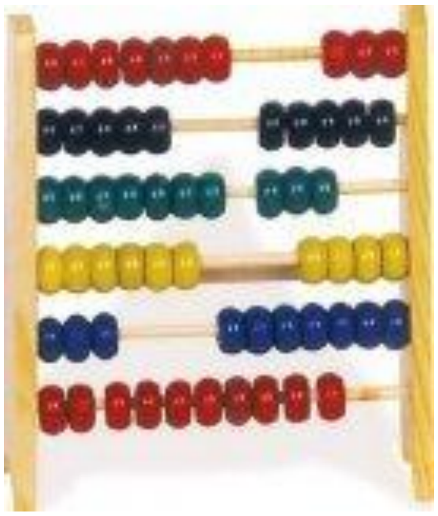
INTRODUCTION TO COMPUTER

HISTORY OF COMPUTER:

In earlier days merchants in Egypt and china used a device called as ABACUS. It was used to do all kinds of mathematical operations like addition, subtraction, multiplication, division etc.,, abacus was in use for around 4000 years. This was the beginning for usage of computers.



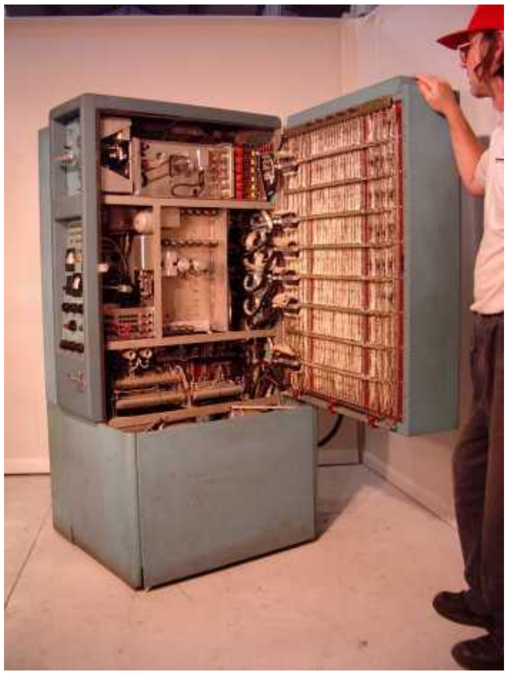
A typical structure of abacus computer

CHARLES BABBAGE (1792 – 1871):

- He is known as FATHER OF COMPUTERS
- He is a mathematics professor who created the design for computer.
- In 1822 he created differential engine to do mathematical calculations.
- In 1833 he invented analytical engine which had 5 functional units :
    - Input unit
    - Output unit
    - Arithmetic unit
    - Control unit
    - Memory unit
- The ideas formulated by babbage in designing the analytical engine helped the scientists to lay foundation for the human's greatest achievement COMPUTER.

GENERATIONS OF COMPUTER

First generation computers : (1945 – 1956) **(sample figures of first generation computers)**



Machine language or binary language was the only language that was understood by computers. Binary language will be in the form of 0 and 1.

Major developments of this generation were:

- ENIAC (Electronic Numerical Integrator And Calculator)
- EDVAC (Electronic Discrete Variable And Automatic computer)
- UNIVAC (Universal Automatic Computer)
- VACCUM TUBES were used to manufacture he computers. The heat generated by vaccum tubes damaged the sensitive parts of computer. This was the main problem with vaccum tubes.
- Also machine language was very difficult to understood.

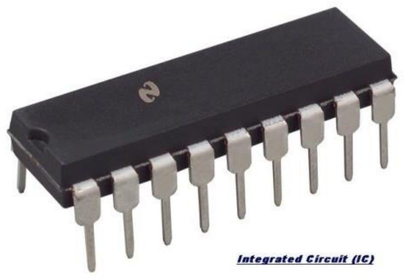
Second generation computers: (1956 – 1963)

(sample figure of second generation computer)

- Second generation computer machines were based on transistor technology.
- Second generation computers were smaller as compared to the first generation computers
- The computational time of Second generation computers was reduced to microseconds from milliseconds.
- Assembly language was used to program Second generation computers. Hence, programming became more time-efficient and less cumbersome.
- Major inventions of this generation were :
  - COBOL (Common Business Oriented Language)
  - FORTRAN (FORmula TRANslator)
  - ASCII CODE (American Standard Code for Information Interchange)

Third generation computers: (1964 – 1971)

- Integrated chip (IC) was used in place of transistors for manufacturing computers.



--------> INTEGRATED CHIP (IC)

- 3rd generation computers were more smaller and cheaper than 2nd generation computers.
- PDP-8, PDP-11 IBM370 are some of the 3rd generation computers
  (**if possible try to search for diagrams from internet**)

Fourth generation computers : (1971 – present)

Major developments of this generation are :

- Personal computer (PC)
- Graphical User Interface (GUI)
- Computers were designed with MICRO PROCESSOR technology.

Fifth generation computers : (future)

- Aim of computer experts is to develop ARTIFICIAL INTELLIGENCE (AI) have all qualities of humans.

BASIC TERMS

Data: it is a collection of raw facts.

Information: it is a processed data(meaningful data). It is the result of processing the data.

Input: the data and information entering (given by user) into a computer.

Output: the resultant information obtained by a computer.

Instruction: is to process the given data.

Program: is a sequence of instructions that can be executed by the computer to solve the given problem.

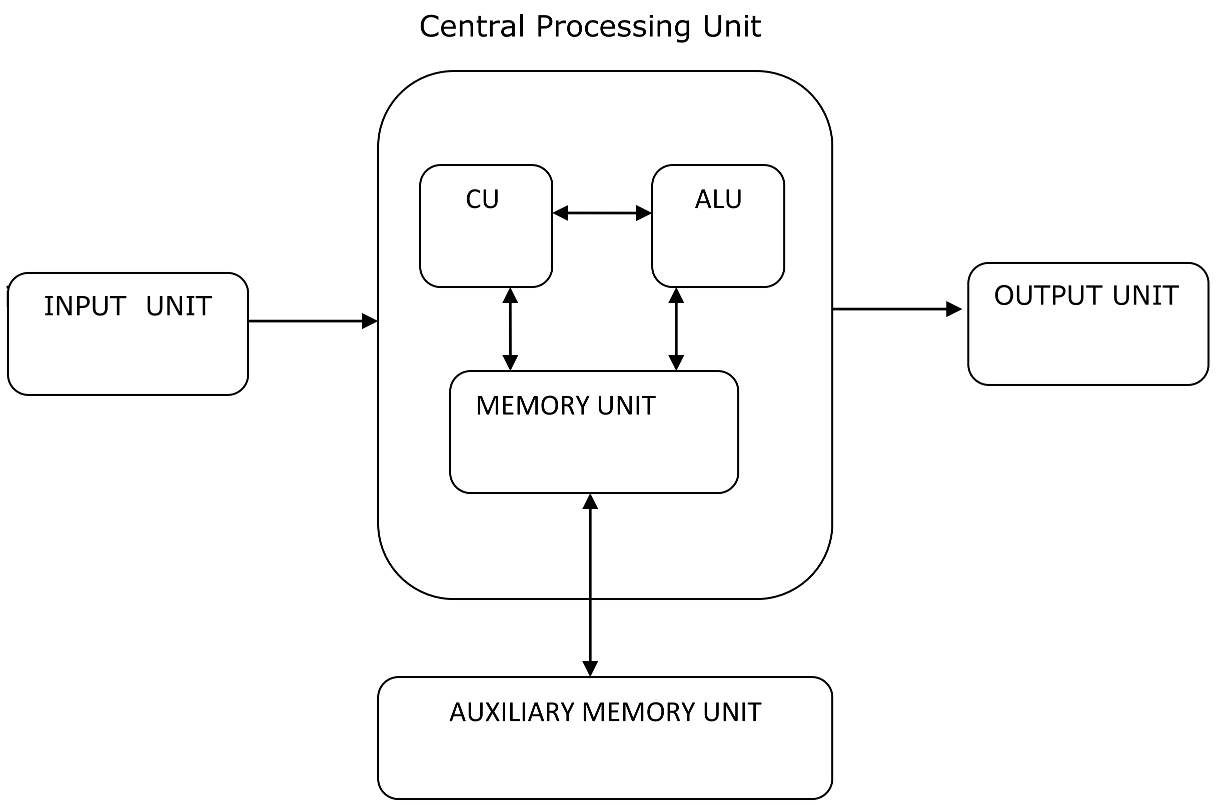
COMPUTER

A computer is a electronic device which accepts data as input, process it according to instructions given, and displays the output.

Characteristics of a computer :

- Huge data storage (large amount of data can be stored)
- High- speed calculations (calculation are done quickly)
- Accuracy (correct result is produced)
- Reliability (gives consistent results even though it is based on electronic circuits)
- Diligence (won't get tired even it works continuously for many hours)

BLOCK DIAGRAM OF COMPUTER



1. **INPUT UNIT:**

It obtains data from various input devices and place this information at the disposal of the other units so that information may be processed. Most information is entered into computer today through keyboard and mouse devices. Information can also enter by speaking and by scanning images. Some of other input devices are touch screen (used in ATM's), joystick, electronic pen etc. unit

2. **CENTRAL PROCESSING UNIT:**

The CPU performs functions similar to the Brain of a human body. It consists of three components :

- Control unit (CU)
- Arithmetic and Logical unit (ALU)
- Memory unit

Control Unit :

The control unit directs all operations inside the computer. It is known as nerve centre of the computer because it controls and coordinates all hardware operations i.e. those of the CPU and input-output devices. Its actions are

- It fetches the required instructions from the main storage.
- It also transfers the results from ALU to the memory and onto the output device for printing.
- It gives command to transfer data from the input device to the memory to ALU.

Arithmetic and Logic Unit (A.L.U):

It operates on the data available in the main memory and sends them back after processing, once again to the main memory A.L.U performs two functions

1) It carries out arithmetical operations like addition, subtraction, multiplication and division .
2) It performs certain logical actions based on AND and OR functions.

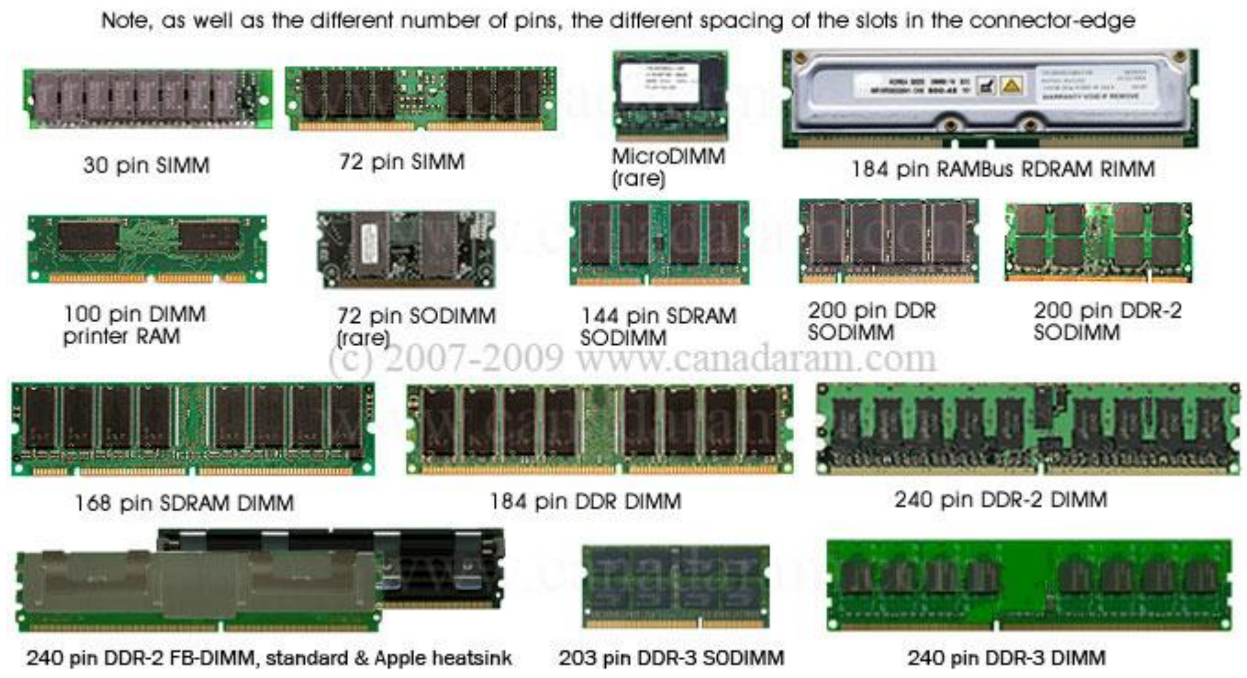
3. **Memory Unit :**

It is also called as main memory / primary memory because, like the human memory it is able to store information, which can be recalled or accessed when required. The program of instructions has to be stored in the main memory in order to make it work automatically.

