

UNIT - I

Overview of Computers and Programming - Electronic Computers Then and Now – Computer Hardware - Computer Software - Algorithm - Flowcharts - Software Development Method – Applying the Software Development Method.

Types, Operators and Expressions: Variable Names - Data Types and Sizes - Constants - Declarations - Arithmetic Operators - Relational and Logical Operators - Type Conversions - Increment and Decrement Operators - Bitwise Operators - Assignment Operators and Expressions - Conditional Expressions - Precedence and Order of Evaluation

Overview of Computers and Programming

Computer Definition:

- The word computer has been derived from the word “Compute” which means to calculate.
- It is an electronic device which inputs the data, stores the data, processes the data and gives the result accurately at a very high speed according to the instructions provided.

Characteristics of a Computer:

- 1) **Automatic:** Given a job, computer can work on it automatically without human interventions
- 2) **Speed:** Computer can perform data processing jobs very fast, usually measured in microseconds (10⁻⁶), nanoseconds (10⁻⁹), and picoseconds (10⁻¹²)
- 3) **Accuracy:** Accuracy of a computer is consistently high and the degree of its accuracy depends upon its design. Computer errors caused due to incorrect input data or unreliable programs are often referred to as Garbage-In-Garbage-Out(GIGO))-6
- 4) **Diligence:** Computer is free from monotony, tiredness, and lack of concentration. It can continuously work for hours without creating any error and without grumbling
- 5) **Versatility:** Computer is capable of performing almost any task, if the task can be reduced to a finite series of logical steps
- 6) **Power of Remembering:** Computer can store and recall any amount of information because of its secondary storage capability. It forgets or loses certain information only when it is asked to do so.
- 7) **No I.Q.:** A computer does only what it is programmed to do. It cannot take its own decision in this regard.
- 8) **No Feelings:** Computers are devoid of emotions. Their judgment's based on the instructions given to them in the form of programs that are written by us (human beings)

Electronic Computers Then and Now

History of computers:-

The development of computer started over the ages and it took thousands of years to mature, let us consider the development of a computer through various stages.

1. Abacus:-

- Around 30 years before the birth of Christ, they developed an earliest form of counting machine known as abacus.
- An abacus consists of beads divided into two parts which are movable on the rods of the two parts.

2. Napier's Logs and Bone's :-

- John Napier developed the idea logarithm.
 - He used logs to transform multiplication problem to addition problem.
 - He also derive a set of numbering rods know as Napier's bone's, which are used to perform multiplication and divisions.
- 3. Pascal's Adding machine :-**
- Blaise Pascal invites a machine called adding machine. It was capable to perform addition & subtraction.
- 4. Leibnitz calculator :-**
- Gottfried Leibnitz improved by adding machine & constricts a new machine in 1671 that was able to perform multiplication & division as well.
- 5. Babbage's Difference Engine :-**
- Charles Babbage developed a machine called difference engine. These machines calculate logarithmic tables to a high degree of precision.
- 6. Babbage's Analytical Engine :-**
- Charles Babbage designs an analytical engine which begins a real model of modern day computer.
 - He included the concept of central process storage area, memory and input/output.
 - Because of inventions of difference engine and analytical engine, Charles Babbage has been considering as "Father of modern Computers".
- 7. Hollerith's Machine :-**
- Herman Hollerith developed electro mechanical punched card that is used for input, output and storing of instructions.
- 8. Mark-1 :-**
- Howard Aiken contracted an electro mechanical computer named mark-1 which could multiply two 10 digit number in 5-seconds.
- 9. ENIAC :-**
- Electronic Numerical Integrator And Calculator were built by Prof.Eckerit and Mauchly.
 - It used about 19000 vacuum tubes and can perform about 300 multiplications per second.
- 10. EDSAC :-**
- Electronic Delay Storage Automatic Computer was developed by Maurice willies.
 - It has ability to input, output, storing the data.
 - It also able to perform and control arithmetic calculations.
- 11. EDVAC :-**
- Electronic Discrete Variable Automatic Computer was developed by Prof.Eckerit and Mauchly.
 - In this both the data and instruction can be stored in binary form instead of decimal number system.
- 12. UNIVAC:-**
- Universal Automatic Computer developed by Remington.
 - It was capable of performing to access both numeric and alphabetic information.
- 13. UNIVAC-1:-**
- It is the first computer which is used for commercial purpose in 1954.

