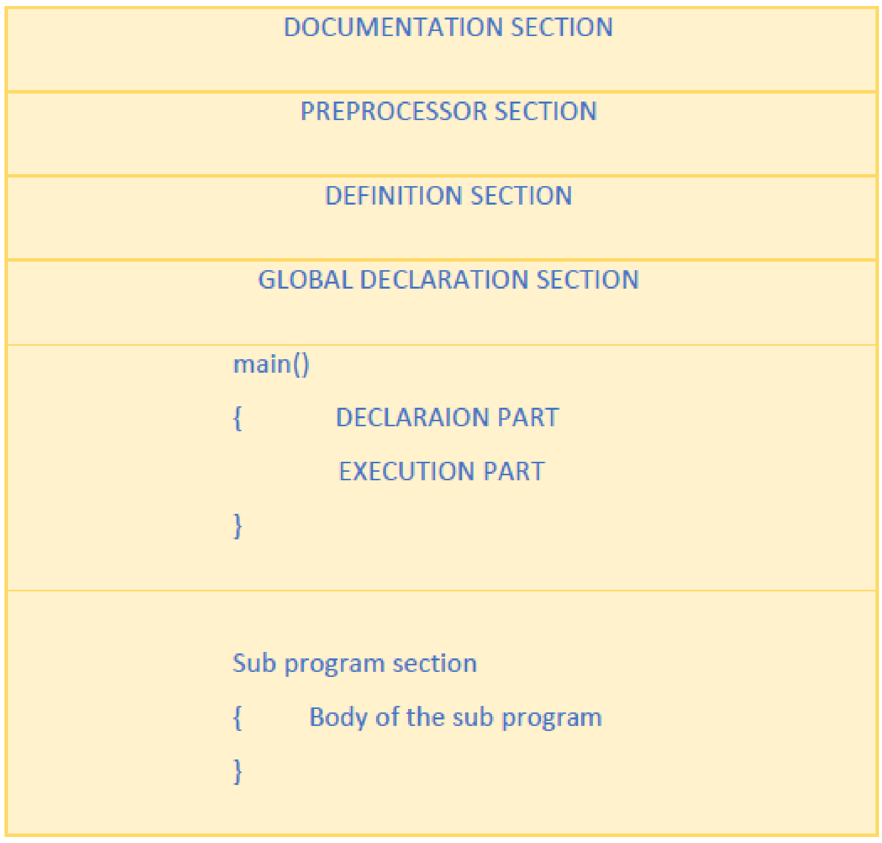
 C is widely available, commercial C compilers are available on most PC's

 Commands may be inserted anywhere in a program

 C has rich set of operators.

 C language allows reference to memory location with the help of pointers, which holds the address of the memory locations

STRUCTURE OF C PROGRAM



i. Documentation Section:

It consists of set of command lines used to specify the name of the program, the author of the program and other details etc..

Comments:

Comments are very helpful in identifying the program features and underlying logi of the program. The lines with '/*' and ending with '*/' are known as comment lines. These are not executable, the compiler is ignored anything in between /* and */

ii. Preprocessor Section:

It is used to link system library files, for defining the macros and for defining the conditional inclusion.
Ex : include <stdio.h>

iii. DefintionSection:

The definition section defines all symbolic constants.
Ex: # define pi 3.14

iv. Global Declaration Section:

The variable that are used in more than one function throughout the program are called global variable and are declared outside of all the function. Ie.mmain() function.

v. Main Function:

Every C program must have one main function, which specify the starting of C program

Declaration Part:

This part is used to declare all the variables that are used in the executable part of the program and these are called local variables

Executable Part:

It contains at least one valid C statements
 The execution of a program begins with opening brace '{' and ends with '}'

Example C program to compare all the sections:

/* C basic structure program **Documentation section**

```
Author: fresh2refresh.com
Date : 01/01/2012
*/
#include <stdio.h>/* Link section */
int total = 0; /* Global declaration and definition section */
int sum (int, int); /* Function declaration section */
int main () /* Main function */
{
printf ("This is a C basic program \n");
total = sum (1, 1);
printf ("Sum of two numbers : %d \n", total);
return 0;
}
int sum (int a, int b) /* User defined function */
{ /* definition section */
return a + b;
}
```

Rules for writing C program:

- All the statements should be in lower case letters.
- Upper case letters are only used for symbolic constants.
- Blank spaces may be inserted between two words. It is not used when declaring variables, keywords, constants and functions.
- The program statements can write anywhere between the two braces following the declaration part
- The user can also write one or more statements in one line separating them with semicolon (;)

Executing C Program

Execution is the process of running the program, to execute a ‘C’ program, we need to follow the steps given below.

i. Create the program

ii. Compiling the program

iii. Linking the program with system library

iv. Executing the program

i. Creating the program:

Creating the program means entering and editing the program in statndard C editors and save the program with an extension .c

ii. **Compiling the program**

- This is the process of converting the high level language program into machine understandable form. For this purpose compiler is used. Usually this can be done in C language by pressing ALT+F9 or compile from compile menu
- Here there are possibility to show errors ie., syntax errors, means the statements written in program are not in proper syntax

iii. **Linking the program with system library**

- C language program is the collection of predicted functions.
- These functions are already written in some standard C header files,
- Therefore, before executing a C program, we need to link system library
- This can be done automatically at the time of execution

iv**. Executing the program:**

- This is the process of running and testing the program with the sample data
- **At this time there is a possibility show two type of errors given below**

o Logical Error:

- These are the errors, in which conditional and control statements cannot end their match after some sequential execution

o Data error:

- These are the errors, in which the input data given, if not in a proper syntax as specified in input statements.

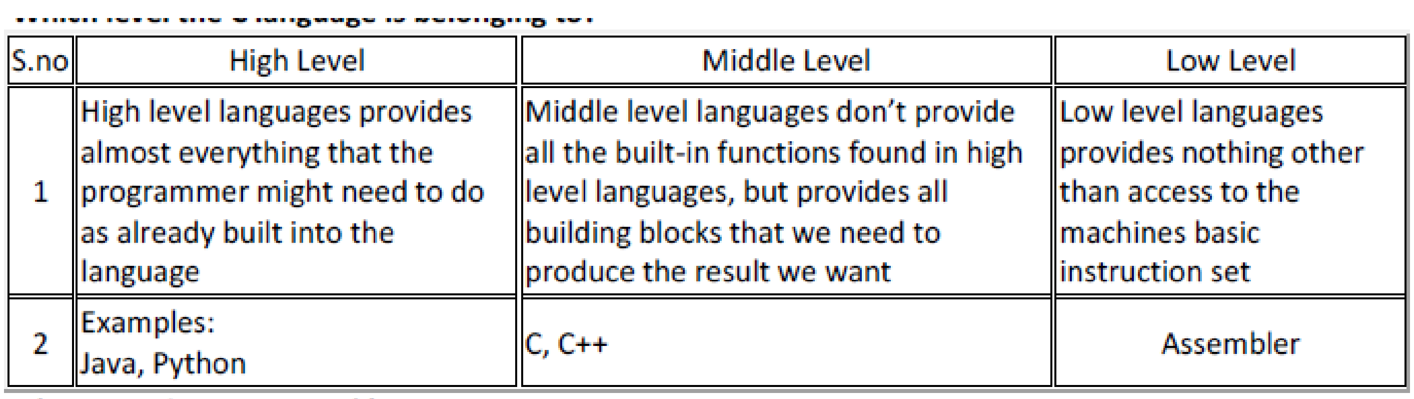
7.4 Uses of C language:

C language is used for developing system applications that forms major portion of operating systems such as Windows, UNIX and Linux. Below are some examples of C being used.

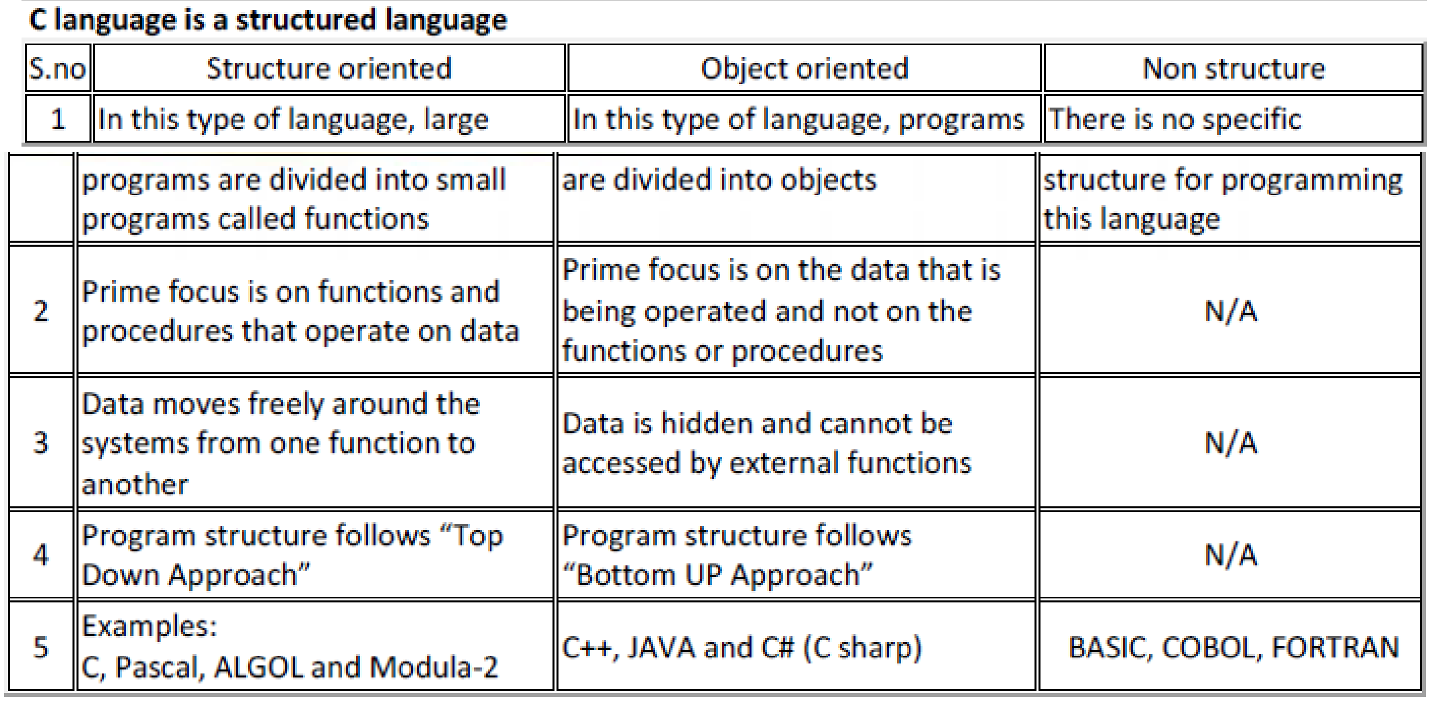
- Database systems

- Graphics packages
- Word processors
- Spread sheets
- Operating system development
- Compilers and Assemblers
- Network drivers
- Interpreters

Which level the C language is belonging to?



S.no	High Level	Middle Level	Low Level
1	High level languages provides almost everything that the programmer might need to do as already built into the language	Middle level languages don't provide all the built-in functions found in high level languages, but provides all building blocks that we need to produce the result we want	Low level languages provides nothing other than access to the machines basic instruction set
2	Examples: Java, Python	C, C++	Assembler



C language is a structured language

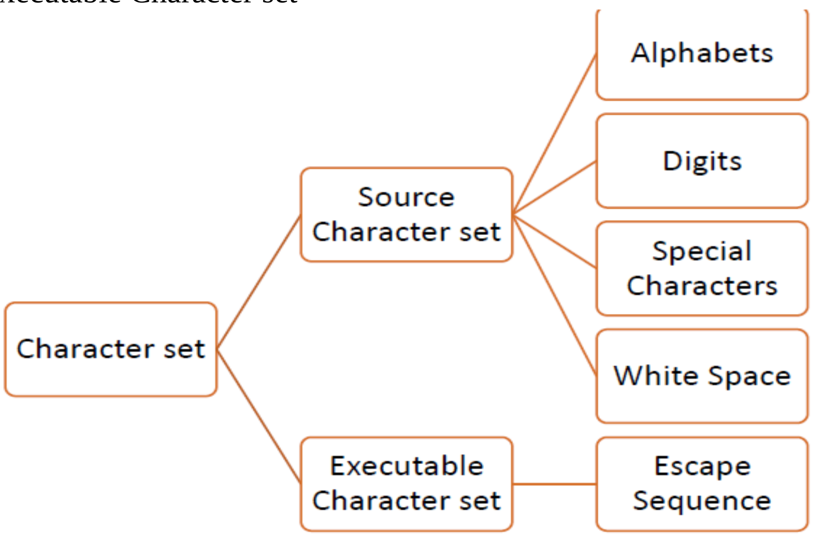
S.no	Structure oriented	Object oriented	Non structure
1	In this type of language, large programs are divided into small programs called functions	In this type of language, programs are divided into objects	There is no specific structure for programming this language
2	Prime focus is on functions and procedures that operate on data	Prime focus is on the data that is being operated and not on the functions or procedures	N/A
3	Data moves freely around the systems from one function to another	Data is hidden and cannot be accessed by external functions	N/A
4	Program structure follows "Top Down Approach"	Program structure follows "Bottom UP Approach"	N/A
5	Examples: C, Pascal, ALGOL and Modula-2	C++, JAVA and C# (C sharp)	BASIC, COBOL, FORTRAN

CHARACTER SET

The character set is the fundamental raw material of any language and they are used to represent information. ***The set of characters used in a language is known as its character set.*** These characters can be represented in the computers.