Types of main memory are :

➔ Random Access Memory (RAM)
➔ Read Only Memory (ROM)

Random Access Memory (RAM) : It is a volatile memory i.e., data cannot be stored permanently. All the data entered into system are directly stored into RAM. Users can READ from or WRITE into this memory. Once the system is turned off, these contents will be erased.



Read Only Memory (ROM) : It is non-volatile memory i.e., data can be stored permanently. The contents of ROM can only be read but cannot be changed. Some variations of ROM are :-

- **PROM (Programmable read only memory)**
  - ➢ Contents of this memory are decided by user. Once the chip is programmed the recorded instructions are permanently loaded during manufacturing of computer.

- **EPROM (Erasable programmable read only memory)**
  - ➢ Similar to PROM but contents of this memory can be changed by exposing his memory to ultra-violet rays.

- **EEPROM (Electrically Erasable programmable read only memory)**
  - ➢ Contents of this memory can be erased electrically and later new information can be stored.

4. **Output Unit :**

The result of any computer processing has to be communicated to the user. Output devices translate the computers output into a form understandable to human beings. Some of the output devices are display screen, printer ,speakers etc.,,,

5. **Auxiliary Memory Unit :**

It is also called as SECONDARY MEMORY / EXTERNAL STORAGE. These are the devices that hold the mass of information, which may be transferred during processing. These devices are used for permanent storage of data. As compared to the primary memory, it has a much larger capacity, but is not as fast. The computer thus takes slightly more time to retrieve from secondary storage.

Example: Hard disk, Compact disks (CDs)

COMPONENTS OF COMPUTER

The major components of computer are **HARDWARE** & **SOFTWARE**.

HARDWARE:

It is a general term used to represent the physical and tangible components of the computer itself i.e. those components which can be touched and seen. It includes

(1) Input devices
(2) Output devices
(3) Central processing Unit
(4) Storage devices

SOFTWARE:

It is a general term to describe all the forms of programs associated with a computer. Without software, a computer is like a human body with out a soul.

It is a general term to describe all the forms of programs associated with a computer. Without software, a computer is like a human body without a soul.

The classifications of software are :-

- System software
- Application software

System software includes :-

i) Operating system
ii) Language Translators

OPERATING SYSTEM is the interface between the user and the computer hardware.

Ex: MS-DOS, Solaris ,Unix , Linux.

LANGUAGE TRANSLATOR is a software that accepts input in one language and convert them into another language.

The types of translators are :

- Compiler
- Assembler
- Interpreter

Application software is a set of programs that allows the computer to help users in solving problems.

Ex:- MS-OFFICE , web browsers etc.,,

OPERATING SYSTEM (OS)

An ***operating system*** is a set of programs which acts as an interface between the user of computer and the computer hardware. User or application programs uses the hardware indirectly through operating system

The OS controls and co- ordinates the use of hardware among the application programs

The primary aim of OS is to provide convenience to the user in using the system.

The secondary aim is to use the system hardware in an efficient way.

The OS is used to manage the various resources and operations of the computer system.Depending on the manfacture,the OS is called by the different names such as *monitor, supervisior , executive, kernel.*

The major functions of OS are:

- Scheduling and loading of programs in order to provide a continuous job processing sequence.
- Controlling the hardware resource such as I/O devices,secondary storage device etc.
- Protecting hardware ,software,and data from the improper use.
- Memory and CPU time management.

- File and software management.
- Facilitates easy communication between the computer and the user.

A typical OS will have the following modules

a)**CPU Schedules**: it allocates CPU to the various resources such as I/O devices, memory devices etc.,

b)**Device Manger :** It makes devices functional .it manages various input and output devices that are interfaced with the system at later date.

c)**File Manager** : it provides facilities to the users for using the system efficiently.It also makes the usage of system easy.

d)**Other Utilities:** For information maintenance such as text editor, copy file from one media to another ,make the various modules usable etc.

Evolution of OS :

1.Batch Processing:several jobs are stacked together and processed them in groups(batches) for efficient operation. several programs run one after another without need of human operator to run each program individually

2.Multi Programming or Multitasking: Multiprogramming refers to the interleaved execution of two or more different and independent programs by the same computer.in this technique ,more than one program are kept in the memory. The OS starts executing a job .when the job needs to perform an I/O operation OS switches the CPU to next job.when the job waits,CPU is switched to execute another job, and so on.Finally,the first job will have finished waiting and will get the CPU back ,As long as there is some job to complete, the CPU will never be idle.

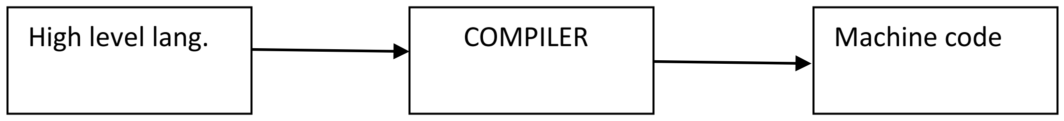
3.Time sharing:A time shared OS allow several users to share the computer on time basis. Many programs reside in main memory. The CPU allots a fixed time slice or quantum to each job. when the time slice of a job expires or the job need to perform an I/O operation ,the OS switches the CPU to next job in the queue .As the CPU switches rapidly from one job to the next job ,users of the computer feel that they are working independently while one computer is shared among many user actually.

4.Multi Processing: A multiprocessing system is one in which more than one processor are linked together in a co coordinated way.They share main memory and I/O devices. They can also execute portions of the same program

1. **Compiler**

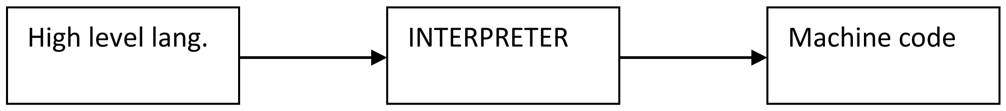
- A compiler is a program which translates a high level language program into machine language of the host computer.
- It translates all the instructions of HLL program at a time. A compiler also produces the error messages that occurs during translation.
- Once a program is error free , then the compiled software is not needed in the memory because the machine code itself can be used again and again.

Ex: Pascal complier , C Complier, COBOL Complier.



2. **Interpreter**

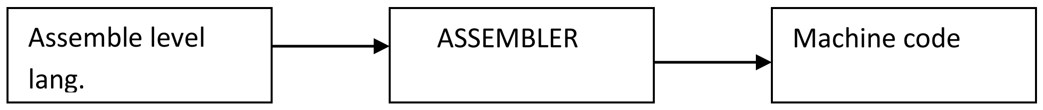
- A interpreter is a program which translate s a high level language program into machine language of the host computer.
- It translates each instruction and executes it immediately. If there is any error in any line ,it shows the error message at the same time and program execution cannot continue until the error is rectified.
- Once a program is error free ,the memory is unnecessarily occupied by interpreter as it has to reside for every run.



3. **Assembler**

An assembler is program which translates an assembly language program into the machine language. It gives one machine language instruction for each assembly language instruction. It also produces error messages .

Ex: 8085 assembler,8086 assembler



Differences between Compiler and Interpreter

Compiler	Interpreter
Popular for developed programs.	Popular in program development environment.
Converts entire code at a time.	Converts the code line by line.
Creates a permanent object file.	No object file is created.
Reports all the errors at a time.	It stops at the first error in program.

TYPES OF PROGRAMMING LANGUAGES

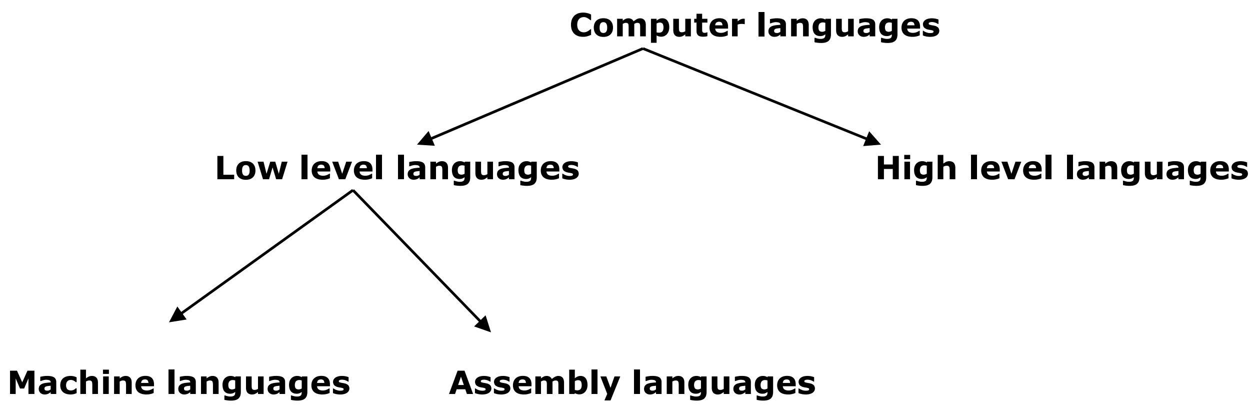
A language is meant for communication purpose

Natural language like English, Hindi, Telugu etc., are generally used by human beings to communicate with each other.

With these languages we can express or exchange our ideas.

Similarly a computer language is used for communication between human being and the computer.

A program language can be defined as **a language used for expressing a set of computer instructions called program**



MACHINE LANGUAGE

MLL can be defined as **a set of instructions coded so that the computer can use it directly without further translation**

Advantages of machine Language:

1.Computer understands only the machine language

2.Programs run very fast

Limitations of Machine Language:

1.Programmer has to remember the following

a)Binary equivalent to the OP code

b)Binary address of the memory location being used

2.Programmer written in machine language are very lengthy

3.Difficult to correct /modify programs

4.Machine language is machine dependent. That is program written for a particular computer has to be changed for using on a different computer.

ASSEMBLY LEVEL LANGUAGE:

Each instruction of assembly level language contains

a) symbolic operation code and
b) symbolic address

Ex:

ADD	is mnemonic op code for addition
SUB	is mnemonic op code for subtraction
LOAD	is mnemonic op code for loading

STORE is mnemonic op code for storing

Suppose to add two number stored in locations B and C and store the result in storage location D

assemble language program as follows:

```
LOAD  B    load the contents  of  B into accumulator

ADD    C   add the contents of C to accumulator

STORE D  store the result (contents of accumulator )into storage location D
```

Advantages of assembly Language:

1. Program correction /modification is relatively easier

2. Less time is required in writing a program than machine language

3. Language is simple when compared with machine language

Limitations of assembly Language:

1.Execution of program takes more time as it has to be converted into machine language

HIGH LEVEL LANGUAGES(HLL)

These are English like languages that are close to our native language.

Ex: BASIC, COBOL, FORTAN, PASCAL, C etc.,

Suppose to add two number stored in locations B and C and store the result in storage location D

```
Pascal          :       D:=B+C;

Cobol           :       ADD B to C GIVING D

C               :        D=B+C
```

Advantages of High level Language:

1.The High level Language are easier to use and understand than the machine languages and the assembly language

2. The programs written in HLL are much more compact and self explanatory than their low level counterparts

3. High level Language are provide   a better documentation than low level languages.

4. High level Language are machine independent

5. 2.Program correction /modification is much easier than low level languages.