Generations of Computer

Evolution of Modern Computers from the olden days is known as "Generation of Computers." Generations of computers are broadly classified based on the following characteristics:

- Increasing in storage capacity.
- Increasing in processing speed.

- Increasing reliability.

There are totally 5 generations of computers till today.

First Generation Computers:

Period: (1945-1955)

Technology: Vacuum tubes

The ENIAC was the first computer developed in this generation. It was capable of performing 5000 additions (or) 350 multiplications per second. It uses about 18000 vacuum tubes and it consumes 150 kw/hr.

Limitations:

The limitations of first generation computers are as follows.

- Less operating capacity.
- High power consumption.
- Very large space requirement.
- Produce high temperature.
- Very costly.

Second Generation Computers:

Period: (1955-1965)

Technology: Transistors

- The second generation computers use transistors as the main component in CPU. They are very small in size when compared to vacuum tubes and produce less heat. They are fast and reliable.
- Due to this invention of Transistors the size, maintenance cost, power consumption has decreased.
- During this period magnetic storage devices have been started their development. Because of this, speed and storage capacity has been increased. They are capable to perform 20,000 to 50,000 additions per second.

Third Generation Computers:

Period: (1965-1975)

Technology: Integrated Circuits {IC}

- In this generation the computers used integrated circuits instead of Transistors
- Integrated circuit is a miniature form of an electronic circuit made of silicon and enclosed in a metal package.
- These IC's are called "chips." The cost and size of the computers were greatly reduced. The magnetic disk technology has improved rapidly. It is capable to perform 10 million additions per second.

Fourth Generation Computers:

Period: (1975-1990)

Technology: Very Large Scale Integrated Circuit (VLSI)

- IC's packing nearly 10,000 transistors are grouped in to a single silicon chip known as "microprocessor". The computers which use micro processors are called "Micro Computers".
- Intel Corporation invented the micro processor in the year 1980 with this development the cost of a computer has reduced a lot.

- The floppy disk technology was developed during this generation.

Fifth Generation Computers:

Period: (1990- till date)

Technology: Artificial Intelligence

- Artificial Intelligence is a technique by which we make the computer to think and take decisions in its own.
- These computers are under research.
- Artificial Intelligence can be achieved by means of problem solving, Game playing, and Expert systems.

Types of computers:-

The available computers are classified into two categories, namely

- (i) Based on Size
- (ii) Based on functionality

Classification of Computers based on the sizes:

Depending upon the size and capability of the computers, they also classified in to four types namely.

- (i) Super Computers
- (ii) Main Frame Computers
- (iii) Mini Computers
- (iv) Micro Computers

Super Computers:

- Complex scientific applications like weather forecasting require a large amount of data to be processed. To solve this we use super computers.
- They are most powerful computers. Even though they are costly, they have high storage capacity and very fast.
- It has capable to perform 100 million instructions per second.
- These computers have multiple processor and capable of performing multiple tasks at a time.
- The word-length of super computers is of 64 bits.

Note: The capacity of processing number of bits at a time is called word-length.

Main Frame Computers:

- Main Frame Computers are also referred to as large scale general purpose computers.
- They have more memory capacity and have very high processing speed.
- It has capable to perform 10-30 MIPS.
- The word-length of the main frame computers is 32-64 bits.
- They are used in large business Enterprises, Government Agencies, Universities, etc where huge volume of data is required.

Mini Computers:

- The Mini Computers are third generation computers.
- They are used in applications that do not required very high speed (or) large memory capacity for storage.
- Its operating speed is low when compared with main frame computers.
- The word-length is of 12-32 bits, they are also called “Mid-Range Computers”.