C programs are basically of two types, namely

1. Source Character set
2. Executable Character set



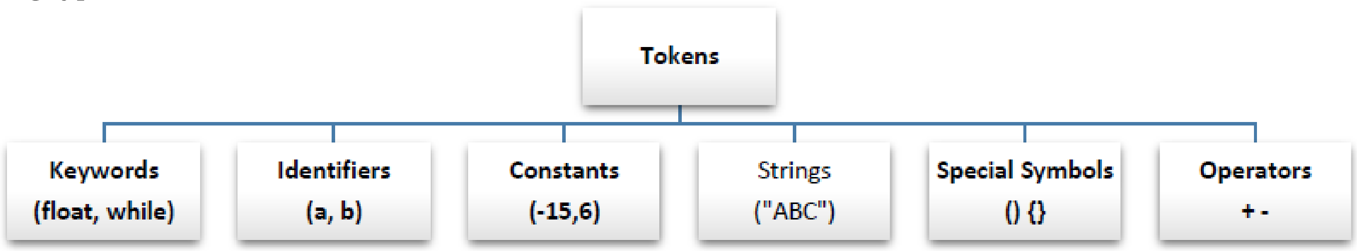
1. Source character set

They are used to construct the statements in the source programs

Alphabets -	A to Z,a to z.
Decimal Digits -	0 to 9
White Spaces -	Blank space, horizontal tab, vertical tab, new line

Special characters - - +,*,, ; ,', /,?,[,{,@,#,%,&,(,<,=,>,},],_,-.
Trigraph Characters:
??= -> # ??(-> [??) ->] ??< -> { ??> -> } ??! -> |

2. Executable character set
\a – Beep
\b – Back space
\t – Horizantal tab
\\ - Back slash
\0 – Null
\n – next line
\v –vertical tab
\f – form feed
\r – carriage return
\' – Single Quote
\"- Double quote
TOKENS
The tokens are usually referred as individual text and punctuation in a passage of text. C tokens has following types.

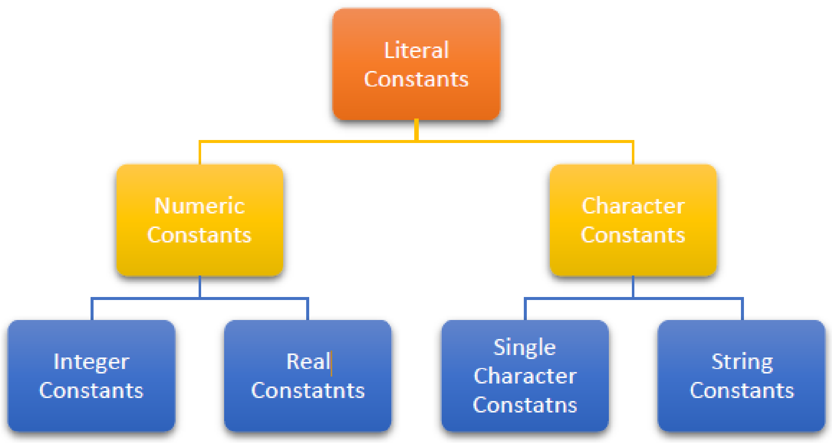


Identifiers and Keywords
In C Language every world is classified into either a keyword or an identifier.
Identifiers:
 Identifiers are names given to various program elements, such as a variable, functions and arrays etc..
 Identifiers are user defined names
 It consists of sequence of letters and digits.
Example of valid Identifiers:
Length, Area, volume, SuM,_Average
Invalid identifiers:
Length of line, Stum: , Year's, 2a.,
Rules for writing Identifiers:
1. It contains letters and digits.
2. '_' can also be used.
3. First character must be a letter or _
4. Contain only 31 characters.
5. No space and special symbols are allowed.
6. It cannot be a keyword.
KEYWORD
 These are reserved words that have standard and predefined meaning in C language.
 It cannot be changed.
 They can't be used as a variable name.
 For utilizing the keyword in a program, no header files are included.

The C support 32 keywords
auto, break, case, char ,const, continue, default, do, double, else , enum, extern, float, for, goto , if, int, register, return, short, signed, sizeof, static, struct, switch, typedef, union, unsigned, void, volatile, while

CONSTANTS
 The item whose values cannot be changed during the execution of program called constants.
 Three types of constants:
o Literal Constants
A literal constant is a value that you put directly in your code for example
o Symbolic Constants
A symbolic constant is a constant that has a name.
Example: #define PI 3.14
o Qualifier Constant:

be The qualifier const can be applied to the declaration of any variable to specify that its value will not changed
 Example: const float pi=3.14;
 Several types of literal constants available.



Numeric constants:

Integer Constants:
 The constants are represented with whole numbers
 They require a minimum of 2 bytes and a maximum of 4 byte of memory

Rules for Constructing Integer Constants
 An integer constant must have at least one digit.
 It must not have a decimal point.
 It can be either positive or negative.
 If no sign precedes an integer constant it is assumed to be positive.
 No commas or blanks are allowed within an integer constant.
 The allowable range for integer constants is -32768to 32767

Valid Examples: 426 +782 -8000 -7605
Invalid example: 2.3, .235, $76, 3,600

Real Constants:
 Real Constants are often known as floating point constants
 Real Constants can be represented in exponential form or floating point form.

Rules for constructing Real Constants:
 A real constant must have at least one digit.
 It must have a decimal point.
 It could be either positive or negative.
 Default sign is positive.
 No commas or blanks are allowed within a real constant
Ex.: +325.34 426.0 -32.76 -48.5792

MANTISSA e EXPONENT..

The exponential form of representation of real constants is usually used if the value of the constant is either too small or too large.

Rules for constructing Exponential form:
 The mantissa part and the exponential part should be separated by a letter e.
 The mantissa part may have a positive or negative sign.
 Default sign of mantissa part is positive.
 The exponent must have at least one digit, which must be a positive or negative integer. Default sign is positive.

Range of real constants expressed in exponential form is -3.4e38 to 3.4e38.
Ex.: +3.2e-5 4.1e8 -0.2e+3 -3.2e-5

Character Constant:

Single Character Constants:

 A character constant is a single alphabet, a single digit or a single special symbol enclosed within single inverted commas (‘’). The maximum length of a character constant can be 1 character

Ex.: 'A' 'I' '5' '=

String Constant:

 String Constants are sequence of characters within double quote marks (“”)

 The string may be combination of all kinds of symbols

Ex.: “Hello”, “India”, “444”, “a”, “”

Example program using const keyword in C:

```
#include <stdio.h>
void main()
{
constint height = 100; /*int constant*/
const float number = 3.14; /*Real constant*/
const char letter = 'A'; /*char constant*/
const char letter_sequence[10] = "ABC"; /*string constant*/
const char backslash_char = '\?'; /*special char cnst*/
printf("value of height : %d \n", height );
printf("value of number : %f \n", number );
printf("value of letter : %c \n", letter );
printf("value of letter_sequence : %s \n", letter_sequence);
printf("value of backslash_char : %c \n", backslash_char);
}
```

C – Variable

 C variable is a named location in a memory where a program can manipulate the data. This location is used to hold the value of the variable.
 The value of the C variable may get change in the program.
 C variable might be belonging to any of the data type like int, float, char etc.

Rules for naming C variable:

1. Variable name must begin with letter or underscore.
2. Variables are case sensitive
3. They can be constructed with digits, letters.
4. No special symbols are allowed other than underscore.
5. sum, height, _value are some examples for variable name

Declaring & initializing C variable:

Memory space is not allocated for a variable while declaration. It happens only on variable definition.
 Variable initialization means assigning a value to the variable.

S.No	Type	Syntax	Example
1	Variable declaration	data_type variable_name;	int x, y, z; char flat, ch;
2	Variable initialization	data_type variable_name = value;	int x = 50, y = 30; char flag = ‘x’, ch=’l’;

There are two types of variables in C program They are,

1. Local variable
2. Global variable

local variable in C:

 The scope of local variables will be within the function only.
 These variables are declared within the function and can’t be accessed outside the function.
 In the below example, m and n ivariables are having scope within the main function only. These are not visible to test function.
 Like wise, a and b variables are having scope within the test function only. These are not visible to main function.

```
#include<stdio.h>
void test();
intmain()
{
int m = 22, n = 44;
// m, n are local variables of main function    /*m and n variables are having scope
within this main function only.These are not visible to test funtion.*/
/* If you try to access a and b in this function,
you will get 'a' undeclared and 'b' undeclarederror */
printf("\nvalues : m = %d and n = %d", m, n);
test();
}
```

global variable in C:

o The scope of global variables will be throughout the program. These variables can be accessed from anywhere in the program.

o This variable is defined outside the main function. So that, this variable is visible to main function and all other sub functions.

```
#include<stdio.h>
void test();
int m = 22, n = 44;
int a = 50, b = 80;
intmain()
{
printf("All variables are accessed from main function");
printf("\nvalues : m= %d : n= %d : a= %d : b= %d",m,n,a,b);
test();
}
void test()
{
printf("\n\nAll variables are accessed from" \ " test function");
printf("\nvalues : m= %d : n= %d : a= %d : b= %d",m,n,a,b);
}
```

Data Types

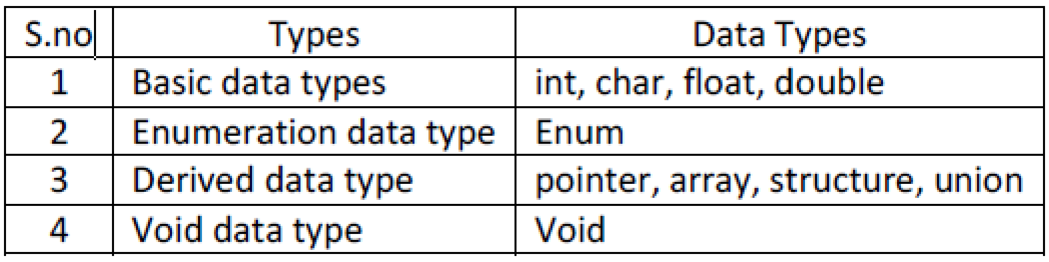
 C data types are defined as the data storage format that a variable can store a data to perform a specific operation.

 Data types are used to define a variable before to use in a program.

 Size of variable, constant and array are determined by data types.

C – data types:

There are four data types in C language. They are,



S.no	Types	Data Types
1	Basic data types	int, char, float, double
2	Enumeration data type	Enum
3	Derived data type	pointer, array, structure, union
4	Void data type	Void

Basic data types in C:

Integer data type:

 Integer data type allows a variable to store numeric values.

 "int" keyword is used to refer integer data type.

 The storage size of int data type is 2 or 4 or 8 byte.

 It varies depend upon the processor in the CPU that we use. If we are using 16 bit processor, 2 byte (16 bit) of memory will be allocated for int data type.

 Like wise, 4 byte (32 bit) of memory for 32 bit processor and 8 byte (64 bit) of memory for 64 bit processor is allocated for int datatype.

 int (2 byte) can store values from -32,768 to +32,767

 int (4 byte) can store values from -2,147,483,648 to +2,147,483,647.

 If you want to use the integer value that crosses the above limit, you can go for "long int" and "long longint" for which the limits are very high.

Character data type:

 Character data type allows a variable to store only one character.

 Storage size of character data type is 1. We can store only one character using character data type.

 "char" keyword is used to refer character data type.

 For example, 'A' can be stored using char datatype. You can't store more than one character using char data type.

 Please refer C – Strings topic to know how to store more than one characters in a variable.

Floating point data type:

Floating point data type consists of 2 types. They are,

1. float
2. double

1. float:

 Float data type allows a variable to store decimal values.
 Storage size of float data type is 4. This also varies depend upon the processor in the CPU as “int” data type.
 We can use up-to 6 digits after decimal using float data type.
 For example, 10.456789 can be stored in a variable using float data type.

2. double:

Double data type is also same as float data type which allows up-to 10 digits after decimal.
 The range for double datatype is from 1E–37 to 1E+37.

sizeof() function in C:

sizeof() function is used to find the memory space allocated for each C data types.

```
#include <stdio.h>
#include <limits.h>
intmain()
{
int a;
char b;
float c;
double d;
printf("Storage size for int data type:%d \n",sizeof(a));
printf("Storage size for char data type:%d \n",sizeof(b));
printf("Storage size for float data type:%d \n",sizeof(c));
printf("Storage size for double data type:%d\n",sizeof(d));
        return 0;
}
```

S.No	C Data types	storage Size	Range
1	Char	1	–127 to 127
2	Int	2	–32,767 to 32,767
3	Float	4	1E–37 to 1E+37 with six digits of precision
4	Double	8	1E–37 to 1E+37 with ten digits of precision
5	long double	10	1E–37 to 1E+37 with ten digits of precision
6	long int	4	–2,147,483,647 to 2,147,483,647
7	short int	2	–32,767 to 32,767
8	unsigned short int	2	0 to 65,535
9	signed short int	2	–32,767 to 32,767
10	long long int	8	–(2power(63) –1) to 2(power)63 –1
11	signed long int	4	–2,147,483,647 to 2,147,483,647
12	unsigned long int	4	0 to 4,294,967,295
13	unsigned long long int	8	2(power)64 –1

Enumeration data type in C:

 Enumeration data type consists of named integer constants as a list.
 It start with 0 (zero) by default and value is incremented by 1 for the sequential identifiers in the **list.**

Enum syntax in C:

enum identifier [optional{ enumerator-list }];
 Enum example in C:
enum month { Jan, Feb, Mar };
or
enum month { Jan = 1, Feb, Mar };
enum month { Jan = 20, Feb, Mar };

Derived data type in C:

 Array, pointer, structure and union are called derived data type in C language.

Void data type in C:

 Void is an empty data type that has no value.
 This can be used in functions and pointers.

OPERATORS AND EXPRESSIONS

 An **Operator** is a symbol that specifies an operation to be performed on the operands

 The data items that operators acts upon are called **Operands**

 An **Operation** indicates an operation to be performed on data that may yield a new value

 An operator can operate on integer, character and floating point numbers

Types of operator:

• Arithmetic Operators
• Relational Operators
• Logical Operators
• Assignment Operators
• Increment and decrement Operators
• Conditional Operators
• Bitwise Operators
• Special Operators

Arithmetic Operators:

These are used to perform mathematical calculations like addition, subtraction, multiplication, division and modulus

S.no	Arithmetic Operators	Operation	Example
1	+	Addition	A+B
2	-	Subtraction	A-B
3	*	multiplication	A*B
4	/	Division	A/B
5	%	Modulus	A%B

Example program for C arithmetic operators:

```
intmain()
{
int a=40,b=20, add,sub,mul,div,mod;
add = a+b;
sub = a-b;
mul = a*b;
div = a/b;
mod = a%b;
printf("Addition of a, b is : %d\n", add);
printf("Subtraction of a, b is : %d\n", sub);
printf("Multiplication of a, b is : %d\n", mul);
printf("Division of a, b is : %d\n", div);
printf("Modulus of a, b is : %d\n", mod);
}
```

Output:

```
Addition of a, b is : 60
Subtraction of a, b is : 20
Multiplication of a, b is : 800
Division of a, b is : 2
Modulus of a, b is : 0
```

Assignment Operators

Assignment operators in C:

o In C programs, values for the variables are assigned using assignment operators.
o For example, if the value “10″ is to be assigned for the variable “sum”, it can be assigned as “sum = 10;”
o Other assignment operators in C language are given below.