Limitations of High level Language:

1.Less efficient as they take more execution time when compared to low level languages

2.The programmer cannot completely connect the total power available at hardware level .

ALGORITHM

A step-by-step problem-solving procedure for solving a problem in a finite number of steps is known as algorithm.

Properities of the algorithm are :-

) Finiteness: - an algorithm terminates after a finite numbers of steps.

2) Definiteness: - each step in algorithm is unambiguous. This means that the action specified by the step cannot be interpreted (explain the meaning of) in multiple ways & can be performed without any confusion.

3) Input:- an algorithm accepts zero or more inputs

4) Output:- it produces at least one output.

5) Effectiveness:- it consists of basic instructions that are realizable. This means that the instructions can be performed by using the given inputs in a finite amount of time.

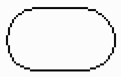
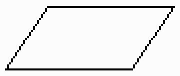
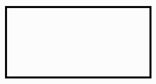
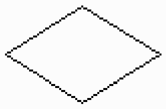
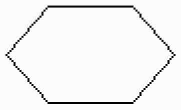
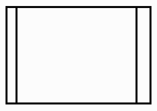
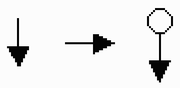
Example : algorithm to find the sum of 'n' integers

```
Step-1 : begin
Step-2 : initialize variables s=0 and n
Step-3: enter the number integers
Step-4: enter the elements
Step-5: calculate sum using  s = n*(n+1)/2
Step-6: print  n value
Step-7: end
```

FLOWCHART

Pictorial representation of an algorithm in known as flowchart.

Different symbols used in flow chart are :

	An oval is used to indicate the beginning or end of an algorithm.
	A parallelogram indicates the input or output of information.
	A rectangle indicates a computation, with the result of the computation assigned to a variable.
	A diamond indicates a point where a decision is made.
	A hexagon indicates the beginning of the repetition structure.
	A double lined rectangle is used at a point where a subprogram is used.
	An arrow indicates the direction of flow of the algorithm. Circles with arrows connect the flowchart between pages.

ADVANTAGES OF USING FLOWCHARTS

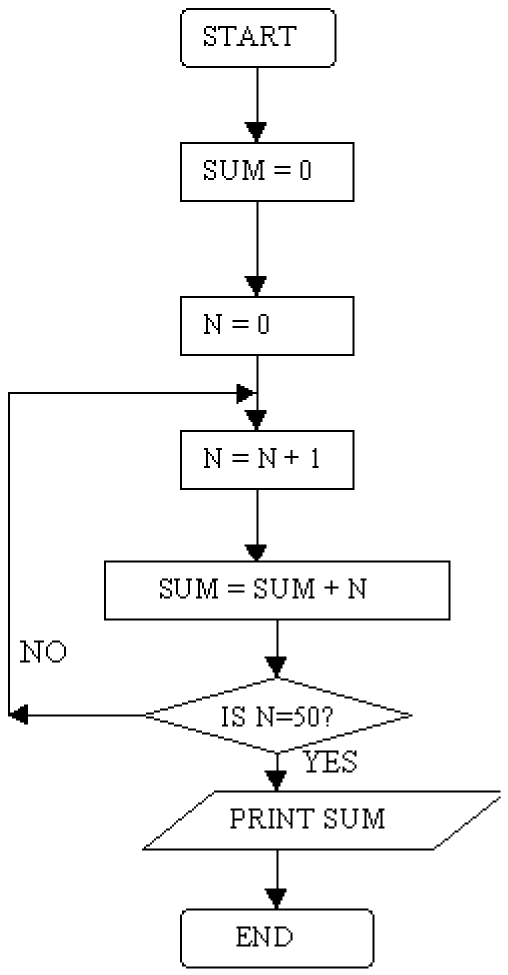
The benefits of flowcharts are as follows:

1. Communication: Flowcharts are better way of communicating the logic of a system to all concerned.
2. Effective analysis: With the help of flowchart, problem can be analysed in more effective way.
3. Proper documentation: Program flowcharts serve as a good program documentation, which is needed for various purposes.
4. Efficient Coding: The flowcharts act as a guide or blueprint during the systems analysis and program development phase.
5. Proper Debugging: The flowchart helps in debugging process.
6. Efficient Program Maintenance: The maintenance of operating program becomes easy with the help of flowchart. It helps the programmer to put efforts more efficiently on that part

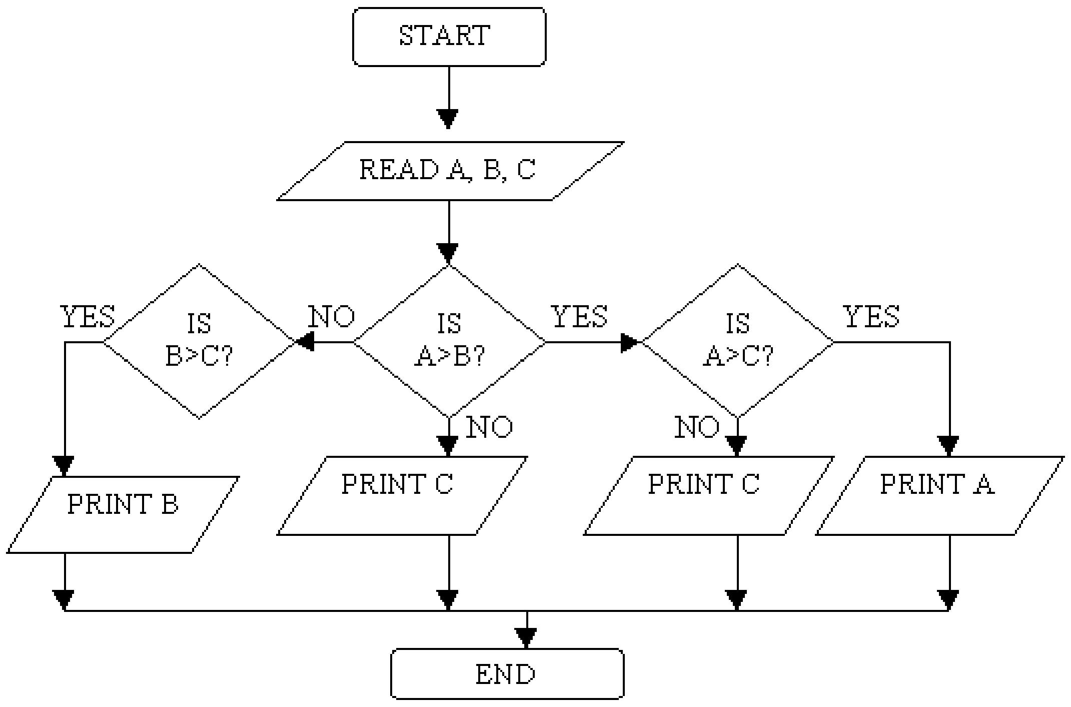
LIMITATIONS OF USING FLOWCHARTS

1. Complex logic: Sometimes, the program logic is quite complicated. In that case, flowchart becomes complex and clumsy.
2. Alterations and Modifications: If alterations are required the flowchart may require re-drawing completely.
3. Reproduction: As the flowchart symbols cannot be typed, reproduction of flowchart becomes a problem.
4. The essentials of what is done can easily be lost in the technical details of how it is done.

Flow chart for finding the sum of first 50 numbers



Flowchart for finding maximum of 3 numbers



NOTE :

Practice writing algorithms for as many programs as possible.

NUMBER SYSTEM

A number system of base (or radix), r is a system that uses distinct symbols for r digits. To determine the quantity that the number represents, it is necessary to multiply each digit by an integer power of r and then form the sum of all weighted digits.

They are of four types:

1. Decimal number system.
2. Binary number system.
3. Octal number system.
4. Hexa decimal number system.

Decimal number system:

The decimal number employs the radix – 10.

Ex:- $724.5_{(10)}$ is represented as

7	2	4	.	5
10^2	10^1	10^0		10^{-1}

$= 7*10^2 + 2*10^1 + 4*10^0 + 5*10^{-1} \quad = \quad 700+20+4+0.5$

Binary number system:

The binary number system uses the radix 2. The two digit symbols used are 0 and 1.

Ex:- $1101_{(2)}$ is represented as

1 1 0 1 =1*8 + 1*4 + 0*2 +1*1 = $13_{(10)}$

2^3 2^2 2^1 2^0

Octal number system:

The octal number system uses the radix 8. The eight numbers used in octal system are 0-7.(i.e, 0,1,2,3,4,5,6,7).

Ex:- $34.6_{(8)}$

3 4 . 6

8^1 8^0 8^{-1}

= 8 * 3 + 1 * 4 + 6 * 0.125 = 24 + 4 + 0.75 = $28.75_{(10)}$.

Hexa decimal number system:

The hexa decimal number system uses the radix 16. The sixteen numbers used in hexa decimal system are 0-F.(i.e., 0,1,2,3,4,5,6,7,8,9,A-10,B-11,C-12,D-13,E-14,F-15).

Ex:- $F3_{(16)}$

F 3 = 15 * 16 + 3 * 1 = 240 + 3 = $243_{(10).}$

16^1 16^0

CONVERSIONS

Decimal to Binary:

1) 36

```
2 | 36
2 | 18  - 0

2 | 9   - 0

2 | 4   - 1

2 | 2   - 0

  | 1   - 0
```

So binary equalent of $36_{(10)}$ is $(1\ 0\ 0\ 1\ 0\ 0)_{(2)}$

2) 41

```
2 | 41
2 | 20  - 1

2 | 10  - 0

2 |  5  - 1

2 | 2   - 1

  | 1   - 0
```

So binary equalent of $41_{(10)}$ is $(1 0 1 0 0 1)_{(2)}$

Floating point to Binary:

1) $36.12_{(10)}$

```
2 | 36
2 | 18  - 0

2 | 9   - 0

2 | 4   - 1

2 | 2   - 0

  | 1   - 0
```

For decimal value 0.12

0.12 * 2 = 0.24

0.24 * 2 = 0.48

0.48 * 2 = 0.96 so binary value for decimal place is taken till

0.96 * 2 = 1.92 1.92 – i.e., (0 0 0 1)

0.92 * 2 = 1.84

0.84 * 2 = 1.68

So binary equalent of $36.12_{(10)}$ is $(1 0 0 1 0 0 . 0 0 0 1)_{(2)}$

2) $14. 625_{(10)}$

```
2 | 14
```

```
2 |  7  - 0
2 |  3  - 0
  |  1  - 1
```

0.625 * 2 = 1.25

0.25   * 2 = 0.50

0.5   * 2 = 1.0   so binary value for decimal place is taken till

Last – i.e., (1 0 1)

So binary equalent of $14.625_{(10)}$ is $(1\ 1\ 0\ 0\ .\ 1\ 0\ 1)_{(2)}$

Binary to Decimal:

1) $(1\ 0\ 0\ 1\ 0\ 0)_{(2)}$

1	0	0	1	0	0
2^5	2^4	2^3	2^2	2^1	2^0

1 * 32 + 0 + 0 + 1 * 4 + 0 + 0 = 32 + 4 = $36_{(10)}$.

2) $(1\ 0\ 0\ 0\ 0\ 1)_{(2)}$

1 0 0 0 0 1

$2^5 \quad 2^4 \quad 2^3 \quad 2^2 \quad 2^1 \quad 2^0$

$1 * 32 + 0 + 0 + 0 + 0 + 1 * 1 = 32 + 1 = 33_{(10)}$.

Octal number to Binary number:

Convert each number to Binary format using 3 Binary digits.

1) $72_{(8)}$ - $(1\,1\,1 \;\; 0\,1\,0)_{(2)}$
2) $64_{(8)}$ - $(1\,1\,0 \;\; 1\,0\,0)_{(2)}$

Hexa Decimal number to Binary number:

Convert each number to Binary format using 4 Binary digits.