Micro Computers:

- A Micro Computer is a small and low cost computer that is built around micro processors for storing and processing.
- They are also called personal computers.
- The word-length of these computers is 4-16 bits.
- They are generally used for office applications.
- There are different models of these computers. Some of them are desktop, laptop, palmtop.

Classification of Computers based on Functionality:

The computers depending on the kind of functionality, they are classified in to three types namely.

- (i) Analog Computers
- (ii) Digital Computers
- (iii) Hybrid Computers

Analog Computers:

- In Analog computers continuous quantities are used.
- Computations are carried out physical quantities such as voltage, temperature.
- The devices that measure such quantities are analog devices.
- Analog computers are mostly used in Engineering and Scientific Applications.

Digital Computers:

- The digital computers work upon discontinuous data.
- They convert the data in two digits [0, 1] and all operations carried out on these digits at extremely fast rates.
- Digital computers are much faster and more accurate than Analog computers.
- These computers are used in business and scientific applications.

Hybrid Computers:

- These computers utilize the best quantities of both the digital and Analog computers.
- These computers are best used in hospital where analog part is responsible for measuring patient heart beat and then the operation is carried out in digital fashion to monitor patient status.
- Thus in these computers some calculations takes place in analog manner and rest of them takes place in digital manner.

A computer is a system made of two major components.

- **1. Hardware**
- **2. Software**

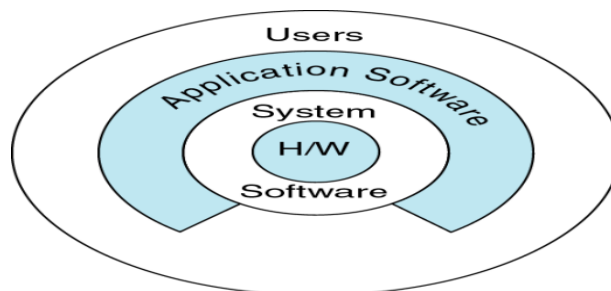


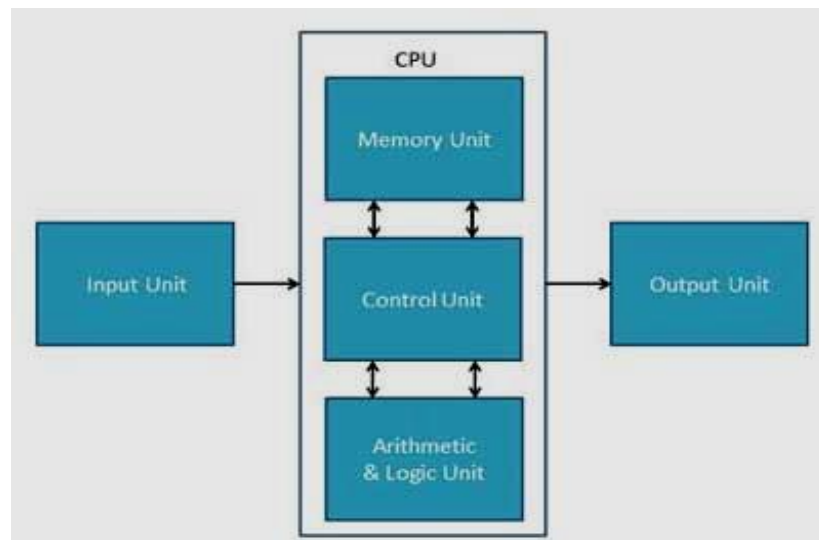
Fig.1: Computer Components

Computer **hardware** is the collection of all the parts you can physically touch. Computer **software**, on the other hand, is not something you can touch. Software is a set of instructions for a computer to perform specific operations. You need both hardware and software for a computer system to work.