Operators		Example	Explanation
Simple assignment operator	=	sum=10	10 is assigned to variable sum
Compound assignment operators	+=	sum+=10	This is same as sum=sum+10
	-=	sum-=10	This is same as sum = sum-10
	*=	sum*=10	This is same as sum = sum*10
	/+	sum/=10	This is same as sum = sum/10
	%=	sum%=10	This is same as sum = sum%10
	&=	sum&=10	This is same as sum = sum&10
	^=	sum^=10	This is same as sum = sum^10

Example program for C assignment operators:

```
# include <stdio.h>
intmain()
{
int Total=0,i;
for(i=0;i<10;i++)
{
Total+=i; // This is same as Total = Toatal+i
}
printf("Total = %d", Total);
}
```

Output:

```
Total = 45
```

Relational Operators

Relational operators in C:

o Relational operators are used to find the relation between two variables. i.e. to compare the values of two variables in a C program.

S.no	Operators	Example	Description
1	>	x > y	x is greater than y
2	<	x < y	x is less than y
3	>=	x >= y	x is greater than or equal to y
4	<=	x <= y	x is less than or equal to y
5	==	x == y	x is equal to y
6	!=	x != y	x is not equal to y

Example program for relational operators in C:

```
#include <stdio.h>
intmain()
{
int m=40,n=20;
if (m == n)
{
printf("m and n are equal");
}
else
{
printf("m and n are not equal");
}
}
```

Output:

```
m and n are not equal
```

Logical Operators

Logical operators in C:

o These operators are used to perform logical operations on the given expressions.

o There are 3 logical operators in C language. They are, logical AND (&&), logical OR (||) and logical NOT (!).

S.no	Operators	Name	Example	Description
1	&&	logical AND	(x>5)&&(y<5)	It returns true when both conditions are true
2	||	logical OR	(x>=10)||(y>=10)	It returns true when at-least one of the condition is true
3	!	logical NOT	!((x>5)&&(y<5))	It reverses the state of the operand "((x>5) && (y<5))" If "((x>5) && (y<5))" is true, logical NOT operator makes it false

Example program for logical operators in C:

```
#include <stdio.h>
intmain()
{
int m=40,n=20;
int o=20,p=30;
if (m>n &&m !=0)
{
```

```
printf("&& Operator : Both conditions are true\n");
}
if (o>p || p!=20)
{
printf("|| Operator : Only one condition is true\n");
}
if (!(m>n && m !=0))
{
printf("! Operator : Both conditions are true\n");
}
else
{
printf("! Operator : Both conditions are true. " \
"But, status is inverted as false\n");
}
}
```

Output:

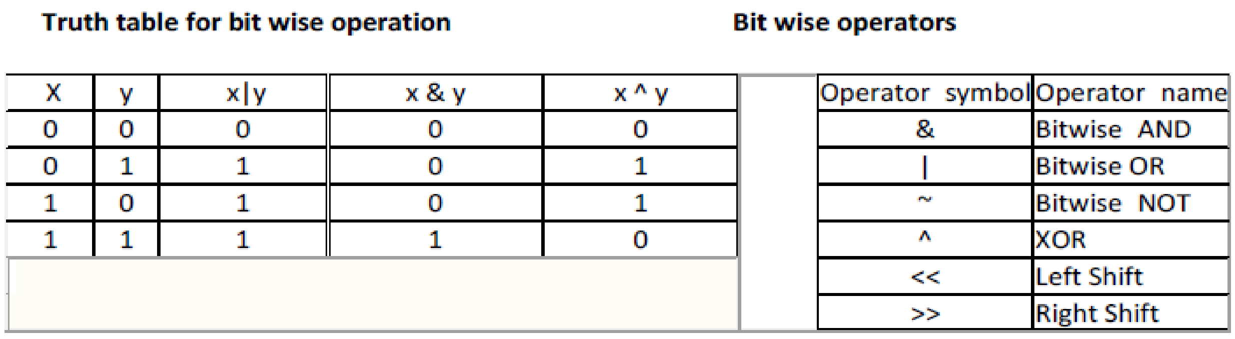
```
&&Operator : Both conditions are true
|| Operator : Only one condition is true
! Operator : Both conditions are true. But, status is inverted as false
```

Bit wise Operators

Bit wise operators in C:

o These operators are used to perform bit operations. Decimal values are converted into binary values which are the sequence of bits and bit wise operators work on these bits.

o Bit wise operators in C language are & (bitwise AND), | (bitwise OR), ~ (bitwise OR), ^ (XOR), << (left shift) and >> (right shift).



Truth table for bit wise operation

X	y	x|y	x & y	x ^ y
0	0	0	0	0
0	1	1	0	1
1	0	1	0	1
1	1	1	1	0

Bit wise operators

Operator symbol	Operator name
&	Bitwise AND
|	Bitwise OR
~	Bitwise NOT
^	XOR
<<	Left Shift
>>	Right Shift

Example program for bit wise operators in C:

```
#include <stdio.h>
intmain()
{
int m=40,n=80,AND_opr,OR_opr,XOR_opr,NOT_opr ;
AND_opr = (m&n);
OR_opr = (m|n);
NOT_opr = (~m);
XOR_opr = (m^n);
printf("AND_opr value = %d\n",AND_opr );
printf("OR_opr value = %d\n",OR_opr );
printf("NOT_opr value = %d\n",NOT_opr );
printf("XOR_opr value = %d\n",XOR_opr );
printf("left_shift value = %d\n", m << 1);
printf("right_shift value = %d\n", m >> 1);
}
```

Output:

```
AND_opr value = 0
OR_opr value = 120
NOT_opr value = -41
XOR_opr value = 120
left_shift value = 80
right_shift value = 20
```

Conditional Operators

Conditional or ternary operators in C:

o Conditional operators return one value if condition is true and returns another value is condition is false.

o This operator is also called as ternary operator.

Syntax : (Condition? true_value: false_value);
Example : (A > 100 ? 0 : 1);

o In above example, if A is greater than 100, 0 is returned else 1 is returned. This is equal to if else conditional statements.

Example program for conditional/ternary operators in C:

```
#include <stdio.h>
intmain()
{
int x=1, y ;
y = ( x ==1 ? 2 : 0 ) ;
printf("x value is %d\n", x);
printf("y value is %d", y);
}
```

Output:

```
x value is 1
y value is 2
```

Increment/decrement Operators

Increment operators are used to increase the value of the variable by one and decrement operators are used to decrease the value of the variable by one in C programs.

Syntax:

Increment operator :

++var_name;(or) var_name++;

Decrement operator :

– - var_name; (or) var_name – -;

Example:

Increment operator :

++i;i++;

Decrement operator: – - i ;i – - ;

Example program for increment operators in C:

```
//Example for increment operators
#include <stdio.h>
intmain()
{
inti=1;
while(i<10)
{
printf("%d ",i);
i++;
}
}
```

Output:

```
1 2 3 4 5 6 7 8 9
```

Example program for decrement operators in C:

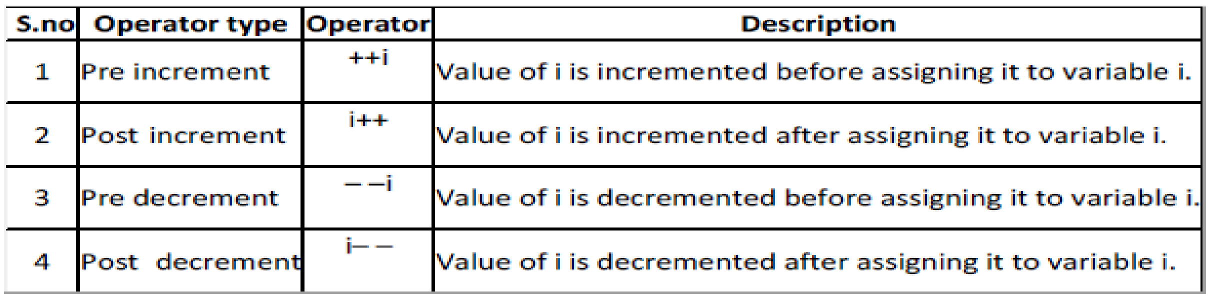
```
//Example for decrement operators
#include <stdio.h>
intmain()
{
inti=20;
while(i>10)
{
printf("%d ",i);
i--;
}
}
```

Output:

```
20 19 18 17 16 15 14 13 12 11
```

Difference between pre/post increment & decrement operators in C:

o Below table will explain the difference between pre/post increment and decrement operators in C.



S.no	Operator type	Operator	Description
1	Pre increment	++i	Value of i is incremented before assigning it to variable i.
2	Post increment	i++	Value of i is incremented after assigning it to variable i.
3	Pre decrement	– –i	Value of i is decremented before assigning it to variable i.
4	Post decrement	i– –	Value of i is decremented after assigning it to variable i.

Example program for pre – increment operators in C:

```
//Example for increment operators
#include <stdio.h>
intmain()
{
inti=0;
while(++i< 5 )
{
printf("%d ",i);
}
return 0;
}
```

Output:

```
        1 2 3 4
```

Step 1 : In above program, value of "i" is incremented from 0 to 1 using pre-increment operator.

o Step 2 : This incremented value "1″ is compared with 5 in while expression.

o Step 3 : Then, this incremented value "1″ is assigned to the variable "i".

o Above 3 steps are continued until while expression becomes false and output is displayed as "1 2 3 4″.

Example program for post – increment operators in C:

```
#include <stdio.h>
intmain()
{
inti=0;
while(i++ < 5 )
{
printf("%d ",i);
}
return 0;
}
```

Output:

```
1 2 3 4 5
```

Step 1 : In this program, value of i "0″ is compared with 5 in while expression.

Step 2 : Then, value of "i" is incremented from 0 to 1 using post-increment operator.

Step 3 : Then, this incremented value "1″ is assigned to the variable "i".

o Above 3 steps are continued until while expression becomes false and output is displayed as "1 2 3 4 5″.

Example program for pre - decrement operators in C:

```
#include <stdio.h>
intmain()
{
inti=10;
while(--i> 5 )
{
printf("%d ",i);
}
return 0;
}
```

Output:

```
9 8 7 6
```

Step 1 : In above program, value of "i" is decremented from 10 to 9 using predecrement operator.

o Step 2 : This decremented value "9″ is compared with 5 in while expression.

o Step 3 : Then, this decremented value "9″ is assigned to the variable "i".

o Above 3 steps are continued until while expression becomes false and output is displayed as "9 8 7 6″.

Special Operators in C:

o Below are some of special operators that C language offers.

S.no	Operators	Description
1	&	This is used to get the address of the variable. Example : &a will give address of a.
2	*	This is used as pointer to a variable. Example : * a where, * is pointer to the variable a.
3	Sizeof ()	This gives the size of the variable. Example : size of (char) will give us 1.

```
#include <stdio.h>
intmain()
{
int *ptr, q;
q = 50;
/* address of q is assigned to ptr */
ptr = &q;
/* display q's value using ptr variable */
printf("%d", *ptr);
return 0;
}
```

Output:

```
50
```

MANAGING INPUT AND OUTPUT OPERATIONS

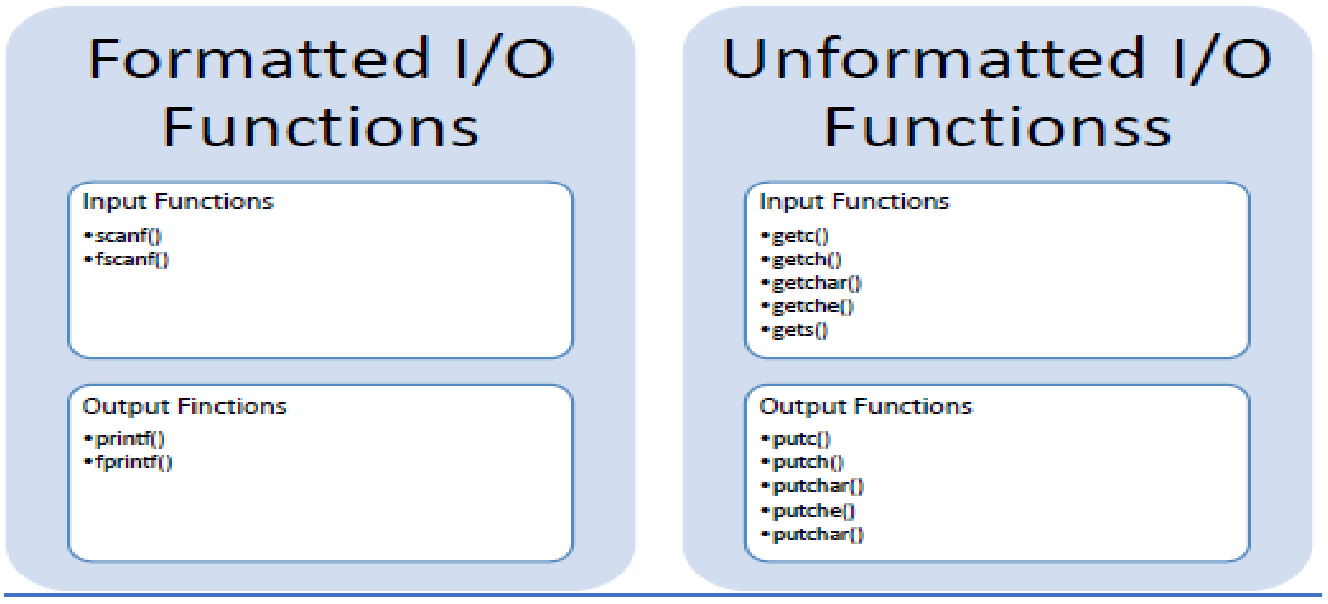
 Reading data from input device, processing it, and displaying the result on the screen are the three task of any program