1) $AF_{(16)}$ - (10 15) - $(1\,0\,1\,0 \;\; 1\,1\,1\,1)_{(2)}$
2) $A3_{(16)}$ - (10 3) - $(1\,0\,1\,0 \;\; 0\,0\,1\,1)_{(2)}$

Hexa Decimal number to Decimal number:

1) $30_{(16)}$ - $3 * 16^1 + 0 = 48 + 0 = 48_{(10)}$
2) $B65F_{(16)}$ - $11 * 16^3 + 6 * 16^2 + 5 * 16^1 + 15 * 16^0 = 46687_{(10)}$

Octal number to Decimal number:

1) $630_{(8)}$ - $6 * 8^2 + 3 * 8^1 + 0 + 4 * 8^{-1} = 384 + 24 + 0 + 0.5 = 408.5_{(10)}$

Binary number to Octal number:

1) $(\underbrace{101}_{5}\ \underbrace{100}_{4}\ \underbrace{011}_{3}\ \underbrace{010}_{2}\ \underbrace{111}_{7}\ .\ \underbrace{111}_{7}\ \underbrace{100}_{4}\ \underbrace{000}_{0}\ \underbrace{110}_{6})_{(2)} = 54327.7406_{(8)}$

2) $(\ \underbrace{11}_{3}\ \underbrace{100}_{4}\ .\ \underbrace{1}_{1}\ \underbrace{001}_{1})_{(2)} = 34.11_{(8)}$

Binary number to Hexa Decimal number:

1) $(\ \underbrace{0101}_{5}\ \underbrace{1111}_{F}\ \underbrace{0001}_{1}\ \underbrace{1010}_{A}\ .\ \underbrace{1110}_{E}\ \underbrace{0011}_{3})_{(2)} = 5F1A\ .\ E3_{(16)}$

2) $(\ \underbrace{101}_{5}\ \underbrace{1010}_{A}\ \underbrace{1110}_{E}\ .\ \underbrace{1100}_{C}\ \underbrace{0010}_{2})_{(2)} = 5AE\ .\ C2_{(16)}$

HISTORY OF C

- ‘C’ language was evolved from two previous languages. BCPL and B.
- ‘C’ is developed by Dennis Ritchie at bell laboratories and was originally implemented on a PDP-11 computer in 1972.
- C uses many of the important concepts of BCPL and B while adding data typing and other powerful features.
- C initially became widely known as the development language of the UNIX operating system.
- Today all new major operating systems are written in C and C++.
- ‘C’ is a hardware independent. With careful design it is possible to write C programs that are portable to most computers.

CHARACTERISTICS OF C-LANGUAGE

1. **MODULARITY**: means ability to break down large modules into sub-modules. The programming language which supports modularity is called as modular programming language.
2. **EXTENDIBILITY**: ability to extend the already existing software by adding new features to it. C software can be extended.
3. **PORTABILITY**: it is the ability to install (port) the existing software in different platforms. C software can be installed in any platform.
   Ex: - Windows platform, Linux platform.
4. **SPEED**: programs written in C are very efficient and fast. This is due to variety of data types and operators available in C. There are only 32 keywords in C and they form the basic building of C-languge.

WHY C LANGUAGE CALLED MIDDLE LEVEL LANGUAGE ?

- Though ‘C’ is a high level language, it is often referred to as **middle level language** because programs written in ‘C’ language run at the speeds matching to that of programs written in assembly language.
- The speed of the ‘C’ language programs is very fast. That’s the reason ‘C’ language is used in system software’s (programs interact with the hardware e.g.. device drivers and operating systems).

GENERAL FORMAT OF A 'C' PROGRAM

```
Documentation section

Link section

Definition section

Global declaration section

main() function section

{

        Declaration part

        Executable part

}

Sub-program section

Function1( );

Function2( );

.

.

Function n( );
```

Documentation Section : To increase the readability of the program the programmer should write the appropriate explanation regarding the statements in the program. These explanatory lines are known as COMMENTS and are not executed by the compiler. Comments are written as given below:-

/* …………………comments………………..*/

Comments can be written anywhere in the program.

Link Section : This section contains the header files which contains library functions.They can be declared as:-

#include< header-file>

By writing a statement in the format as shown above, we are calling the compiler to include all the respective functions present in a header file into our program.

Ex: #include<stdio.h> /* this header filer includes the printf() and scanf()*/

#include<conio.h> /* this header file includes clrscr() and getch() */

Definition section : This section is to define constants in the program and their value cannot be changed throughout the execution of the program.

Syntax: #define constant_name constant_value;

Ex: #define **PI 3.14**;

/* In the program wherever we are using **PI** the constant value 3.14 will be replaced and cannot be changed. */

Global declaration section : The variables declared above main() are known as global variables and these variables can be accessed anywhere in the program.

main() :

- Execution of every c-program starts with main() function.
- It must be written in lower-case letters.
- There must be one and only one main() in a c-program
- Any program in c cannot be executed without main() function.

Curly braces:

- { indicates the beginning of the program or a function.
- } indicates end of the program or function.

Declaration part: In this section, variables of different data types are declared and may be initialized.

Syntax: datatype variable name;

Ex:- int a; /* variable 'a' of integer datatype is declared */

Executable part: The sequence of steps to be executed (logic), in order to solve a particular program will be written here and the compiler will process the input data according to the steps written by the programmer and displays the output .

Sub-program section : The series of tasks that are to be performed by the main() are divided into individual modules / sub-programs which are known as FUNCTIONS. Each function will perform a particular task and returns a value to main() if specified by the programmer.

BASICS OF TYPICAL 'C' PROGRAM DEVELOPMENT ENVIRONMENT

C programs typically go through six phases to be executed .These are

- **Edit**
- **preprocess**
- **compile**
- **link**
- **load** and
- **execution**

All these phases should be written in the same sequence

Edit: The first phase consists of editing a file. This is accomplished with an editor program. The programmer types a C program with the editor and makes corrections if necessary, then storing the program on a secondary storage device such as a disk. C program file names should end with the **.C** extension.

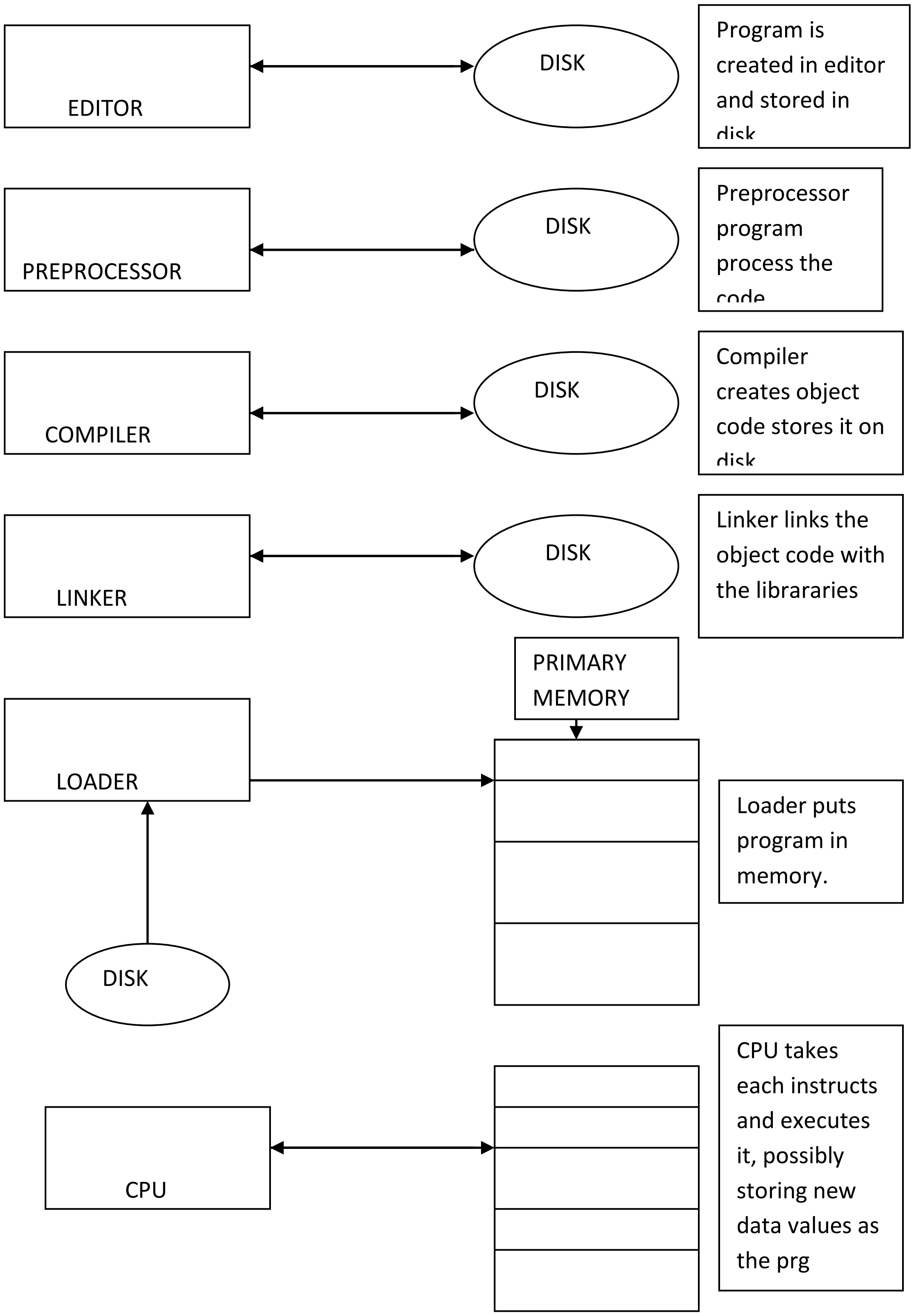
Preprocess: The preprocessor is automatically invoked by the compiler before the program is converted to machine language. A preprocessor program executes automatically before the compiler's translation phase begins.

Compile: Next, the programmer gives the command to compile the program. The compiler translates the C program into machine language code (also referred to as object code).

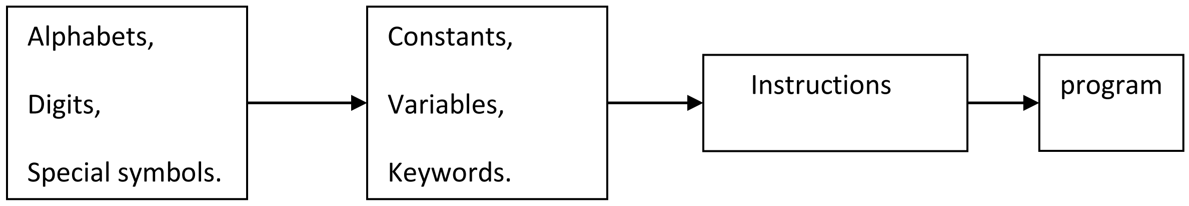
Link: Linker links the object code with the required functions to produce an executable code.

Load: The next phase is loading. Before a program can be executed, the program must first be placed in memory. This is done by the loader, which takes the executable code from disk and loads in to the memory. Additional components from shared libraries that support the program are also loaded by the loader.

Execution: Finally, the computer under the control of the CPU executes the program one instruction at a time. After executing the program we get the required output.


EDITOR
DISK
Program is created in editor and stored in disk
PREPROCESSOR
DISK
Preprocessor program process the code
COMPILER
DISK
Compiler creates object code stores it on disk
LINKER
DISK
Linker links the object code with the librararies
PRIMARY MEMORY
LOADER
Loader puts program in memory.
DISK
CPU
CPU takes each instructs and executes it, possibly storing new data values as the prg

STEPS IN LEARNING C-LANGUAGE :



CHARACTER SET IN C-LANGUAGE :

A Character denotes any alphabet, digit or special symbol used to represent information. There are 255 characters defined in c. Every character is represented with an **ASCII** value. ASCII stands for AMERICAN STANDARD CODE FOR INFORMATION INERCHANGE.