Computer Hardware

1.Hardware: Hardware is a physical component of a computer system which we can see and touch . The hardware components are mainly divided into 3 parts:

- I. Input Devices
- II. Output Devices
- III. CPU



Computer Block diagram

I. Input Devices:

- An input device is any hardware component used to enter data, programs, commands and user responses in to a computer.
- There are varieties of input units which are used by computers. Some of them are listed below. Keyboard, Mouse, Punched Cards, Light Pen, Joy Stick, Touch Screen, Microphone, MICR, Scanner, OCR, OMR, Smart Card Reader, Bar Code Reader, Biometric Sensors, Digital Camera, Web Camera.

Key Board:

- An important data entry device is the keyboard, which is a type writer like device.
- Internally a keyboard contains a matrix of switches, [one switch per key] and a key board controller.
- Through pressing and Releasing of Switches, key board controller generates a scan code. This scan code is sent to personal Computer [pc].
- The PC has another controller which converts the received scan code into a specific character.
- It includes functional keys, special keys[page up, page down, Home, Delete, Insert, End, Arrow keys] and Toggle keys [ctrl, shift, caps].

Mouse:

- The Mouse is a pointing device that fits under palm of a hand.

- It controls movement of pointer called “mouse pointer” on the screen.
- When a mouse moves on a flat surface, the cursor on the screen also moves in the direction of mouse’s movement.
- A mouse generally has two or three buttons and it may or may not have a wheel [scroller].

Punched Cards:

- These were oldest input devices.
- It consists of a card divided into 12 rows and 80 columns.
- The data was stored by making holes on the cards.

Light Pen:

- A light pen is also a pointing device.
- It consists of a photo cell mounted in a pen shaped tube.
- When the pen is brought in-front of a picture element on the screen, it senses light. The light coming from the screen causes the photo cell to respond by generating pulse.
- It is also used to draw images on the screen with the movement of the light pen the lines are drawn on the screen.

Touch Screen:

- A type of a display screen that has a touch sensitive transparent panel covering the screen known as “Touch Screen.”
- Instead of using pointing devices such as mouse or light pen, we can use the finger to point directly to the object on the screen.
- They are used to choose options which are displayed on the screen.

Graphic Tablet:

- A graphic tablet or digitizing tablet is an input device that enables us to enter drawings, and sketches into a computer.
- It consists of an electronic surface and a cursor or pen.
- When we draw on the electronic surface by using pen, the corresponding image is created on the screen.
- The drawings created in this manner are very accurate.

Joy Stick:

- It is a device that lets the user to move an object on the screen.
- Children can play with computers in a simplest way by using joystick. While playing certain games, the user needs to move certain objects quickly on the screen.
- It is generally used to control the velocity of screen cursor movement.

Micro Phone [MIC]:

- We can send sound input to computer through a special input device called Micro processor or MIC in short
- When we input the sound through MIC, the sound card translates the electric signals from microphone into digitized form that the computer can store and process.
- These are best used in multimedia presentation.

Magnetic Ink Character Reader [MICR]:

- MICR reads characters which are printed using a special Ink called “Magnetic Ink” that contains “Iron Oxide”.

- MICR's are mainly used in Banks. In a cheque the branch code, cheque numbers are printed in bottom using Magnetic Ink.
- The cheque can be read by using MICR. This method saves time and also ensures accuracy of data.

Scanner:

- A scanner is a device similar to a photo copiers {Xerox} that prints the given image in a paper. Whereas scanner creates an electronic form of a printed image.
- It digitizes the image i.e., it stores the image in terms of numbers. Image is scanned as a group of dots called "pixels". Each pixel is then stored as a number.
- The resolution of a scanner is a measure of number of pixels.

Optical Character Reader {OCR}:

- An OCR is used to Read Character of special type of font printed on paper. They can also identify hand written text also.
- OCR converts the image of text in to actual text which is editable in word processors.

Optical Mark Reader {OMR}:

- In this method special pre-printed forms are designed with boxes which can be marked with a dark pencil (or) ink.
- Such document form is read by a document reader called "Optical Mark Reader" which translates the darkened marks in to electric pulses which are transmitted to the computer.