 We have two methods for providing data to the program

1. Assigning the data to the variable in a program

2. By using the I/O functions

Two types of Input / Output functions:



The printf() function:

 The output data or result of an operation can be displayed from the computer to a standarad output device ie., Monitor

 The function is used to output any combination of data

Syntax:
printf("format string", list of variables) ;

The format string can contain:

a) Characters that are simply printed as they are

b) b) Conversion specifications that begin with a % sign

c) Escape sequences that begin with a \ sign

Example:
main( )

```
{
intavg = 346 ;
float per = 69.2 ;
printf( "Average = %d\nPercentage = %f", avg, per ) ;
}
```

Output:

```
Average = 346
Percentage = 69.200000
```

Format Specifications:

Data Type		Format Specifier
Integer	short signed	%d or %I
	short unsigned	%u
	long singed	%ld
	long unsigned	%lu
	unsigned hexadecimal	%x
	unsigned octal	%o
Real	Float	%f
	Double	%lf
Character	signed character	%c
	unsigned character	%c
String		%s

Escape Sequences :

\a – Beep \b – Back space \t – Horizantal tab
\\ - Back slash \n – next line \v –vertical tab
\f – form feed \r – carriage return \' – Single Quote
\"- Double quote

The Scanf() Function:

 The scanf() function is usd to read information the standarad input device

o It is used for runtime assignment of variables

o This function is used to enter any combination of input

Syntax:

scanf("format string", list of addresses of variables) ;

Example:

scanf("%d %f %c", &c, &a, &ch) ;

 & denotes the address of the variable. The values received from keyboard must be dropped into variables corresponding to these addresses

The sprintf() function:

 This function writes the output to an array of characters

Example:

```
inti = 10 ;
char ch = 'A' ;
float a = 3.14 ;sprintf ( str, "%d %c %f", i, ch, a ) ;
```

UNFORMATTED I/O FUNCTIONS

 These statements are used to I/O a single / group of characters from the I/O Device

 Here the user can't specify the type of data that is going to be Input / Ouptput

UNFORMATTED OUTPUT FUNCIONS:

1) The getch () function:

 getch() accepts only single character from keyboard.

 The character entered through getch() is not displayed in the screen (monitor).

Syntax:

variable_name = getch();

2) The getche() function:

 getche() also accepts only single character, but unlike getch(), getche() displays the entered character in the screen.

Syntax:

variable_name = getche();

3) The getchar() function:

- getchar() accepts one character type data from the keyboard.
- It requires ***Enter key*** to be typed following the character that you typed

Example:

```
void main( )
{
char ch ;
printf( "\nPress any key to continue" ) ;
getch( ) ; /* will not echo the character */
printf( "\nType any character" ) ;
ch = getche( ) ; /* will echo the character typed */
printf( "\nType any character" ) ;
getchar( ) ; /* will echo character, must be followed by enter key */
}
```

Output:

```
Press any key to continue
Type any character B
Type any character W
W
```

4) **The gets() function:**

- It accepts any line of string including spaces from the standard Input device (keyboard).
- It stops reading character from keyboard only when the enter key is pressed.

Syntax:

gets(variable_name);

Example:

```
char ch[20];
gets(ch);
```

Unformatted Output Functions:

1) The putch() Function:

`
- putch displays any alphanumeric characters to the standard output device.
- It displays only one character at a time.

Syntax:

putch(variable_name);

Example:

```
char z[20]=”welcome”;
putch(z);
```

2) The putchar() function:

- putchar displays one character at a time to the Monitor.

Syntax:

putchar(variable_name);

Example:

```
char z[20]=”welcome”;
putchar(z);
```

3) The puts() Function

- puts displays a single / paragraph of text to the standard output device.

Syntax:

Puts(variable_name);

Example:

```
#include<stdio.h>
#include<conio.h>
void main()
{
char a[20];
gets(a);
puts(a);
getch();
}
```

Output:

Abcdefgh
Abcdefgh

CONTROL STATEMENTS

 Control Statement are program statements that are cause a jump of control from one part of program to another part of program

 These statements are classified into two types:
- o Branching Statements
- o Looping Statements

Branching Statements:

 In decision making statements, group of statements are executed when condition is true. If condition is false, then else part statements are executed.

 There are 3 types of decision making control statements in C language. They are,
- o if statements
- o if else statements
- o nested if statements
- o else if statements
- o Switch statements

If Statement:

 An if statement consists of a boolean expression followed by one or more statements.

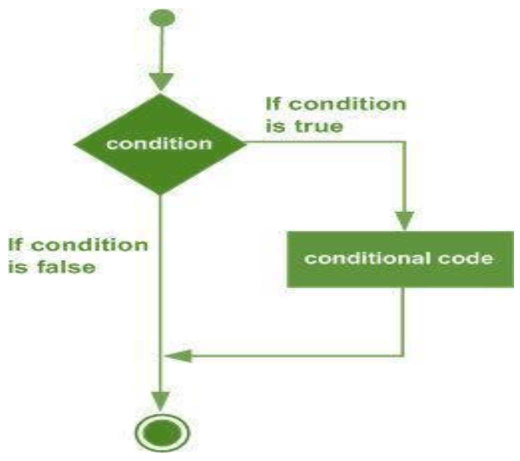
Syntax:

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

 If the condition is true, then the block of code inside the if statement will be executed.

 If the condition is false, then the first set of code after the end of the if statement(after the closing curly brace) will be executed.

Flow Diagram:



Example:

```
#include <stdio.h>
void main ()
{
int a = 10;
if( a< 20 )
{
printf("a is less than 20\n" );
}
printf("value of a is : %d\n", a);
            }
```

Output:

```
a is less than 20;
value of a is : 10
```

if..else statements:

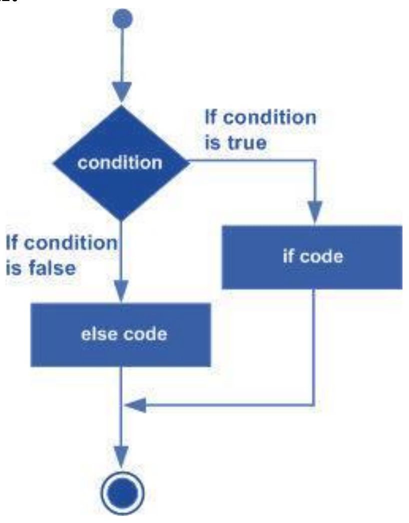
 In if else control statement, group of statements are executed when condition is true.

 If condition is false, then else part statements are executed.

Syntax:

```
if(condition)
{
True Statements
}
          else
{
False statements
}
```

Flow Diagram:



Example:

```
#include <stdio.h>
void main ()
{
int a = 100;
if( a< 20 )
{
printf("a is less than 20\n" );
}
else
{
printf("a is not less than 20\n" );
}
printf("value of a is : %d\n", a);
}
```

Output:

```
a is not less than 20;
value of a is : 100
```

else if ladder / else if statements:

 An **if** statement can be followed by an optional **else if...else** statement, which is very useful to test various conditions using single if...else if statement.

When using if , else if , else statements there are few points to keep in mind:

o An if can have zero or one else's and it must come after any else if's.

o An if can have zero to many else if's and they must come before the else.

o Once an else if succeeds, none of the remaining else if's or else's will be tested.

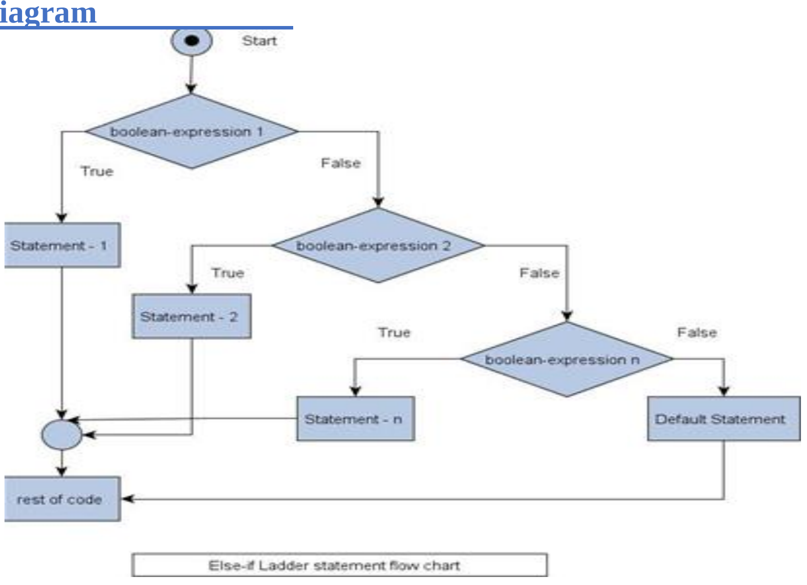
Syntax:

```
if(Condition1)
{
Statements 1;
}
else if(Condition 2)
{
```

```
Statements 2;
}
else if(Condition 3)
{
Statements 4;
}
else
{
Else statements;

}
```

FlowDiagram



Example:
```
#in
clude
<stdio.h>void
main()

{
            int a =
             100;if(
             a==10)
            {
               printf("Valueofais10\n");
            }
            elseif(a ==20)
            {
                printf("Valueofais20\n");
            }
            elseif(a ==30)
            {
                printf("Valueofais30\n");
            }
            else
            {
                printf("Noneofthevaluesismatching\n");
            }
            printf("Exactvalueofais: %d\n",a);
}
```

```
Noneof thevaluesismatching

Exact valueofais:100
```

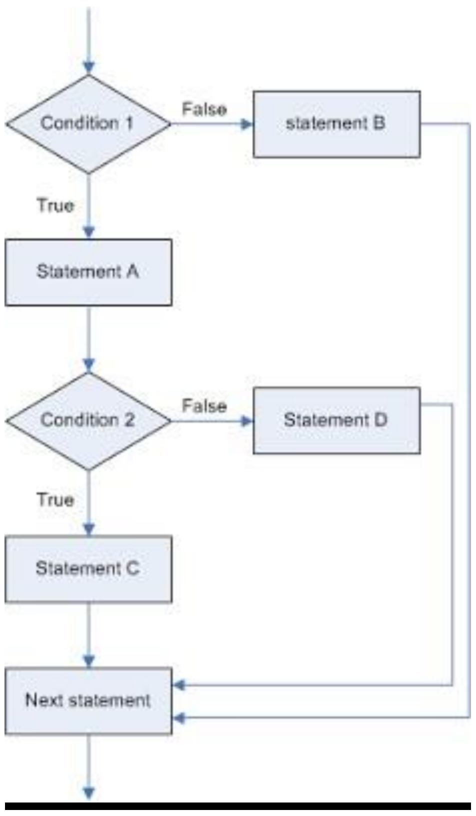
Nested ifStatements:

- An if .. else statement is placed inside another if..else statements, this is known asnested if..elsestatements
- Theseareusedwhenseriesof decisionsareinvolved

Syntax:

```
if(Condition 1)
{
  /* Executes when the condition 1 is true
  */if(Condition2)
  {
    /*Executeswhenthecondition2istrue*/
  }
}
else
{
/*Executeswhenthecondition1isfale*/
}
```

FlowDiagram:



Example:#in
clude
<stdio.h>void
main()

```
{
int a = 100;
int b = 200;
if( a == 100 )
{
if( b == 200 )
{
printf("Value of a is 100 and b is 200\n" );
}
}
printf("Exact value of a is : %d\n", a );
printf("Exact value of b is : %d\n", b );
}
```

Output:

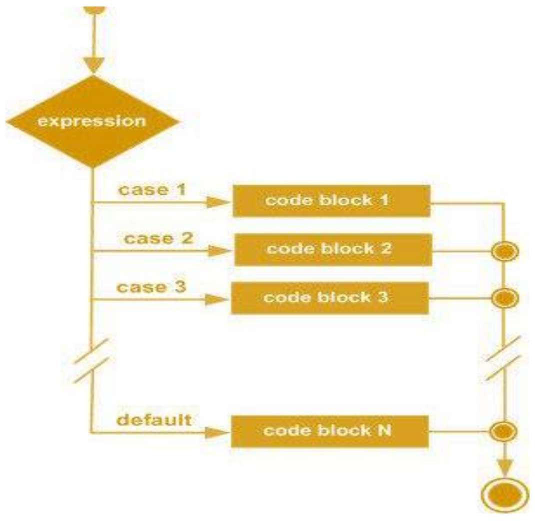
```
Value of a is 100 and b is 200
Exact value of a is : 100
Exact value of b is : 200
```

Switch Statements:

 Switch statements is a multiway branching statements based on the value of an expression

 The control is transferred to one of the many possible points
 If no match is there, then the default block is executed
 Every case statements should be terminated with a break statements

Syntax: switch (expression) { case label1:
statements; break;
case label2:
statements; break;
default:
statements;
}



Example:

```
#include <stdio.h>
void main ()
{
char grade = 'B';
switch(grade)
{
case 'A' :
printf("Excellent!\n" );
break;
case 'B' :
case 'C' :
printf("Well done\n" );
break;
case 'D' :
printf("You passed\n" );
break;
case 'F' :
printf("Better try again\n" );
break;
default :
printf("Invalid grade\n" );
}
printf("Your grade is %c\n", grade );
}
```

Output:

```
Well done
Your grade is B
```

LOOPING STATEMENTS

 Loop control statements in C are used to perform looping operations until the given condition is true. Control comes out of the loop statements once condition becomes false.

 There are 3 types
o for loop
o while loop
o do while loop

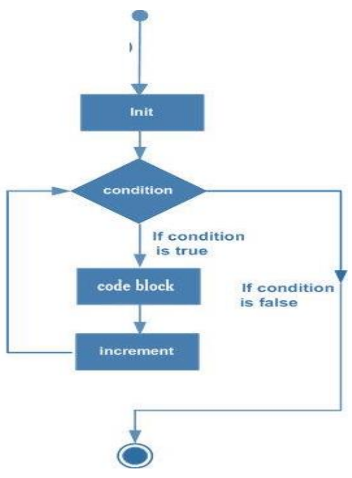
for loop:
 A **for** loop is a repetition control structure that allows you to efficiently write a loop that needs to execute a specific number of times.