CHARACTERS	**SYMBOLS**	**ASCII VALUES**
ALPHABETS	A,B,C…………..Z a,b,c,……………z	65-90 97-122
DIGITS	0,1,2,3,………..9	48-57
SPECIAL SYMBOLS	~,!,@,#,$,%,^,&,*,(,),_,+, |,`,..,>,<,=,-,?,/,\,},{,[,] :,;,, etc	0-47,58-64,91-96 123-127

C – TOKENS

In a passage of text, individual words and punctuation marks are known as tokens. Similarly, in C , the smallest individual units are called as **C-TOKENS.**

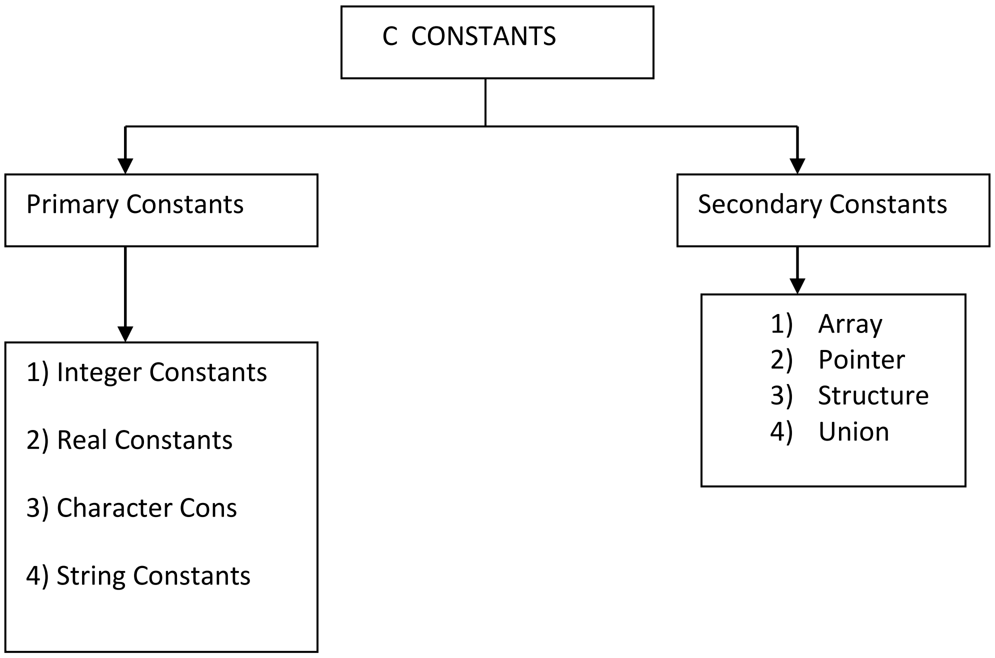
There are six types of c-tokens:-

- Keywords
- Identifiers (variables)
- Constants
- Special symbols
- Strings
- Operators

CONSTANTS: are referred to fixed values whose value do not change during the execution of c –program

C constants are divided into two categories:

(1) Primary constants
(2) Secondary constants



INTEGER CONSTATNTS

a) An Integer constant must have at least one digit.
b) It must not have a decimal point.
c) It can be either positive or negative.
d) If no sign precedes it considers as positive no.
e) No commas or blanks are allowed within an integer constant.
f) The allowable range for integer constants is -32768 to 32767.

Valid-Ex:	426	Invalid Ex:-	546.89 (decimal point)
	+789		10,000 (comma is present)
	-3456		20 000(blank is present)

REAL CONSTANTS

a) A Real constants must have at least one digit.

b) It may or may not have a decimal point.

c) It can be either positive or negative.

d) If no signs precede it considers as positive no.

e) No commas or blanks are allowed within a real constant.

EX: 325.345

-345.322

CHARACTER CONSTANTS

A character constant is a single alphabet, a single digit or a single special symbol enclosed with in **‘ ’**(single inverted commas).

a) The maximum length of a character constant can be 1 character. If more than 1 characters it is not a character constant.

Valid Examples:	‘a’	Invalid :	‘ab’
	‘1’		‘34’
	‘%’		‘2a’
	‘&’		

Note: Character constant stored in 1 byte

STRING CONSTANTS:

A single character or group of characters enclosed in **" "**(double quotes) is called as string constant.

Examples: "a" "ab" "Hello" "2345"

NOTE: String constants stored in 2 bytes.

REPRESENTATION OF NUMBERS

INTRODUCTION:

Language is used for communication.

Language → HLL → english-like language/human understanding language.

➔ LLL → machine language. (binary format ie, 0 and 1)

1. The english like language which we use may consists of:-
   a) Alphabets,
   b) Numbers,
   c) Special symbols.
2. Each of the above is assigned with a specific number called as ASCII code.
   Ex: - a → 97, A → 65, etc.
3. Whatever the code, we write on system is converted to binary format(0,1).
4. All these binary format are stored in <u>registers.</u>
5. Register is a collection of <u>flip-flops.</u>
   Flip-flop is the smallest cell, which can hold a value.(0→ off state, 1→ on state).
   Ex: - 4-bit register is a combination of 4 flip-flops.

- Number representation in binary format, can be expressed in 2ways:
  a) FIXED-POINT REPRESENTATION,
  b) FLOATING-POINT REPRESENTATION.

FIXED-POINT REPRESENTATION: - means the position of decimal point is fixed.

Ex: - 0.5, 0.1, etc.

FLOATING-POINT REPRESENTATION: - means the position of decimal point is changing as per requirement.
Ex: - 1/3 =0.3333 = $3.333*10^{-1}$ =$33.33*10^{-2}$ etc.

<u>FIXED-POINT REPRESENTATION</u>

- If in a register (r1), if the decimal position is fixed at <u>right most position</u> → value is an <u>integer.</u>
- If in a register (r1), if the decimal position is fixed at <u>left most position</u> → value is an <u>fraction.</u>
  Ex: - 10.25

R1	R2
10.	.25
Integer	Fraction

- The representation / manipulation in arithmetics or any processing of data in a register(computer) is done in the following manner.
  a) Whenever we give a number without representing its sign, by default it is positive.
  b) As, in computer, it performs only addition.(-, *, /, % operations done only in form of addition).

  Ex: - A + B = A + B.

  A – B = A + (-B).

So,we need to have separate representation for positive and negative numbers.

REPRESENTATION OF THE POSITIVE NUMBERS

+ 2 = 0 0 1 0 (4-bit repn).

REPRESENTATION OF THE NEGATIVE NUMBERS

1. Representating a negative number or a number in negative form is of 3 types: -

- SIGNED-MAGNITUDE REPN:
  The magnitude is represented in the same form and sign bit is '0' for positive and '1' for negative.

Ex: -	+ 2	=	0	0 1 0
	-2	=	1	0 1 0

  When we go for signed repn, left most bit represents sign and remaining bits give magnitude.
  Ex: - 4-bit repn

Unsigned	:	highest value	=	15	=		1 1 1 1
Signed	:	highest value	=	7	=	0/1	1 1 1

- SIGNED-1'S COMPLEMENT REPN:
  As, we need to have separate repn for negative and positive numbers, we take 1's complement form for negative numbers.

+ 2	=	0	0 1 0	
-2	=	1	1 0 1	(1's complement form – changing 1 to 0 and vice-versa).

- SIGNED-2'S COMPLEMENT REPN:
  When we have arithmetic calculation for zero, we need to have the same repn for +0 and -0. So, to have same repn for both, we take 2's complement form for negative numbers.

Ex: -	+2	=	0	0 1 0	
	-2	=	1	1 0 1	(1's complement repn)
				+ 1	
	-2	=	1	1 1 0	(2's complement repn)

<u>ARITHMETIC ADDITION: -</u>

3 + 2 = 5;

		Sign	magniutde	
3	=	0	0 1 1	
2	=	0	0 1 0	
5	=	0	1 0 1	(0 – positive, 1 0 1 – 5)

ARITHMETIC SUBTRACTION: -

+3 - 2 = +1;

		Sign	magniutde	
+3	=	0	0 1 1	
-2	=	1	1 1 0	(take -2 in 2’s complement repn)
+1	=	0	0 0 1	(0 – positive, 0 0 1 - 1)

Carry is discarded, as we are taking 4-bit register.

Whenever we add, 2 positive numbers or 2 negative numbers, the resultant answer will be above the capacity of resultant register.

Ex: - 6 + 7

6	=	0	1 1 0	
7	=	0	1 1 1	
-5	=	1	1 0 1	(carry from magnitude is added to the sign bit).

So go for 8-bit register.

6	=	0	0 0 0 0 1 1 0
7	=	0	0 0 0 0 1 1 1
13	=	0	0 0 0 1 1 0 1

- so, resultant register should be high enough to hold the result including its sign.
- This carry from the magnitude bit into sign bit is called as “overflow”.
- To overcome, overflow situation, we need to take the values according to registers capacity.

Variables:

Variable names are the names given to memory location. A variable is a location in the memory that can hold a value. An entity that may change during execution of a program is called a variable. These locations contain integer/character constants.

Ex: -

$$\begin{tikzcd} \text{int} \arrow[d] & \text{n;} \arrow[d] \\ \text{Data type} & \text{variable} \end{tikzcd}$$

RULES FOR DECLARING VARIABLES:

1. A variable name is any combination of alphabets, digits or underscore
2. First character in the variable name must be an alphabet or underscore.
3. A Variable name should not be a keyword.
4. White spaces are not allowed.
5. No other characters except underscore (_) should be used.
   Ex:-

int	n ;	allowed
float	area ;	allowed
int	area of circle;	not allowed
int	area_of_circle;	allowed
int	char ;	not allowed
float	int ;	not allowed

Keywords:

In 'C' every word is classified into either a keyword or an identifier(refer name of variable). All keywords have fixed meaning and this cannot be changed.

Keywords are reserve words whose meaning has been explained to the system.(i.e., C compiler). System defined words can be called as 'keywords'. Keywords canno0t be used as variable names.

- Keywords serve as a basic building block for program statement.
- All the keywords must be written in lower case letters.
- There are 32 keywords available in 'C' language.
  They are:

auto	else	long	switch
break	enum	register	typedef
case	extern	return	union

char	float	short	unsigned
const	for	signed	void
continue	goto	sizeof	volatile
default	if	static	while
double	int	struct	do

- main() is not a keyword.

Arithmetics in C:

‘c’ evaluates arithmetic expressions in a sequence determined by the following rules of precedence:-

- ➢ Expressions in pair of parenthesis are evaluated first. If there are more than one pair of parenthesis, then, innermost parenthesis expressions are evaluated first.

  Ex:

  (10 (5 (6 + 2)))

  This expression is evaluated first
- ➢ Multiplication, division and modulus operations are evaluated next. If an expression contain several multiplication, division and modulus operations , then evaluation starts from left and then proceeds to right.
- ➢ Addition and subtraction operations are evaluated at last. If many + and – operators are there in an expression, then evaluation proceeds from left to right.

Example :-

Evaluate the expression 2*((9/3)+4*(5-2))

= 2 * (($\underline{9/3}$) + 4 * (5-2))

= 2 * (3 + 4 * ($\underline{5-2}$))

$= 2 * (3 + \underline{4 * 3})$

$= 2 * (\underline{3 + 12})$

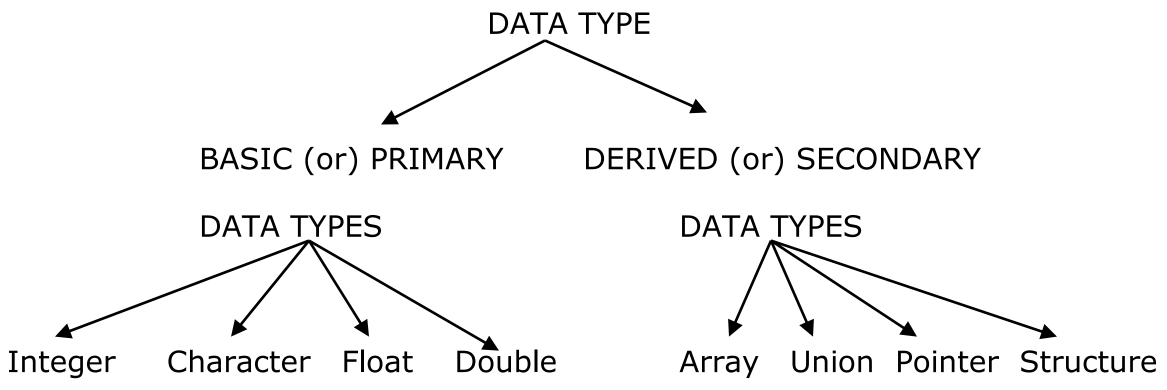
$= \underline{2 * 15}$

$= 30$

MATHEMATICAL EXPRESSION	C – EQUALENT FORM
a+b	a+b
$\frac{x * y}{z}$	(x * y) / z
$\frac{(a+b)^2}{(a-b)^2}$	((a+b) * (a+b)) / ((a-b) * (a-b))
$2x^2 + 3y + 1$	2*x*x + 3*y +1

DATA TYPES IN C

There are various kinds of data types available in c that allows programmer, to use appropriate data type for the program.


DATA TYPE
BASIC (or) PRIMARY DATA TYPES
DERIVED (or) SECONDARY DATA TYPES
Integer
Character
Float
Double
Array
Union
Pointer
Structure

DATA TYPES	NUMBER OF BYTES	FORMAT SPECIFIER	RANGE
Signed Char	1	%c	-128 to 127
Unsigned Char	1	%c	0 to 255
Short Signed Int	2	%d	-32768 to 32767
Short Unsigned Int	2	%u	0 to 65535
Long Signed Int	4	%ld	-2147483648 to

			2147483647
Long Unsigned Int	4	%lu	0 to 4294967295
Float	4	%f	-3.4e38 to 3.4e38
Double	8	%lf	-1.7e308 to 1.7e308
Long Double	10	%Lf	-1.7e4932 to 1.7e4932

TO PRINT TEXT IN C :

- In order to print text or values in c-language we printf() statement.
- This printf() statement in defined in a header file called stdio.h (standard input output)
- **Syntax for printf() statement is :**
  printf(" text to be displayed");
- In order to print the value of a variable printf() should be written in the following way: **printf(" format specifier " , variable);**

TO ASSIGN VALUES TO VARIABLES IN C :

Two ways of assigning values to variables in he program is:

1. Initializing the value in the program itself.

   Ex: int x = 20;

2. Assigning values at run-time (using scanf( ) statement)

SYNTAX OF SCANF STATEMENT :

Scanf() statement is used to assign values at run-time i.e., at the time of executing the program.

Syntax for scanf() is :

scanf (“format specifier “, address of variable);

example:

```
int  x;
scanf(“%d”,&x);
printf(“%d”, x);
```

<u>EVALUATION OF EXPRESSIONS:</u>

An arithmetic expression is a combination of variables, constants and operators arranged as per the syntax of the language. Every arithmetic expression is evaluated according to the precedence rules. Precedence rules tells to compiler that while executing expressions, what operators are executed first, what are their executions and what is result.

The basic evaluation procedure includes two or more left-to-right passes through the expression. During the first phase the higher priority operators (if any) are evaluated. In next phase the next level operators are applied. This continues upto last level.

The order of precedence for different operators is :

Precedence	**Execution Order**
( ),[ ], ->	left-to-right
++,--	right-to-left
!	right-to-left
Unary minus(-)	right-to-left
Sizeof	right-to-left
*, /, %	left-to-right
+, -	left-to-right
<<,>>	left-to-right
< ,<=, >, >=	left-to-right
== ,!=	left-to-right
&&	left-to-right
||	left-to-right
?:	right-to-left
=,%=,*=,/=,+=,-=	left- to-right

OPERATORS IN C-LANGAUGE

OPERATOR: An operator is a symbol which instructs the computer to perform a certain task. The operand is a value on which the operation is performed.

OPERAND : A variable or constant on which the operation is performed.

Ex: a + b = c [here, a,b,c are known as operands and +,= are known as operators.]

An operator may either Unary or Binary.

Unary operator means which act on only single operand. Ex: a++

Binary operator means which acts on two operands.   Ex: a+b

The types of operators present in c are:-

1. Arithmetic operators
2. Assignment operators
3. Bitwise operators
4. Conditional operators
5. Increment & Decrement operators
6. Logical operators
7. Relational operators
8. Special operators

1. **Arithmetic operators:**

The different arithmetic operators are:

+ (Addition), - (Subtraction), * (multiplication), / (Division) and % (Modulus)

Here + & - act as both Unary and Binary.

Any operation between two integers yields result as an Integer.

Any operation between two floats yields results as a float.

Any operation between an integer and float results in float value.

i.e. If two operands are of different type, then the result is larger data type among these two.

Difference between division and modulo division is that :

Consider:

445 / 10 = 44

440 % 10 = 5

When we divide 445 with 10 its quotient will be 44 which is result or normal division operation.

Where as for modulo division the result of integer division will be the result of modulo division. When we divide 445 with 10 remainder is 5 which is result of modulo division.

2. **Relational operators:**

These are used for comparison. These are used for decision making.

An expression containing a relational operator is termed as a relational expression.

The value of a relational expression is either zero or one. If the value is zero then it is false, if the value is nonzero then it is true.

The different relational operators are:

< (less than), <= (less than or equal to), > (greater than), >= (greater than or equal to), == (is equal to), != (not equal to)

A simple relational expression contains only one relational operator and takes the following form: **ae1 relational operator ae2**.

3. **Logical operators:**

&& (Logical AND), || (Logical OR), ! (Logical NOT)

&&, || are used when we want to test more than one condition and make decisions. An expression which combines two or more relational expressions is termed as a logical expression. Logical expressions yields zero or nonzero according to the truth table.

OP1	OP2	&&	||
Nonzero	Nonzero	Nonzero	Nonzero
Nonzero	Zero	Zero	Nonzero
Zero	Nonzero	Zero	Nonzero
Zero	Zero	Zero	Zero

OP	! NOT
Nonzero	Zero
Zero	Nonzero

4. **Assignment operators:**

These are used to assign the result of an expression to a variable.

The general form is: **V op = exp;**

Here V is a variable expression is an expression and op is a binary arithmetic operator.

V op = exp is equal to v = v op (exp);

Ex: a+=10; is equal to a=a+10;

These are

+=, -=, *= and %=

These are also called **shorthand operators**. The advantages are:

- The left side operand need not be repeated and therefore easier to write.
- The statement is more concise and easier to read.
- The statement is more efficient.

5.<u>Increment and Decrement operators:</u>

C has 2 very useful operators which are generally not found in other languages.

These are **++ (Increment)**, **- - (decrement)**

++integer_variable indicates **pre-increment operation** in which, first the value of variable is incremented and then specified instruction is performed.

integer_variable++ indicates **post-increment operation** in which, first the specified instruction is performed and then the value of variable is incremented .

--integer_variable indicates **pre-decrement operation** in which, first the value of variable is decremented and then specified instruction is performed.

integer_variable-- indicates **post-decrement operation** in which, first the specified instruction is performed and then the value of variable is decremented.

Ex:-

int a=10 , b ;

Pre -increment	**Post - increment**
b = a++;	b = ++a ;
b = a ;	a = a + 1 ;
a = a + 1 ;	b = a ;
b = 10 , a = 11 ;	b = 11 , a = 11 ;

Pre - decrement	**Post - decrement**
b = a-- ;	b = --a ;
b = a ;	a = a - 1 ;
a = a - 1 ;	b = a ;
b = 10 , a = 9 ;	b = 9 , a = 9 ;

6. **Conditional or Ternary operators:**

A ternary operator pair “?:” is available in C to construct conditional expressions of the form

exp1?exp2:exp3;

Here exp1 is evaluated first. If it is nonzero (true), then the expression exp2 is evaluated and becomes the value of the expression. If exp1 is false, exp3 is evaluated and its value becomes the value of the expression. Only one of the expressions (either exp2 or exp3) is evaluated.

7. **Bitwise operators:**

C has a distinction of supporting special operators known as bitwise operators for manipulation of data at the bit level. These operators are used for testing the bits, or shifting them right or left. Bitwise operators may not be applied to float or double. These are & (bitwise AND), | (bitwise OR), ^ (bitwise ExOR), <<(shift left), >> (shift right), ~ (complement).

Bit1	Bit2	&	|	^
0	0	0	0	0
0	1	0	1	1
1	0	0	1	1
1	1	1	1	0

Bit	~
0	1
1	0

Examples:

a=5 b=11

a&b	a|b	a^b
00000101	00000101	00000101
00001011	00001011	00001011
--------------	--------------	--------------
00000001	00001111	00001110

~a=~(00000101)

11111010

<<:Left Shift

Bits will be moved towards left by one position ,0 will be added at the right most bit and left most bit will be lost.

a<<2

5<<1= 00001010

<<2= 00010100

>> : Right Shift

Bits will be moved towards right by one position ,0 will be added at the left most bit and right most bit will be lost.

a>>2

5>>1=00000010

>>2=00000001

8. **Special operators:**

These are comma operator, sizeof() operator,Ampersand(&). Comma(,) operator can be used to link the related expressions together.

Sizeof operator returns the number of bytes required for the operand storage. Ampersand symbol is used to denote the address of a variable.

Example :

```
int  x;

printf("%d", sizeof(  x ) );  // output will be 2
```

TYPE CONVERSION:

When an operator has operands of different types, they are converted to a common type according to a small number of rules. In general, the only automatic conversions are those that convert a ``narrower'' operand into a ``wider'' one without losing information, such as converting an integer into floating point in an expression like `f + i`. Expressions that don't make sense, like using a `float` as a subscript, are disallowed. Expressions that might lose information, like assigning a longer integer type to a shorter, or a floating-point type to an integer, may draw a warning, but they are not illegal.

If either operand is `long double`, convert the other to `long double`.
- Otherwise, if either operand is `double`, convert the other to `double`.
- Otherwise, if either operand is `float`, convert the other to `float`.
- Otherwise, convert `char` and `short` to `int`.
- Then, if either operand is `long`, convert the other to `long`.

explicit type conversions can be forced (``coerced'') in any expression, with a unary
operator called a `cast`. In the construction

(*type name*) *expression;*

the *expression* is converted to the named type by the conversion rules above. The precise meaning of a cast is as if the *expression* were assigned to a variable of the specified type.
EX:

```
int  y;
y = (int) 9.8;
printf("%d", y);  // output will be 9 that is fraction part .8 is truncated.
```

CONTROL STRUCTURES IN C-LANGUAGE:

Some times in a program it will be necessary to :-

- Select a set of statements from several alternatives
- Skip certain statements depending on some conditions in the program and continue execution from some other point
- Repeat the execution of certain statements depending on the truthness of the condition.

Under such conditions, the control statements are used. There are three types of control structures:-

- ➔ Sequence control structure
- ➔ Selection control structure
- ➔ Repetition control structure

SEQUENCE CONTROL STRUCTURE

In this structure, the statements are executed in the same order in which they appear in the program.

Ex:-
```
#include<stdio.h>
main()
{
    printf("hello world");
    printf( " welcome to c");
}
```

In the above program both the statements are executed in the order in which they are written.

SELECTION CONTROL STRUCTURE :

These statements are also called as CONDITIONAL CONTROL STATEMENTS.

In some situations, it is necessary to check the condition to make the decision.

This involves performing a **logical test**. This results in either **true** or **false**.

Depending on the truthness of condition ,some statements are executed.

Thus, this is known as **conditional execution.** C provides various kinds of control statements. They are as follows:-

- **if**-statement
- **if-else** statement
- **else-if** ladder
- **nested-if** statement
- **switch** statement

if statement:-

It is used to execute a statement or set of statements conditionally. It is a simple 'if' statement.

Syntax:-

```
if(condition)

{

        Statements;

}

  Statement-x;
```

The logical condition will be tested. It results in either true or false. If the condition is true the statements are executed or else not executed and the control is transferred to the next executable statement i.e, statement-x.