Syntax:
for (init; condition; increment)
{
statement(s);
}
1. The **init**step is executed first, and only once. This step allows you to declare and initialize any loop control variables.

2. Next, the **condition** is evaluated. If it is true, the body of the loop is executed. If it is false, the body of the loop does not execute and flow of control jumps to the next statement just after the for loop.

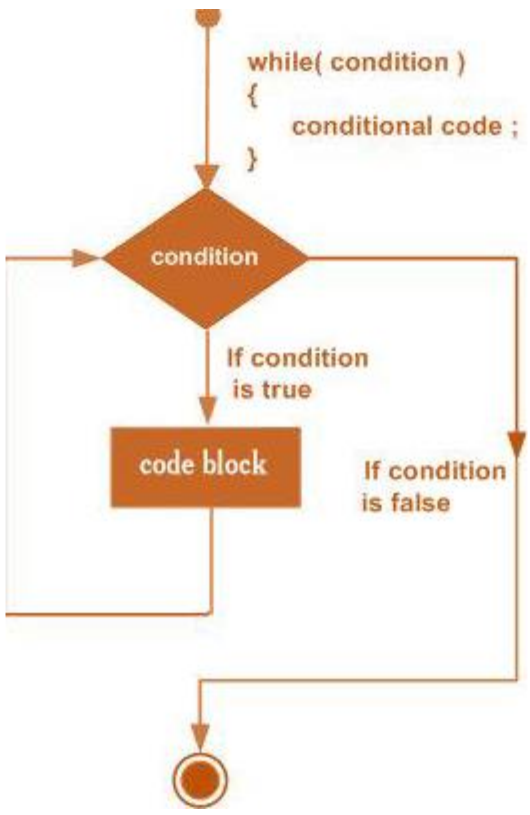
3. After the body of the for loop executes, the flow of control jumps back up to the **increment** statement. This statement allows you to update any loop control variables.

4. The condition is now evaluated again. If it is true, the loop executes and the process repeats itself (body of loop, then increment step, and then again condition). After the condition becomes false, the for loop terminates.
Flow Diagram:



Example:
```
#include <stdio.h>
void main ()
{
int a;
for( a = 10; a < 20; a = a + 1 )
{
printf("%d", a);
}
}
```
Output:
```
10 11 12 13 14 15 16 17 18 19
```
While Loop:
 A while loop statement repeatedly executes a target statement as long as a given condition is true.
 Here the condition is checked first, so it is also called as entry control statements

Syntax:
while(condition)
{
statement(s);
}
Flow Diagram:



Example:

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

Output:

```
10 11 12 13 14 15 16 17 18 19
```

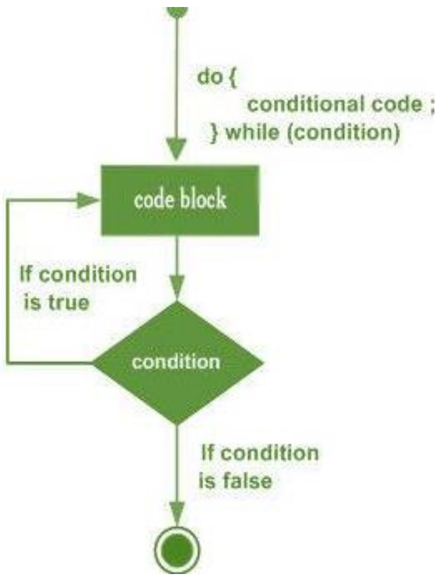
Do…while Loop:

- Unlike **for** and **while** loops, which test the loop condition at the top of the loop, the**do...while** loop in C programming language checks its condition at the bottom of the loop.
- A **do...while** loop is similar to a while loop, except that a do...while loop is guaranteed to execute at least one time.
- This is also called exit control statement

Syntax:

```
do
{
statement(s);
}while( condition );
```

Flow Control



Example:

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

```
void main ()
{
int a = 10;
do
{
printf("%d", a);
a = a + 1;
}while( a < 20 );
}
```

Output:

```
10 11 12 13 14 15 16 17 18 19
```

Break Statement:

The **break** statement has the following two usages:

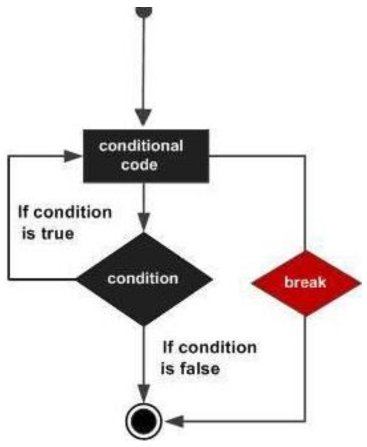
 When the **break** statement is encountered inside a loop, the loop is immediately terminated and program control resumes at the next statement following the loop.

 It can be used to terminate a case in the **switch** statement

Syntax:

```
break;
```

Flow Diagram:



Example:

```
#include <stdio.h>
void main ()
{
int a = 10;
while( a< 20 )
{
printf("%d", a);
a++;
if( a> 15)
{
break;
}
}
}
```

Output:

```
10 11 12 13 14 15
```

Continue Statement:

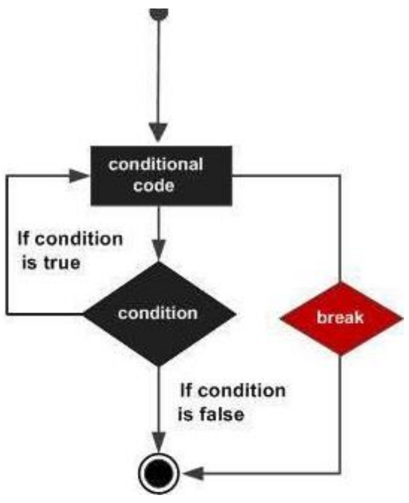
 The continue statement works somewhat like the break statement.

 Instead of forcing termination of loop, however, continue forces the next iteration of the loop to take place, skipping any code in between.

Syntax:

continue;

Flow Diagram:



Example:

```
#include <stdio.h>
void main ()
{
int a = 10;
while( a< 20 )
{
printf("%d", a);
a++;
if( a> 15)
{
continue;
}
}
}
```

Output:

```
10 11 12 13 14 16 17 18 19
```

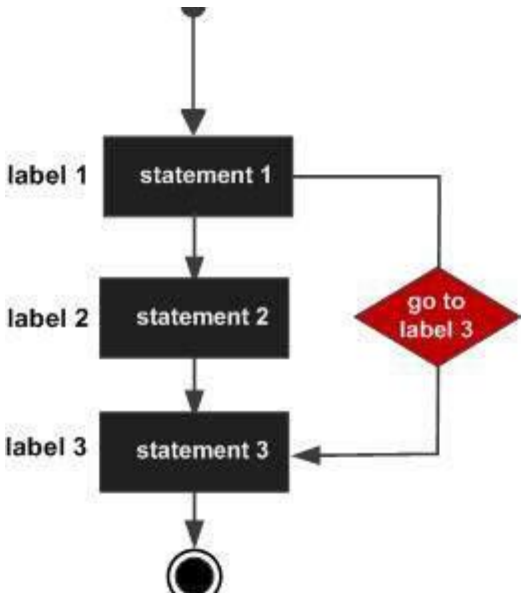
Goto Statement:

A goto statement in C programming language provides an unconditional jump from the goto to a labeled statement in the same function.

Syntax:

```
goto label;
….
….
label: statement;
```

Flow Diagram:



Example:

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

```
void main()
{
int a = 10;
LOOP: do
{
if( a == 15)
{
a = a + 1;
goto LOOP;
}
printf(" %d", a);
a++;
}while( a < 20 );
}
```

Output:

```
10 11 12 13 14 16 17 18 19
```

UNIT -III

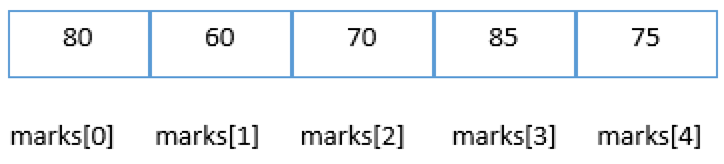
ARRAYS AND STRINGS	9
Arrays - Initialization - Declaration - One dimensional and Two dimensional arrays. String- String operations - String Arrays. Simple programs –Bubble Sort – Linear Search -Matrix Operations.	

1 Introduction

The C Programming language generally uses the variables which are nothing but a name given to a storage area that our programs can manipulate. Each variable in C has a specific type, which determines the size and layout of the variable's memory; the range of values that can be stored within that memory and the limitations are it can’t store more than one value at a time. The limitation of variables could be avoided by using the arrays.

Arrays.

An array is a collection of data that holds fixed number of values of same type. For example: if you want to store marks of 100 students, you can create an array for it. intmark[100]



Initialization of Array

Array Declaration.

To declare an array in C, a programmer specifies the type of the elements and the number ofelements required by an array as follows,

Syntax

<type><array Name> [array Size];

This is called a single-dimensional array. The **arraySize**must be an integer constant greater thanzero and type can be any valid C data type. For example, to declare a 10-element array called balance oftype double, use this statement:***double balance[10];***

Accessing Array Elements

An element is accessed by indexing the array name. This is done by placing the index of theelement within square brackets after the name of the arrayFor example:***double salary = balance[9];***

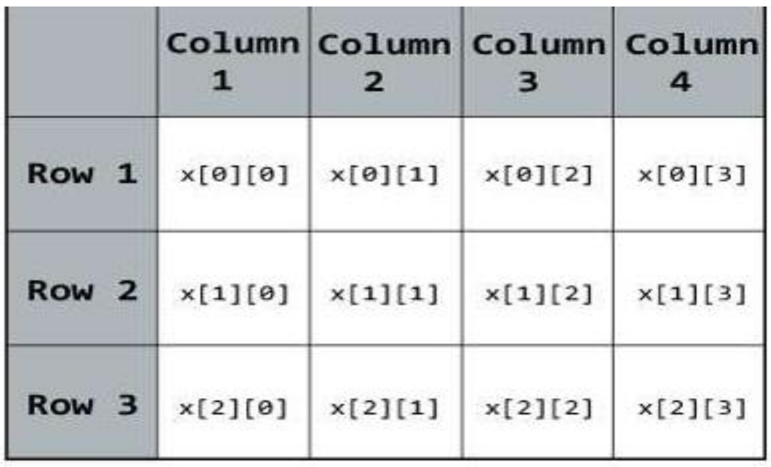
```
#include <stdio.h>
int main ()
{
intn[ 10 ]; /* n is an array of 10 integers */
inti,j;
/* initialize elements of array n to 0 */
for ( i = 0; i< 10; i++ )
{
n[i ] = i + 100; /* set element at location i to i + 100 */
}
/* output each array element's value */
for (j = 0; j < 10; j++ )
{
printf("Element[%d] = %d\n", j, n[j] );
}
return 0;
}
```

result:

```
Element[0] = 100
Element[1] = 101
Element[2] = 102
Element[3] = 103
Element[4] = 104
Element[5] = 105
Element[6] = 106
Element[7] = 107
Element[8] = 108
Element[9] = 109
```

Arrays are of two types:

One-dimensional arrays
intmark[5] = {19, 10, 8, 17, 9};
Arrays have 0 as the first index not 1. In this example, mark[0]
If the size of an array is n, to access the last element, (n-1) index is used. In this **example,mark[4]**
Multidimensional arrays
In C programming, you can create an array of
arrays known as multidimensional array. For**example,float x[3][4];**
Here, x is a two-dimensional (2d) array. Thearray can hold 12 elements. You can think the arrayas table with 3 row and each row has 4 column.
Similarly, you can declare a three-dimensional (3d) array. For example,float y[2][4][3];
Here,The array y can hold 24 elements.



	Column 1	Column 2	Column 3	Column 4
Row 1	x[0][0]	x[0][1]	x[0][2]	x[0][3]
Row 2	x[1][0]	x[1][1]	x[1][2]	x[1][3]
Row 3	x[2][0]	x[2][1]	x[2][2]	x[2][3]

Two-Dimensional Arrays
The simplest form of the **multidimensional** array is the **two-dimensional** array. A twodimensionalarray is, in essence, a list of one-dimensional arrays. To declare a two-dimensional integerarray of size x, y you would write something as follows:
type arrayName [x][y];
EXAMPLE

```
inta[3][4] = {
{0, 1, 2, 3} ,/* initializers for row indexed by 0 */
{4, 5, 6, 7} ,/* initializers for row indexed by 1 */
{8, 9, 10, 11} /* initializers for row indexed by 2 */
};
```

```
#include <stdio.h>
int main ()
{
/* an array with 5 rows and 2 columns*/
inta[5][2] = { {0,0}, {1,2}, {2,4}, {3,6},{4,8}};
inti, j;
/* output each array element's value */
for ( i = 0; i< 5; i++ )
{
for ( j = 0; j < 2; j++ )
{
printf("a[%d][%d] = %d\n", i,j, a[i][j] );
}}
```

result:

```
a[0][0]: 0
a[0][1]: 0
a[1][0]: 1
a[1][1]: 2
a[2][0]: 2
a[2][1]: 4
a[3][0]: 3
a[3][1]: 6
a[4][0]: 4
a[4][1]: 8
```