The simple-if statement is demonstrated by a sample program given below:-

```
main()
 {
      int  a;
      printf(" enter value of a ");
      scanf("%d",&a);
      if( a>10)
      {
             printf(" a is greater than 10");
      }
      printf( " if statement not executed");
}
```

In the above program:-

- ➢ If the value of 'a' is greater than 10 then the statement in if block is executed.
- ➢ If the value of 'a' is less than 10 then the if block will not be executed and the control transfers to next executable statement i.e, printf(" if statement is not executed").

if-else statement:

The if/else structure performs an action if the condition is true and performs a different action if condition is false. The if/else structure is called **double-selection structure** because it selects between two different actions.

Syntax:

```
if(condition)

{

        Statement-1;

}

else

statement-2;
```

If the condition is true, then statement-1 is executed, other wise; the control moves to else part and statement-2 is executed.

If-else is demonstrated by the following program:-

/* program to find whether a number is even or odd */

```
#include<stdio.h>
#include<conio.h>
void main()
{
   int a;
   printf("Enter the no\n");
   scanf("%d",a);
   if(a%2==0)
   printf(" %d is even");
   else
}
```

OUTPUT :

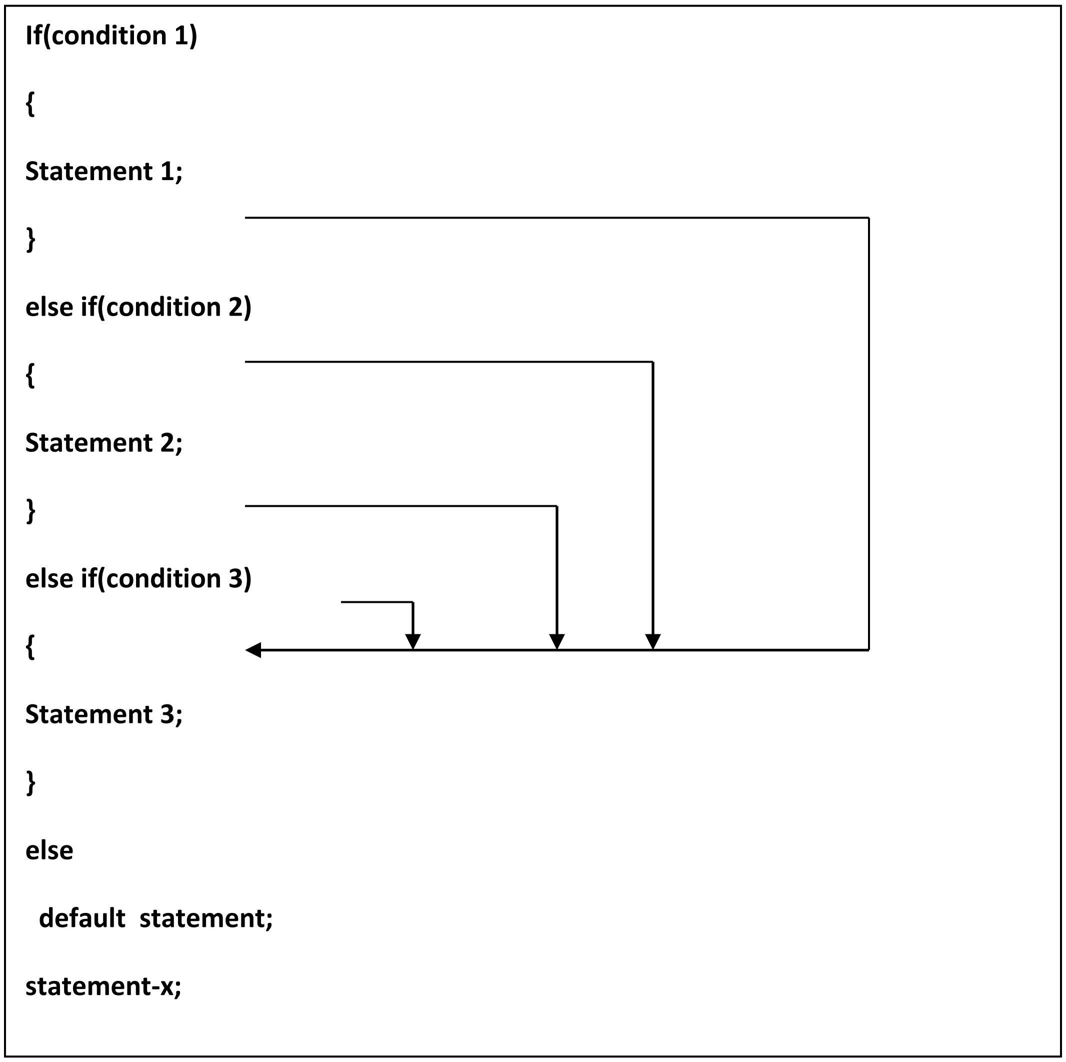
```
Enter the no   21
21 is odd
```

Value of 'a' is given as 21…and 21%2 is not equal to zero…so, the control is transferred to else part leaving if block.

Else–if ladder :

There is another way of putting **if**s together when multipath decisions are involved. A multipath decision is a chain of **if**s in which the statement associated with **else** is an **if**

Syntax of else-if ladder is :-



```
If(condition 1)
{
Statement 1;
}
else if(condition 2)
{
Statement 2;
}
else if(condition 3)
{
Statement 3;
}
else
  default  statement;
statement-x;
```

Example:

/***program to calculate roots of quadratic equations***/

```
#include<stdio.h>
#include<math.h>
main()
{
  int a,b,c;
  float r1,r2,d;
  printf("Enter the coefficients of equation\n");
  scanf("%d %d %d",&a,&b,&c);
  d=(b*b)-(4*a*c);
  if(d<0)
  {
     printf("The roots are imaginary");
  }
  else if(d==0)
  {
    printf("Roots are equal\n");
    r1=-b/(2*a);
    r2=r1;
    printf("r1=%f   r2=%f",r1,r2);
  }
  else if(d>0)
  {
     printf("Roots are real");
     r1=(-b+sqrt(d))/(2*a);
     r2=(-b-sqrt(d))/(2*a);
     printf("r1=%f r2=%f",r1,r2);
  }
}
```

Nested –if statement :

Writing one "if" statement within another "if" statement is known as nested-if statement. When there are more than one condition to be tested then nested-if structure is used. The syntax for nested-if statement is as follows:-

```
if(condition-1)    /* if condition-1 is true then condition-2 is tested*/
{
        if(condition-2)  /* if condition-2 is true then statement-1 is executed*/
        {
                Statement-1;
        }
        else    /* if condition-1 is true and condition-2 is false the statement-2  is
                                        executed */
        Statement-2;
}
else  if(condition-3)  /* if both condition1 and 2 are false then control comes
                                to else-if block and condition-3 will be tested.if it is
{                               true statement- 3 is o/p or else statement-4 is o/p*/
        Statement-3;
}
else
Statement-4;
```

Nested-if statement is demonstrated by a sample program:-

```
/*Program to find greatest of three no’s */
#include<stdio.h>
main()
{
   int a,b,c;
   printf(“Enter the values\n”);
   scanf(“%d %d %d”,&a,&b,&c);
if(a>b)
{
     if(a>c)
      printf(“%d is greater”,a);
     else
       printf(“%d is greater”,c);
}
else
{
     if(b>c)
     printf(“ %d is greater”, b);
}
     else
     printf(“ % d is greater”, c);
}
```

Output : Enter the values 10 54 40   54 is greater

Switch Statement :

The switch selection structure performs one of many different actions depending on the value of an expression. The switch structure is called **multiple-selection structure** because it selects one option among many different options available. The syntax for switch statement will be:-

```
switch(expression)
{
       case   label1  : block-1;
                        break;
       case   label2  : block-2;
                        break;
       case   label3  : block-3;
                        break;
       case   label4  : block-4;
                        break;
       ...........
       ...........
       case   labeln  : block-n;
                        break;
       default  :  block;
                   break;
}
Statement-x;
```

The expression is an integer expression or characters.

Each of the label values should be unique within a switch statement.

block-1, **block-2...** are statements lists and may contain zero or more statements. There is no need to put braces around these blocks. Note that case labels end with a **colon(:)**

When the switch is executed, the values of the expression is successively compared against the values label1, label2.... if a case is found whose value matches with the value of the expression, then the block of statements that follows the case are executed.

The **break** statement at the end of each block signals the end of a particular case and causes an exit from the **switch** statement, transferring the control to the statement-x following the switch.

The default is an optional case. When present, it will be executed if the value of the expression does not match with any of the case values. If not present, no action take place if all matches fail and the control goes to the statement-x.

Ex:-

/* Calculator program using switch case */

```
#include<stdio.h>
 main()
{
  int a,b,c,n;
  printf("Enter two no's\n");
  scanf("%d %d", &a,&b);
  printf("1.Addition\n2.Substration\n3.Multiplication\n4.Division");
  scanf("%d",&n);
  switch(n)
  {
      case 1:c=a+b; break;
      case 2:c=a-b; break;
      case 3:c=a*b; break;
      case 4:c=a/b; break;
      default:  printf("Invalid Number\n");
   }
          printf("%d",c);
         }
```

REPETITION CONTROL STRUCTURE :

C provides three repetition structures

1) while loop
2) do-while loop
3) for loop

A loop is a group of instructions the computer executes repeatedly while some loop-continuation condition remains true.

Repetition control structures are also called as **iterative control structures** or **looping control structures**.

WHILE   LOOP

The basic format of while loop is

```
While(condition)

{

  body of loop

}
```

The while is entry-controlled loop statement. The condition is evaluated and if the condition is true then the body of the loop is executed. After execution of the body, the condition is once again evaluated and if it is true, the body is executed once again. This process of repeated execution of the body continues until the condition finally becomes false and the control is transferred out of the loop. On exit, the program continues with the statements. The braces needed only if the body contains one or more statements.

/* A simple program for while loop */

```
#include<stdio.h>

main()

{

  int a=1;
  while (a>=3)
  {
printf("a=%d",a);
a++;
}
printf("hello");
  }
```

OUTPUT: hello

DO-WHILE LOOP

The general format of do-while is

```
do
{
  body of loop
}
while(condition);
```

The while loop constructs that we have discussed in the previous section makes a test of condition before the loop is executed. Therefore, the body of the loop may not be executed at all if the condition is not satisfied at the very first attempt. On some occasions it might be necessary to execute the body of the loop before the test is performed. Such situations can be handled with the help of the **do** statement.

On reaching the do statement, the program proceeds to evaluates the body of the loop first. At the end of the loop, the condition in the while is evaluated. If the condition is true, the program continues to evaluates the body of the loop once again. This process continues as long as the condition is true. When the condition become false, the loop will be terminated and the control goes to the statement that appears immediately after the while condition.

/* **A simple program for <u>do while</u> loop***/

```
#include<stdio.h>

main()

{

  int a=1;


do

{

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

a++;

} while (a>=3);


printf("Hello");

  }
```

OUTPUT: 1

```
Hello
```

FOR LOOP

The for loop is another entry-controlled loop that provides a more concise loop control structure.

```
for(initialization; test-condition; increment)

{

  Statements;

}
```

The execution of the for statement is as follows:

- Initialization of the control variables is done first, using assignment statements such as i=1 and count=0. The variable i and count are known as loop-control variables.

- The value of the control variable is tested using the test-condition. The test-condition is a relational expression, such as i<10 that determines when the loop will exit. If the condition is true, the body of the loop is executed; otherwise the loop is terminated and the execution continues with the statement that immediately followed the loop.

- When the body of the loop is executed, the control is transferred back to the for statement after evaluating the last statement in the loop. Now, the control variable is incremented using an assignment statement such as i=i+1 and the new value of the control variable is again tested to see whether it satisfied the loop condition. If the condition is satisfied, the body of the loop is again executed. This process continues still the value of the control variable fails to satisfy the test-condition.

Example:

/* **program to demonstrate a simple for loop** */

```
        #include<stdio.h>
              main()
             {
                int a;
                 for(a=1;a<=10;a++)
                 {
                       printf("%d\t",a);
                  }
                  getch();
      }
```

OUTPUT:
```
1    2    3    4    5    6    7    8    9    10
```

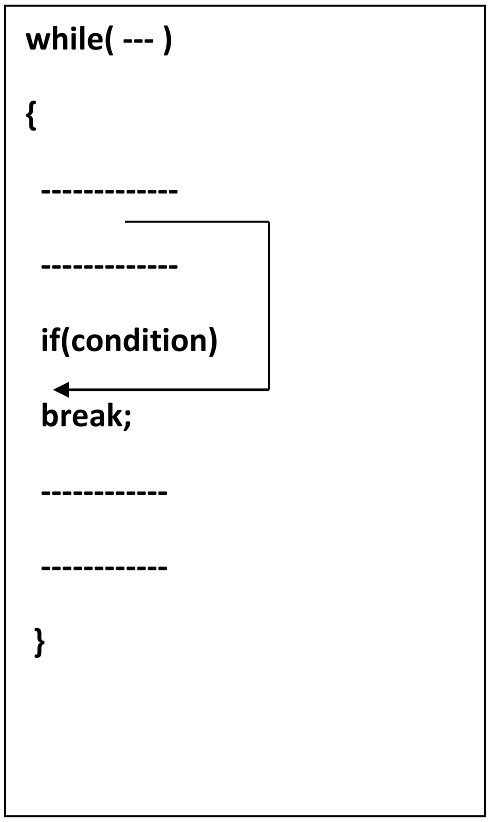
UNCONDITIONAL CONTROL STATEMENTS :

Without testing any condition the floe of program will be altered. There are 3 such statements in c :

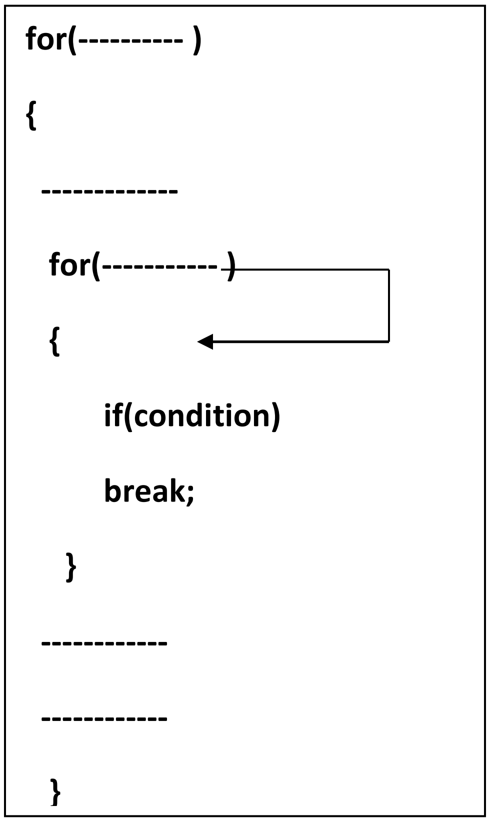
1. break
2. continue
3. goto

BREAK statement :

When the break statement is encountered inside a loop, the loop is immediately exited and the program continues with the statement immediately following the loop. When the loops are nested, the **break** would only exit from the loop containing it. That is, break will exit only a single loop.



```
while( --- )

{

  -------------

  -------------

  if(condition)

  break;

  ------------

  ------------

 }
```



```
for(---------- )

{

  -------------

   for(-----------)

   {

        if(condition)

        break;

     }

  ------------

  ------------

   }
```

Example:

```
#include<stdio.h>
main()
{
      int i;
      clrscr();
      for(i=1;i<=10;i++)
      {
         if(i==5)
          break;
          printf("%d\t",i);
       }
       printf("\nBroke out of the loop at x == %d\n",x);
 }
```

OUTPUT:

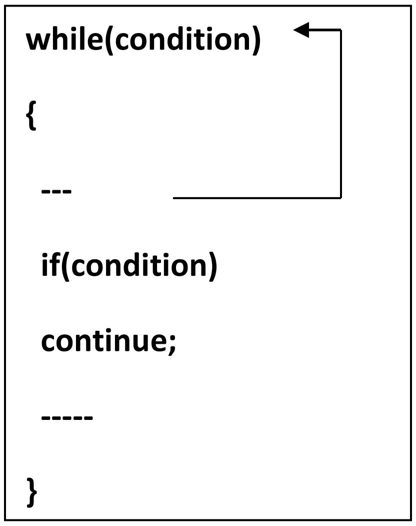
```
1  2  3  4
   Broke out of the loop at x==5
```

CONTINUE statemen :

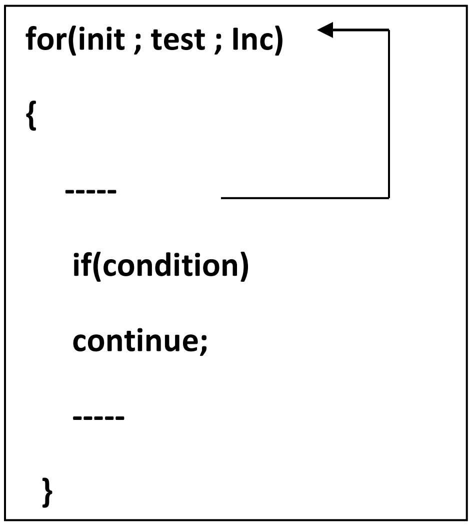
When the continue statement is encountered in the loop, as the name implies, it causes the loop to be continued with the next iteration after skipping any statements in between. The continue statement tells the compiler, "**SKIP THE FOLLOWING STATEMENTS AND CONTINUE THE NEXT ITERATION**". The format of the continue statement is

continue;

In **while** and **do** loops, continue causes the control to go directly to the test-condition and then to continue the iteration process. In the case of **for** loop, the increment section of the loop is executed before the test-condition is evaluated.



```
while(condition)
{
 ---
 if(condition)
 continue;
 -----
}
```



```
for(init ; test ; Inc)
{
   -----
    if(condition)
    continue;
    -----
 }
```

Example:

```
#include<stdio.h>
  main()
  {
      int i;
      for(i=1;i<=10;i++)
      {
           if(i==5)
           continue;
           printf("%d\t",i);
       }
      printf("\nUsed continue to skip printing the value 5",x);
   }
```

OUTPUT:

```
1  2  3  4  6  7   8   9  10
 Used continue to skip printing the value 5
```

GOTO :

It is used to alter the normal flow of control of a program. Goto requires a label in order to identify the place where the branch is to be made.A label is any valid variable name and must be followed by colon. The label is to be placed immediately before the statement where the control is to be transferred.

```
Syntax:   goto label;  (for goto)
           label:
                       {
                               block of statements;  (for label)
                       }
```

ex:

```
void main()
{
        int a=20;
        goto abc;  /* here 'abc' is the name of the label */
        printf("going to goto statement");/*this statement is not executed*/
        abc: printf("%d",a); /*when goto statement executes control comes here*/
}
```

Output: 20

STORAGE CLASSES:

All variables will have not only data type, but also a storage class.

In order to define a variable totally, we need to mention both its data type and storage class.

Storage Class of a Variable defines:

i) Where the variable is stored
ii) Default initial value of variable.
iii) Scope of variable i.e, in which functions the variable is available.
iv) What is a life of a variable i.e, how long will the variable exist.

There are 4 types of storage classes in C:

1. Automatic storage classes
2. Register storage classes
3. Static storage classes
4. External storage classes

Inside **CPU**, 2 types of storage spaces are present
i) memory unit. ii) registers. (limited in number)

a) **Automatic storage classes:**

Features of a variable having an automatic storage class will be:

Storage	→ memory
Default initial value	→ garbage value
Scope	→ Local to the block in which it is defined.
Life	→ Till the control remains within the block, the variable is defined.
Keyword	→ auto

```
include<stdio.h>void
main()
{
auto int i;
printf("%d",i);
}
```

Output: 1011 (unpredictable/unexpected value if differs from compiler to compiler)

Program to demonstrate scope and life of automatic variable:

```
main()
{
 auto int i;
{
        {
                {
                        printf("%d",i);
                }
                printf("%d",i);
        }
}
```

Output: 1 1 1

Reason: in the above program, i is a automatic variable, whose scope is local, to the block, in which it is defined.

There fore when the control comes out of the block, in which the variable is defined. The variable and its value are lost.

```
 main()
{
auto int i=1;
        {
                auto int i=2;
                {
                        auto int i=3;
                        printf("%d",i);
        }
                printf("%d",i);
        }
        printf("%d",i);
}
```

Output:

3 2 1

Reason: Compiler treats 3 i's as totally different variables, since they are defined in different blocks.Once control comes out of the inner most block, the variable i with value 3 is lost. Therefore, i in 2nd printf() statement. Refers to i with value 2. When control comes out of the next inner most blocks i=2 is lost and 3rd pritnf() statement. Refers to i with value i.

b) Register Storage class:

Features of a variable defined under register storage class are:

Storage	→ CPU register
Default initial value	→ garbage value
Scope	→ Local to the block in which variable is defined.
Life	→ Till the control remains within the block, the variable is defined.
Keyword	→ register

A value stored in CPU register, can be accessed faster than a value stored in memeory. If a variable is to be used at many places/ many times in the program, then it would be better to declare its storage class as register.

Disadvantage: CPU registers will be limited in number. If all the register of CPU are busy with some other task, then variable storage class is taken as auto.

```
#include<stdio.h>
mian()
{
register int i;
for (i=1;i<=10;i++)
printf("%d \t ",i);
}
```

Output: 1 2 3 4 5 6 7 8 9 10

c) Static Storage Class:

Features of a variable defined with static storage class are:

Storage	→ Memory
Default initial value	→ Zero
Scope	→ Local to the block in which variable is defined.
Life	→ Till the control remains within the block, the variable is defined.
Keyword	→ static

Difference between static and automatic variables is that they don't disappear, when the function is not active. Their value exists and when control comes back to same function again, the static variables have the same values, they had last time.

Ex:

```
#include<stdio.h>
void increment();
main()
{
increment();
increment();
increment();
}
void inclremnt()
{
static int i=1;
printf("%d",i);
i=i+1;
}
```

Output: 1 2 3

Reason: In the above program, if we declare variable i as integer and static variable the output will be 1 2 3.

Ex:

```
#include<stdio.h>
void increment();
main()
{
increment();
increment();
increment();
}
void inclremnt()
{
auto int i=1;
printf("%d",i);
i=i+1;
}
```

Output: 1 1 1

Reason: In the above program, when I is of automatic storage class, each time increment() is called, it will be re-initialized to 1. When function terminates value i=2 will be lost.

Instead of 'auto' if we use 'static', i will be initialized to 1 only once. During first call of increment 'i' will be 2 and i is of static type, the value will not be lost. Similarly during second function call of increment(), i will be 3.

d) External Storage Class:

Features of a variable defined with External storage class are:

Storage	→ Memory
Default initial value	→ Zero
Scope	→ Global
Life	→ As long as execution of program doesn't come to an end.
Keyword	→ Extern

External variables are declared outside the function, So they are available to all the functions in program.

Ex:

```
#include<stdio.h>
int i=20;
main()
{
extern int j;
printf("%d",i);
printf("%d",j);
}
int j=40;
```

Out put: 20   40

Keyword Extern indicates that variable j is defined some where after or outside the main().

Here i and j are global variable. Therefore they are defined outside function, both enjoy external storage class.

Difference between them is:

extern int j;   → Declaration; initially memory is not registered for it.

Int j=40; → Defination

S.No	**Storage Classes**	**Keyword**	**Storage**	**Default Value**	**Scope**	**Life Time**
1	Automatic	Auto	CPU	Garbage	Local	→ Till the control remains within the block, the variable is defined.
2	Register	Register	Register	Garbage	Local	→ Till the control remains within the block, the variable is defined.
3	Static	Static	CPU	Zero	Local	Persist between diff function calls
4	External	Extern	CPU	Zero	Global	The variable exits through out the execution of the program.