Passing array to function

While passing arrays as arguments to the function, only the name of the array ispassed. C program to pass an array containing age of person to a function. This functionshould find average age and display the average age in main function.
SYNTAX:

void myFunction(int *param)
{...
}

OR

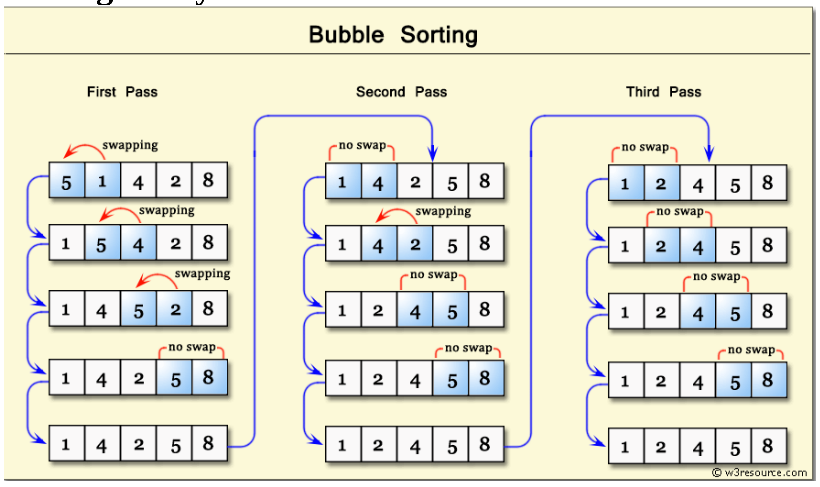
void myFunction(intparam[10])
{...
}

OR

void myFunction(intparam[])
{...
}

```
#include <stdio.h>
float average(float age[]);
intmain()
{
float avg, age[] = { 23.4, 55, 22.6, 3, 40.5, 18 };
avg = average(age); /* Only name of array is passed as argument. */
printf("Average age=%.2f", avg);
return 0;
}
float average(float age[])
{
inti;
float avg, sum = 0.0;
for (i = 0; i< 6; ++i) {
sum += age[i];
}
avg = (sum / 6);
return avg;
}
```

Sorting Array



This C Program sorts the numbers in ascending order using bubble sort. Bubble sort is a simplesorting algorithm that works by repeatedly stepping through the list to be sorted, comparingeach pair of adjacent items and swapping them if they are in the wrong order. Here we need tosort a number in ascending order.

```
void main()
{
int array[maxsize];
inti, j, num, temp;
printf("enter the value of num \n");
scanf("%d", &num);
printf("enter the elements one by one \n");
for (i = 0; i<num; i++)
{
scanf("%d", &array[i]);
}
printf("input array is \n");
for (i = 0; i<num; i++)
{
printf("%d\n", array[i]);
}
/* bubble sorting begins */
for (i = 0; i<num; i++)
{
for (j = 0; j < (num - i - 1); j++)
{
if (array[j] >array[j + 1])
{
temp = array[j];
array[j] = array[j + 1];
array[j + 1] = temp;
}
}
}
printf("sorted array is...\n");
for (i = 0; i<num; i++)
{
printf("%d\n", array[i]);
}
}
```

Searching

Linear search in C programming: The following code implements linear search (Searchingalgorithm) which is used to find whether a given number is present in an array and if it ispresent then at what location it occurs. It is also known as sequential search

```
#include <stdio.h>
intmain()
{
intarray[100], search, c, n;
printf("Enter the number of elements in array\n");
scanf("%d",&n);
printf("Enter %d integer(s)\n", n);
for (c = 0; c < n; c++)
scanf("%d", &array[c]);
printf("Enter the number to search\n");
scanf("%d", &search);
for (c = 0; c < n; c++)
{
if (array[c] == search) /* if required element found */
{
printf("%d is present at location %d.\n", search, c+1);
break;
}
}
if (c == n)
printf("%d is not present in array.\n", search);
return 0;
}
```

```
Enter the number of elements in array 5
Enter 5 numbers
12 23 35 71 50
Enter the number to search 71
71 is present at the location 4
```

FUNCTIONS AND POINTERS	9
Function - Definition of function - Declaration of function - Pass by value - Pass by reference - Recursion - Pointers - Definition - Initialization - Pointers arithmetic - Pointers and arrays- Example Problems.	

Functions in C.

Introduction

Function is a group of statements that together perform a task. Every C program has at least onefunction, which is main(), and all the most trivial programs can define additional functions.

You can divide up your code into separate functions. How you divide up your code among differentfunctions is up to you, but logically the division usually is so each function performs a specific task.

A function declaration tells the compiler about a function's name, return type, and parameters. Afunction definition provides the actual body of the function.

Defining a Function.

A function definition in C programming language consists of a function header and a functionbody. General form of a function definition in C programming language is as follows:

```
return_typefunction_name( parameter list )
{
body of the function
}
```

Return Type: A function may return a value. The return_type is the data type of the value thefunction returns. Some functions perform the desired operations without returning a value. In this case, the return type is the keyword void.

Function Name: This is the actual name of the function. The function name and the parameter list together constitute the function signature.

Parameters: A parameter is like a placeholder. When a function is invoked, you pass a value tothe parameter. This value is referred to as actual parameter or argument. Theparameter list refers to the type, order, and number of the parameters of a function.Parameters are optional; that is, a function may contain no parameters.

Function Body: The function body contains a collection of statements that define what thefunction does.

```
/* function returning the max between two numbers */
intmax(int num1, int num2)
{
/* local variable declaration */
int result;
if (num1 > num2)
result = num1;
else
result = num2;
return result;
}
```

Function Declarations

A function declaration tells the compiler about a function name and how to call the function.The actual body of the function can be defined separately.A function declaration has the following parts:

```
return_typefunction_name( parameter list );

intmax(int num1, int num2);
```

Calling a Function

While creating a C function, you give a definition of what the function has to do. To use afunction, you will have to call that function to perform the defined task.

When a program calls a function, program control is transferred to the called function. A called functionperforms defined task, and when its return statement is executed or when its function-ending closingbrace is reached, it returns program control back to the main program.

To call a function, you simply need to pass the required parameters along with function name, and iffunction returns a value, then you can store returned value. For example:

```
#include <stdio.h>
/* function declaration */
intmax(int num1, int num2);
int main ()
{
/* local variable definition */
int a = 100;
int b = 200;
int ret;
/* calling a function to get max value */
ret = max(a, b);
printf( "Max value is : %d\n", ret );
return 0;
}
/* function returning the max between two numbers */
```

```
intmax(int num1, int num2)
{
/* local variable declaration */
int result;
if (num1 > num2)
result = num1;
else
result = num2;
return result;
}
```

Function Arguments

If a function is to use arguments, it must declare variables that accept the values of thearguments. These variables are called the formal parameters of the function.The formal parameters behave like other local variables inside the function and are created upon entryinto the function and destroyed upon exit.While calling a function, there are **two ways** that arguments can be passed to a function:

Function call by value

The **call by value** method of passing arguments to a function copies the actual value of anargument into the formal parameter of the function

```
/* function definition to swap the values */
void swap(int x, int y)
{
int temp;
temp = x; /* save the value of x */
x = y; /* put y into x */
y = temp; /* put x into y */
return;
        }


#include <stdio.h>
/* function declaration */
void swap(int x, int y);
int main ()
{
/* local variable definition */
int a = 100;
int b = 200;
printf("Before swap, value of a : %d\n", a );
printf("Before swap, value of b : %d\n", b );
/* calling a function to swap the values */
swap(a, b);
printf("After swap, value of a : %d\n", a );
printf("After swap, value of b : %d\n", b );
return 0;
        }
```

Function call by reference

The call by reference method of passing arguments to a function copies the address of anargument into the formal parameter. Inside the function, the address is used to access the actualargument used in the call. This means that changes made to the parameter affect the passed argument.To pass the value by reference, argument pointers are passed to the functions just like any other value.So accordingly you need to declare the function parameters as pointer types as in the following functionswap(), which exchanges the values of the two integer variables pointed to by its arguments.

```
/* function definition to swap the values */
void swap(int *x, int *y)
{
int temp;
temp = *x; /* save the value at address x */
*x = *y; /* put y into x */
*y = temp; /* put x into y */
return;
        }

#include <stdio.h>
/* function declaration */
void swap(int *x, int *y);
int main ()
{
/* local variable definition */
int a = 100;
int b = 200;
printf("Before swap, value of a : %d\n", a );
printf("Before swap, value of b : %d\n", b );
/* calling a function to swap the values.
* &a indicates pointer to a ie. address of variable a and
* &b indicates pointer to b ie. address of variable b.
*/
swap(&a, &b);
printf("After swap, value of a : %d\n", a );
printf("After swap, value of b : %d\n", b );
return 0;
        }
```

Recursive function

Recursion is the process of repeating items in a self-similar way. Same applies in programminglanguages as well where if a programming allows you to call a function inside the same function that iscalled **recursive call** of the function as follows.

```
void recursion()
{
recursion(); /* function calls itself */
}
intmain()
{
recursion();
}
```

The C programming language supports recursion, i.e., a function to call itself. But while using recursion,programmers need to be careful to define an exit condition from the function, otherwise it will go ininfinite loop.Recursive function are very useful to solve many mathematical problems like to calculate factorial of anumber, generating Fibonacci series, etc.

Number Factorial

Following is an example, which calculates factorial for a given number using a recursive function:

```
void recursion()
{
recursion(); /* function calls itself */
}
intmain()
{
recursion();
}
#include <stdio.h>
intfactorial(unsigned inti)
{
if(i<= 1)
{
return 1;
}
return i * factorial(i - 1);
}
intmain()
{
inti = 15;
printf("Factorial of %d is %d\n", i, factorial(i));
return 0;
}
```

When the above code is compiled and executed, it produces the following result:**Factorial of 15 is 2004310016**

Fibonacci Series

Following is another example, which generates Fibonacci series for a given number using arecursive function:
When the above code is compiled and executed, it produces the following result:

```
0 1 1 2 3 5 8 13 21 34
```

```
#include <stdio.h>
intfibonaci(inti)
{
if(i == 0)
{
return 0;
}
if(i == 1)
{
return 1;
}
return fibonaci(i-1) + fibonaci(i-2);
}
intmain()
{
inti;
for (i = 0; i< 10; i++)
{
printf("%d\t%n", fibonaci(i));
}
return 0;
}
```

result:

```
0 1 1 2 3 5 8 13 21 34
```

Pointers

A pointer is a variable whose value is the address of another variable, i.e., directaddress of the memory location. Like any variable or constant, you must declare apointer before using it to store any variable address.

The general form of a pointervariable declaration is –

type *var-name;

Here, type is the pointer's base type; it must be a valid C data type and var-nameis the name of the pointer variable. The asterisk * used to declare a pointer is the sameasterisk used for Multiplication. However, in this statement the asterisk is being used todesignate a variable as a pointer. Take a look at some of the valid pointer declarations −

```
int *ip; /* pointer to an integer */
double *dp; /* pointer to a double */
float *fp; /* pointer to a float */
char *ch /* pointer to a character */
```

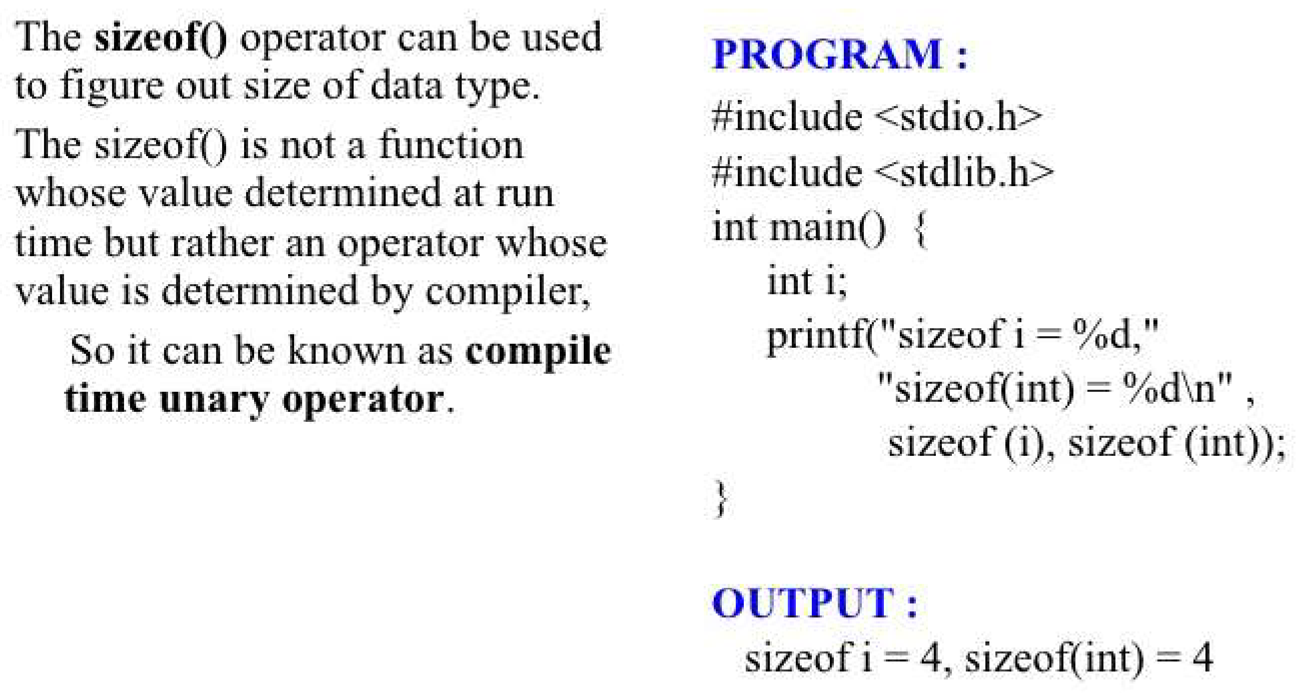
The actual data type of the value of all pointers, whether integer, float, character,or otherwise, is the same, a long hexadecimal number that represents a memory address.

The only difference between pointers of different data types is the data type of thevariable or constant that the pointer points to.

Pointer to function

C programming allows passing a pointer to a function. To do so, simply declarethe function parameter as a pointer type.Following is a simple example where we pass an unsigned long pointer to afunction and change the value inside the function which reflects back in the callingfunction –

Sizeof()



The **sizeof()** operator can be used to figure out size of data type.

The sizeof() is not a function whose value determined at run time but rather an operator whose value is determined by compiler,

So it can be known as **compile time unary operator**.

PROGRAM :

```
#include <stdio.h>
#include <stdlib.h>
int main()  {
    int i;
    printf("sizeof i = %d,"
            "sizeof(int) = %d\n" ,
             sizeof (i), sizeof (int));
}
```

OUTPUT :

```
  sizeof i = 4, sizeof(int) = 4
```

Pointer Expressions and Pointer Arithmetic**Character and**

- Like other variables, pointer variables can be used in expressions.
- If p1 and p2 are properly declared and initialized pointers, the following statements are valid:

```
y = *p1 * *p2        same as (*p1) * (*p2)
sum = sum + *p1;
z = 5* - *p2/ *p1; same as (5 * (-(*p2)))/(*p1)
*p2 = *p2 +10;
```

- Note the blank space between / and *. the following is wrong

```
z = 5* - *p2 /*p1;
```

- The symbol /* is considered as the beginning of a comment

String Declaration

1) char str1[]={'A', 'B', 'C', 'D', '\0'};

2) char str1[]="ABCD";

⇩

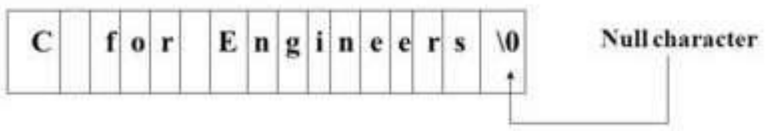
'\0' would automatically insterted at the end in this type of declaration

Character and String constant

A character string constant is a sequence of characters enclosed in double quotation mark. Examples of character string constant are as follows.

"C for Engineers"

"Programming is fun"



String to number conversion function

The following functions are present in stdlib.h

FUNCTION	PARAMETER	ACTION
`atoi`	C-string	converts C-string to an `int` value, returns the value
`atol`	C-string	converts C-string to a `long` value, returns the value
`atof`	C-string	converts C-string to a `double` value, returns the value
`itoa`	`int`, C-string, `int`	converts 1st `int` parameter to a C-string, stores it in 2nd parameter. 3rd parameter is base of converted value

String Manipulation Function

The C Library provides a rich set of string handling functions that are placed under the header file **<string.h> and <ctype.h>**.

Some of the string handling functions are (**string.h**):

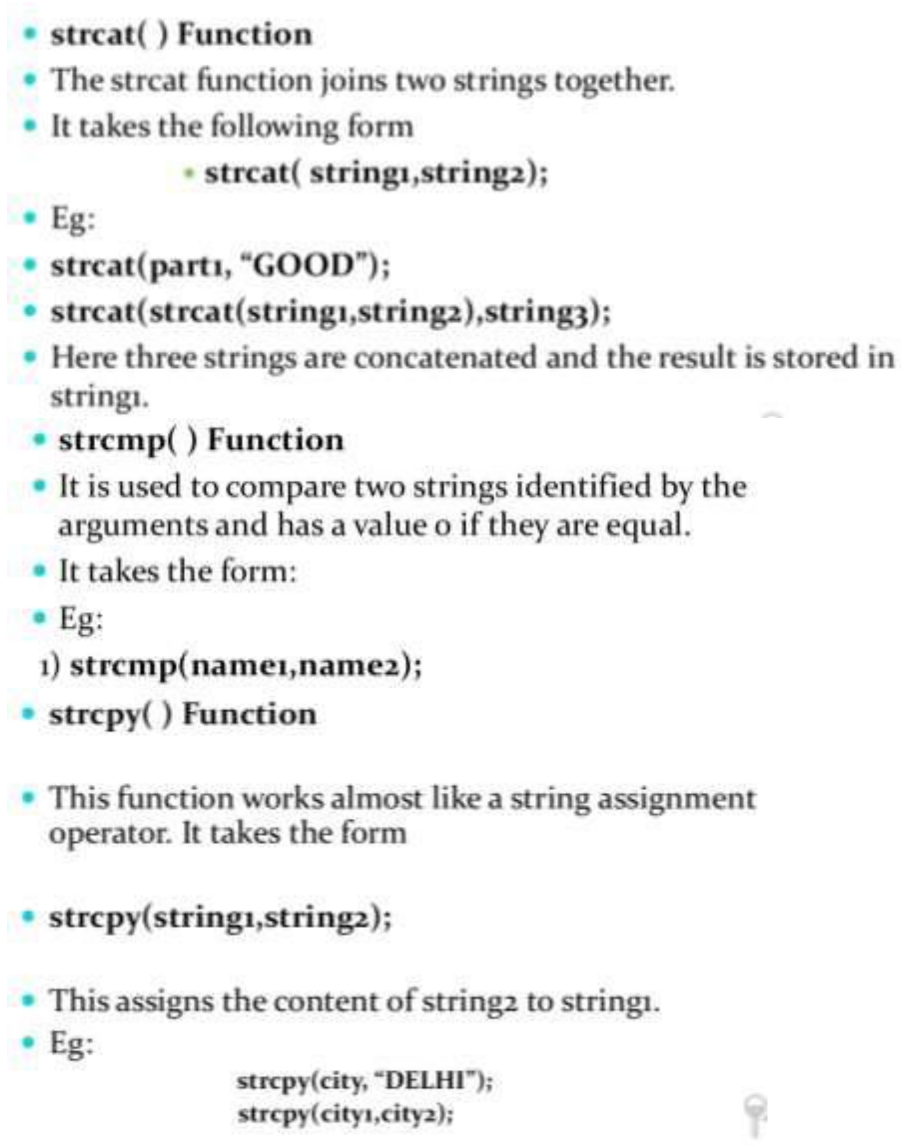
strlen()	strcat()	strcpy()	strrchr()
strcmp()	strstr()	strchr()	strrev()

Some of the string conversion functions are (**ctype.h**):

toupper()	tolower()	toascii()

All I/O functions are available in **stdio.h**

scanf()	printf()	gets()	puts()
getchar()	putchar()		



- **strcat( ) Function**
- The strcat function joins two strings together.
- It takes the following form
  - **strcat(string1,string2);**
- Eg:
- **strcat(part1, "GOOD");**
- **strcat(strcat(string1,string2),string3);**
- Here three strings are concatenated and the result is stored in string1.
- **strcmp( ) Function**
- It is used to compare two strings identified by the arguments and has a value 0 if they are equal.
- It takes the form:
- Eg:

1) **strcmp(name1,name2);**

- **strcpy( ) Function**
- This function works almost like a string assignment operator. It takes the form
- **strcpy(string1,string2);**
- This assigns the content of string2 to string1.
- Eg:

strcpy(city, "DELHI");
strcpy(city1,city2);

POINTER

A pointer is a variable that store memory address or that contains address ofanother variable where addresses are the location number always contains wholenumber. So, pointer contain always the whole number. It is called pointer becauseit points to a particular location in memory by storing address of that location.

Syntax-

Data type *pointer name;

Here * before pointer indicate the compiler that variable declared as a pointer.

e.g.

```
        Int *p1; //pointer to integer type
        float *p2; //pointer to float type
        char *p3; //pointer to character type
```

When pointer declared, it contains garbage value i.e. it may point any value in thememory.

Two operators are used in the pointer i.e.

address operator(&) and
indirectionoperator or
dereference operator (*).

Indirection operator gives the values stored at a particular address.Address operator cannot be used in any constant or any expression.

Example:

```
void main()
{
inti=105;
int *p;
p=&i;
t
printf("value of i=%d",*p);
printf("value of i=%d",*/&i);
printf("address of i=%d",&i);
printf("address of i=%d",p);
printf("address of p=%u",&p);
}
```

Pointer Expression-Pointer assignment
inti=10;
int *p=&i;//value assigning to the pointerHere declaration tells the compiler that P will be used to store the address ofinteger value or in other word P is a pointer to an integer and *p reads the **value atthe address contain in p.**
P++;
printf("value of p=%d");
We can assign value of 1 pointer variable to other when their base type and datatype is same or both the pointer points to the same variable as in the array.
Int *p1,*p2;
P1=&a[1];
P2=&a[3];
We can assign constant 0 to a pointer of any type for that symbolic constant'**NULL**' is used such as
*p=NULL;
It means pointer doesn't point to any valid memory location.
Pointer Arithmetic
Pointer arithmetic is different from ordinary arithmetic and it is perform relative tothe data type(base type of a pointer).
Example:-
If integer pointer contain address of 2000 on incrementing we get address of 2002instead of 2001, because, size of the integer is of 2 bytes.
Note:-When we move a pointer, somewhere else in memory by incrementing ordecrement or adding or subtracting integer, it is not necessary that, pointer stillpointer to a variable of same data, because, memory allocation to the variable aredone by the compiler.But in case of array it is possible, since there data are stored in a consecutivemanner.
Ex:-
```
void main( )
{
static inta[ ]={20,30,105,82,97,72,66,102};
int *p,*p1;
P=&a[1];
P1=&a[6];
printf("%d",*p1-*p);
printf("%d",p1-p);
}
```

Arithmetic operation never perform on pointer are:
Operation performed in pointer are:-
/* Addition of a number through pointer */
Example
```
inti=100;
int *p;
p=&i;
p=p+2;
p=p+3;
p=p+9;
```
ii /* Subtraction of a number from a pointer'*/
Ex:-
```
inti=22;
*p1=&a;
p1=p1-10;
p1=p1-2;
```

iii- Subtraction of one pointer to another is possible when pointer variable point toan element of same type such as an array.
Ex:-
```
in tar[ ]={2,3,4,5,6,7};
int *ptr1,*ptr1;
ptr1=&a[3]; //2000+4
ptr2=&a[6]; //2000+6
```

Precedence of dereference (*) Operator and increment operator anddecrement operator
The precedence level of difference operator increment or decrement operatoris same and their associatively from right to left.

Example :-

int x=25;
int *p=&x;
Let us calculate int y=*p++;
Equivalent to *(p++)
Since the operator associate from right to left, increment operator will applied tothe pointer p.
i) int y=*p++; equivalent to *(p++)
p =p++ or p=p+1
ii) *++p;→*(++p)→p=p+1
y=*p
iii) int y=++*p
equivalent to ++(*p)
p=p+1 then *p
iv) y=(*p)++→equivalent to*p++
y=*p then
P=p+1 ;
Since it is postfix increment the value of p.

Pointer Comparison

Pointer variable can be compared when both variable, object of same data typeand it is useful when both pointers variable points to element of same array.Moreover pointer variable are compared with zero which is usually expressed asnull, so several operators are used for comparison like the relational operator.
==,!=,<=,<,>,>=, can be used with pointer. Equal and not equal operators used tocompare two pointer should finding whether they contain same address or not andthey will equal only if are null or contains address of same variable.
Ex:-

```
void main()
{
static intarr[]={20,25,15,27,105,96}
int *x,*y;

x=&a[5];
y=&(a+5);
if(x==y)
printf("same");
else
printf("not");
}
```

Pointer to pointer

Addition of pointer variable stored in some other variable is called pointer to
pointer variable.
Or
Pointer within another pointer is called pointer to pointer.
Syntax:-
Data type **p;

```
int x=22;
int *p=&x;
int **p1=&p;
printf("value of x=%d",x);
printf("value of x=%d",*p);
printf("value of x=%d",*&x);
printf("value of x=%d",**p1);
printf("value of p=%u",&p);
printf("address of p=%u",p1);
printf("address of x=%u",p);
printf("address of p1=%u",&p1);
printf("value of p=%u",p);
printf("value of p=%u",&x);
```

Pointer vs array

Example :-

```
void main()
{
static char arr[]="Rama";
char*p="Rama";
printf("%s%s", arr, p);
```

In the above example, at the first time printf( ), print the same value array andpointer.

Here array arr, as **pointer to character** and **p act as a pointer to array ofcharacter .**When we are trying to increase the value of arr it would give the errorbecause its known to compiler about an array and its base address which is alwaysprinted to base address is known as constant pointer and the base address of array which is not allowed by the compiler.
printf("size of (p)",size of (ar));

```
size of (p)          2/4 bytes
size of(ar)           5 byes
```

STRUCTURES AND FILES	**9**
Introduction - need for structure data type - structure definition - Structure declaration - Structure within a structure - Union - Programs using structures and Unions - File Manipulation - Storage classes - Pre-processor directives.	

Sructure

It is the collection of dissimilar data types or heterogenous data types groupedtogether. It means the data types may or may not be of same type.

<u>Structure declaration</u>

Structtagname
{
Data type member1;
Data type member2;
Data type member3;
………
………
Data type member n;
};

OR

struct
{
Data type member1;
Data type member2;
Data type member3;
………
………
Data type member n;
};

OR

structtagname
{
struct element 1;
struct element 2;
struct element 3;
………
………
Structelement n;
};

Structure variable declaration;

struct student
{
int age;
char name[20];
char branch[20];

}; struct student s;

Initialization of structure variable-

Like primary variables structure variables can also be initialized when they aredeclared. Structure templates can be defined locally or globally. If it is local it canbe used within that function. If it is global it can be used by all other functions ofthe program.

We cant initialize structure members while defining the structure

struct student
{
int age=20;
char name[20]="sona";
}s1;

The above is **invalid.**

A structure can be initialized as

struct student
{
intage,roll;
char name[20];
} struct student s1={16,101,"sona"};
struct student s2={17,102,"rupa"};

If initialiser is less than no.of structure variable, automatically rest values are takenas zero.

Accessing structure elements-

Dot operator is used to access the structure elements. Its associativety is from leftto right.

structure variable ;
s1.name[];
s1.roll;
s1.age;

Elements of structure are stored in contiguous memory locations. Value ofstructure variable can be assigned to another structure variable of same type usingassignment operator.

Example:

```
#include<stdio.h>
#include<conio.h>
void main()
{
Introll, age;
char branch;
} s1,s2;
printf("\n enter roll, age, branch=");
scanf("%d %d %c", &s1.roll, &s1.age, &s1.branch);
s2.roll=s1.roll;
printf(" students details=\n");
printf("%d %d %c", s1.roll, s1.age, s1.branch);
printf("%d", s2.roll);

}
```

Unary, relational, arithmetic, bitwise operators are not allowed within structurevariables.

Size of structure-

Size of structure can be found out using sizeof() operator with structure variablename or tag name with keyword.

sizeof(struct student); or
sizeof(s1);
sizeof(s2);

Size of structure is different in different machines. So size of whole structure maynot be equal to sum of size of its members.

Array of structures

When database of any element is used in huge amount, we prefer Array ofstructures.

Example: suppose we want to maintain data base of 200 students, Array of structures is used.

```
#include<stdio.h>
#include<string.h>
struct student
{
char name[30];
char branch[25];
int roll;
};
void main()
{
struct student s[200];
inti;
s[i].roll=i+1;
printf("\nEnter information of students:");
for(i=0;i<200;i++)
{
printf("\nEnter the roll no:%d\n",s[i].roll);
printf("\nEnter the name:");
scanf("%s",s[i].name);
printf("\nEnter the branch:");
scanf("%s",s[i].branch);
printf("\n");
}
printf("\nDisplaying information of students:\n\n");
for(i=0;i<200;i++)
{
printf("\n\nInformation for roll no%d:\n",i+1);
printf("\nName:");
puts(s[i].name);
printf("\nBranch:");
puts(s[i].branch);
}
}
```

In Array of structures each element of array is of structure type as in aboveexample.

Arraywithin structures

```
struct student
{
char name[30];
introll,age,marks[5];
}; struct student s[200];
```

We can also initialize using same syntax as in array.

Nested structure

When a structure is within another structure, it is called Nested structure. Astructure variable can be a member of another structure and it is represented as

```
struct student

{
element 1;
element 2;
………
………
struct student1
{
member 1;
member 2;
}variable 1;
……….
……….
element n;
}variable 2;
```

It is possible to define structure outside & declare its variable inside otherstructure.

```
struct date
{
intdate,month;
};
struct student
{
char nm[20];
int roll;
struct date d;
}; struct student s1;
struct student s2,s3;
```

Nested structure may also be initialized at the time of declaration like in aboveexample.

struct student s={“name”,200, {date, month}};
{“ram”,201, {12,11}};

Nesting of structure within itself is not valid. Nesting of structure can beextended to any level.

```
struct time
{
inthr,min;
};
struct day
{
intdate,month;
struct time t1;
};
struct student

{
char nm[20];
struct day d;
}stud1, stud2, stud3;
```

Passing structure elements to function

We can pass each element of the structure through function but passing individualelement is difficult when number of structure element increases. To overcome this,we use to pass the whole structure through function instead of passing individualelement.

```
#include<stdio.h>
#include<string.h>
void main()
{
struct student
{
char name[30];
char branch[25];
int roll;
}struct student s;
printf("\n enter name=");
gets(s.name);
printf("\nEnter roll:");
scanf("%d",&s.roll);
printf("\nEnter branch:");
gets(s.branch);
display(name,roll,branch);
}
display(char name, int roll, char branch)
{
printf("\n name=%s,\n roll=%d, \n branch=%s", s.name, s.roll. s.branch);
}
```

Passing entire structure to function

```
#include<stdio.h>
#include<string.h>
struct student
{
char name[30];
intage,roll;
};
display(struct student); //passing entire structure
void main()
{
struct student s1={"sona",16,101 };
struct student s2={"rupa",17,102 };
display(s1);
display(s2);
}
display(struct student s)
{
printf("\n name=%s, \n age=%d ,\n roll=%d", s.name, s.age, s.roll);
}
Output: name=sonaroll=16
```

UNION

Union is derived data type contains collection of different data type or dissimilarelements.

All definition declaration of union variable and accessing member issimilar to structure, but instead of keyword struct the keyword union is used, themain difference between union and structure is

Each member of structure occupy the memory location, but in the unionsmembers share memory. Union is used for saving memory and concept is usefulwhen it is not necessary to use all members of union at a time.

Where union offers a memory treated as variable of one type on one occasionwhere (struct), it read number of different variables stored at different place ofmemory.

Syntax of union:

```
union student
{
datatype member1;
datatype member2;
};
```

Like structure variable, union variable can be declared with definition or separatelysuch as
union union name
{
Datatype member1;
}var1;

Example:- union student s;
Union members can also be accessed by the dot operator with union variable and ifwe have pointer to union then member can be accessed by using (arrow) operatoras with structure.

Example:-struct student
struct student
{
inti;
char ch[10];
};struct student s;
Here datatype/member structure occupy 12 byte of location is memory, where as inthe union side it occupy only 10 byte.
Nested of Union
When one union is inside the another union it is called nested of union.
Example:-
union a
{
inti;
int age;
};
union b

{
char name[10];
union a aa;
}; union b bb;
There can also be union inside structure or structure in union.
Example:-
void main()
{
struct a
{
inti;
char ch[20];
};
struct b
{
inti;
char d[10];
};
union z
{
struct a a1;
struct b b1;
}; union z z1;
z1.b1.j=20;
z1.a1.i=10;
z1.a1.ch[10]= " i";
z1.b1.d[0]="j ";
printf(" ");
Dynamic memory Allocation
The process of allocating memory at the time of execution or at the runtime, iscalled dynamic memory location.Two types of problem may occur in static memory allocation.If number of values to be stored is less than the size of memory, there would bewastage of memory.
If we would want to store more values by increase in size during the execution onassigned size then it fails.
Allocation and release of memory space can be done with the help of some libraryfunction called dynamic memory allocation function. These library function arecalled as **dynamic memory allocation function.** These library function prototypeare found in the header file, "alloc.h" where it has defined.
Function take memory from memory area is called heap and release when notrequired.
Pointer has important role in the dynamic memory allocation to allocate memory.

malloc():
This function use to allocate memory during run time, its declaration is
void*malloc(size);
malloc ()
returns the pointer to the 1st byte and allocate memory, and its return type is void,which can be type cast such as:
int *p=(datatype*)malloc(size)
If memory location is successful, it returns the address of the memory chunk thatwas allocated and it returns null on unsuccessful and from the above declaration apointer of type**(datatype)** and size in byte.
And **datatype** pointer used to typecast the pointer returned by malloc and this
typecasting is necessary since, malloc() by default returns a pointer to void.

Example int*p=(int*)malloc(10);
So, from the above pointer p, allocated IO contigious memory space address of 1st
byte and is stored in the variable.
We can also use, the size of operator to specify the the size, such as

p=(int)malloc(5*size of int) Here, 5 is the no. of data.
Moreover , it returns null, if no sufficient memory available , we should alwayscheck the malloc return such as,
if(p==null)
printf("not sufficient memory");

Example:
```
/*calculate the average of mark*/
void main()
{
intn ,avg,i,*p,sum=0;
printf("enter the no. of marks ");
scanf("%d",&n);
p=(int *)malloc(n*size(int));
if(p==null)
printf("not sufficient");
exit();
}
for(i=0;i<n;i++)
scanf("%d",(p+i));
for(i=0;i<n;i++)
Printf("%d",*(p+i));
sum=sum+*p;
avg=sum/n;
printf("avg=%d",avg);
```

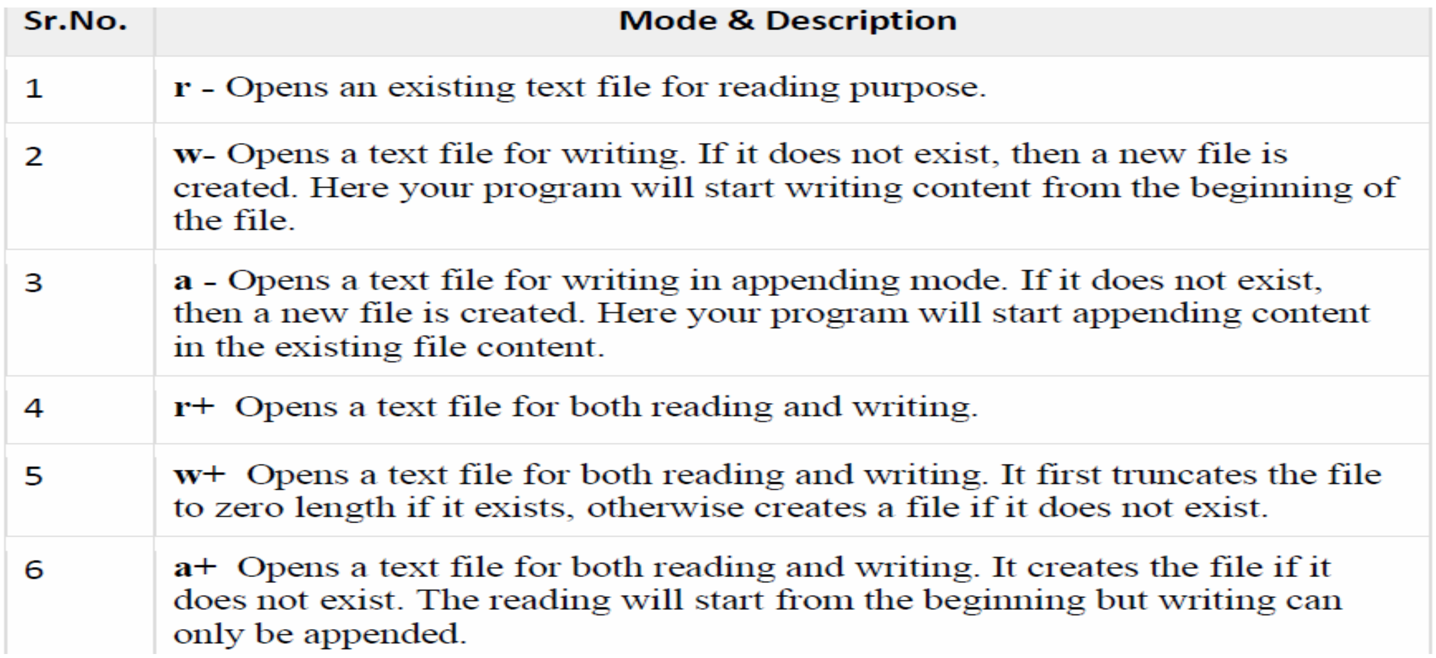
calloc()
Similar to malloc only difference is that calloc function use to allocate multiple block of memory . two arguments are there
1st argument specify number of blocks
2nd argument specify size of each block.
Example:-
int *p= (int*) calloc(5, 2);
int*p=(int *)calloc(5, size of (int));
Another difference between malloc and calloc is by default memory allocated bymalloc contains garbage value, where as memory allocated by calloc is initialisedby zero(but this initialisation) is not reliable.
realloc()
The function realloc use to change the size of the memory block and it alter thesize of the memory block without loosing the old data, it is called reallocation ofmemory.

C - FILE I/O
This chapter cover how C programmers can create, open, close text or binary files for their data storage. A file represents a sequence of bytes, regardless of it being a text file or a binary file.
C programming language provides access on high level functions as well as low level OSlevelOSlevel calls to handle file on your storage devices. This chapter will take you through the important calls for file management.

Opening Files
You can use the **fopen** function to create a new file or to open an existing file. This call will initialize an object of the type **FILE**, which contains all the information necessary to control the stream. The prototype of this function call is as follows –

```
FILE *fopen( const char * filename, const char * mode );
```

Here, **filename** is a string literal, which you will use to name your file, and access **mode** can have one of the following values –



Sr.No.	Mode & Description
1	**r** - Opens an existing text file for reading purpose.
2	**w**- Opens a text file for writing. If it does not exist, then a new file is created. Here your program will start writing content from the beginning of the file.
3	**a** - Opens a text file for writing in appending mode. If it does not exist, then a new file is created. Here your program will start appending content in the existing file content.
4	**r+** Opens a text file for both reading and writing.
5	**w+** Opens a text file for both reading and writing. It first truncates the file to zero length if it exists, otherwise creates a file if it does not exist.
6	**a+** Opens a text file for both reading and writing. It creates the file if it does not exist. The reading will start from the beginning but writing can only be appended.

Closing a File

To close a file, use the fclose function. The prototype of this function is −

```
int fclose( FILE *fp );
```

The **fclose**−− function returns zero on success, or EOF if there is an error in closing the file. This function actually flushes any data still pending in the buffer to the file, closes the file, and releases any memory used for the file.

The EOF is a constant defined in the header file stdio.h. There are various functions provided by C standard library to read and write a file, character by character, or in the form of a fixed length string.

Writing a File

Following is the simplest function to write individual characters to a stream –

int fputc(int c, FILE *fp);

The function fputc writes the character value of the argument c to the output stream referenced by fp. It returns the written character written on success otherwise EOF if there is an error. You can use the following functions to write a null-terminated string to a stream –

int fputs(const char *s, FILE *fp);

The function fputs writes the string s to the output stream referenced by fp. It returns a nonnegative value on success, otherwise EOF is returned in case of any error.
You can use int printfFILE∗fp,constchar∗format,...FILE∗fp,constchar∗format,... function as well to write a string into a file. Try the following example.

```
#include <stdio.h>
main() {
FILE *fp;
fp = fopen("/tmp/test.txt", "w+");
fprintf(fp, "This is testing for fprintf...\n");
fputs("This is testing for fputs...\n", fp);
fclose(fp);
}
```

Reading a File

Given below is the simplest function to read a single character from a file –

int fgetc(FILE * fp);

The fgetc function reads a character from the input file referenced by fp. The return value is the character read, or in case of any error, it returns EOF. The following function allows to read a string from a stream −

char *fgets(char *buf, int n, FILE *fp);

The functions fgets reads up to n-1 characters from the input stream referenced by fp. It copies the read string into the buffer buf, appending a null character to terminate the string. If this function encounters a newline character '\n' or the end of the file EOF before they have read the maximum number of characters, then it returns only the characters read up to that point including the new line character. You can also use int fscanfFILE∗fp,constchar∗format,...FILE∗fp,constchar∗format,... function to read strings from a file, but it stops reading after encountering the first space character.

```
#include <stdio.h>
main() {
FILE *fp;
char buff[255];
fp = fopen("/tmp/test.txt", "r");
fscanf(fp, "%s", buff);
printf("1 : %s\n", buff );
fgets(buff, 255, (FILE*)fp);
printf("2: %s\n", buff );
fgets(buff, 255, (FILE*)fp);
printf("3: %s\n", buff );
fclose(fp);
}
```

Binary I/O Functions
There are two functions, that can be used for binary input and output −
size_t fread(void *ptr, size_t size_of_elements, size_t number_of_elements, FILE *a_file);

.Both of these functions should be used to read or write blocks of memories - usually arrays or structures.

Random Access to File

There is no need to read each record sequentially, if we want to access a particular record.C supports these functions for random access file processing.

1. fseek()
2. ftell()
3. rewind()

fseek():

This function is used for seeking the pointer position in the file at the specified byte.

Syntax: fseek(file pointer, displacement, pointer position);

Where **file pointer ----** It is the pointer which points to the file.

displacement ---- It is positive or negative.This is the number of bytes which are skipped backward (if negative) or forward(if positive) from the current position.This is attached with L because this is a long integer.

ftell()

This function returns the value of the current pointer position in the file.The value is count from the beginning of the file.