Smart Card Reader {SMR}:

- The enhanced version of cards with magnetic strips is called the "Smart Card".
- The special reader machines that can read information on smart cards are called "Smart Card Readers."
- They have variety of applications including banking, security, medical records.

Bar Code Reader {BCR}:

- A Bar code is a pattern of printed bars on various types of products.
- A Bar code reader emits a beam of light, generally a laser beam, which reflects off the bar code image.
- Once the bar code is identified the bar pattern is converted in to numeric code that can be processed in any manner.

Bio-Metric Sensors:

- Bio-Metric sensors are input devices used for identifying a person's identity.
- Bio-Metrics is a technology that verifies a person's identity by measuring a unique character belongs to the individual.
- Bio-Metric sensors are used in finger print capturing, voice Recognition, dynamic signature verification etc.

Digital Camera: {DigiCam}

- A camera that stores images digitally rather than recording them on film is called digital camera.
- Once a picture has been taken, it can be downloaded in to a computer system.
- The main advantage is that making photos is both is expensive and fast because there is no film processing.

Web Camera: {webcam}

- A Web cam is essentially a camera that is connected to a computer either directly (or) wirelessly.
- A Web camera gathers a series of digital images that can be displayed on web browser.

II. Output Devices:

- The data should be retrieved to view it by user.
- To view data, the computer must be connected to an output device.
- The output device produces output in human readable form.
- The most output common devices are monitors, printers, plotters, speakers and projectors.

Monitors:

- Monitors (or) screen are the most common output form of a computer. It is also called VDU { Visual Display Unit }.
- It displays information in a similar way to that shown on television screen.
- The picture on, monitor is made of 1000 of tiny colored data's called "pixels".
- The 2 most common types of monitors are
 - Cathode Ray Tube Monitors [CRT]
 - Liquid Crystal Display Monitors [LCD]

CRT Monitors:

- The CRT works in the same way as television. It contains an electron gun at the back of glass tube.
- This fails electrons on phosphorous coat screen, when electron strikes the phosphorous screen. It glows to give the color.

LCD Monitors:

- This is smaller and lighter than the CRT which makes them for use with portable laptops and palmtops.
- It is also called TFT display [Thin Film Transistors].
- CRT monitors are big and require lot of power where as LCD's use less power and occupies less space.

Printers:

- It is a hard copy output device. They can produce text and images on paper. They can produce both color and black & white prints.
- It can be divided in to two categories.

Impact printers:

- In these printers, there is a mechanical contact between the print head and paper.
- They are having low operating speed and can print on continuous stationary.
- They are having low resolution i.e., print quality is poor. They are very slow and very noisy.
- The impact printers come in lot of varieties, important two types of printers are
 - Dot-Matrix printer
 - Drum printer

Dot-Matrix Printers:

- It is the most popular serial printer i.e., prints one character at a time.
- The head moves across the paper against an inked ribbon to form a pattern of dots on the paper.
- They are robust and can be used in harsh conditions such as in factories.

Drum Printers:

- It is another important impact printer that is generally used in various office applications.
- It has a roller drum on which alphabets are placed in a particular order.
- They are very slow and noisy and can't produce color prints.

Non-Impact Printers:

- In these printers there is no mechanical contact between the print head and paper.

- They are having high resolution and are having high operating speed than that of impact printers.
- The two types of Non-impact printers are
 - Inkjet Printers
 - Laser Printers

Inkjet Printers:

- It is a printer that fires extremely small droplets of ink on to paper to create impression of text (or) image.
- It directs a high velocity stream of ink towards the paper.
- They have good resolution {nearly 300 to 600 data per inch}
- They are cheaper, light weight and very quiet.
- They can print Black & White as well as color on the same page.

Laser Printers:

- The desired output image is written on a copier drum with the help of a light beam controlled by a computer.
- With these certain parts get electrically charged then this drum is exposed to laser beam.
- These laser exposed areas attract toner that forms the image.
- The laser printers are quiet and are capable of produce a high quality prints.
- The speed of laser printers can be up to 10-20 pages per minute {PPM}.
- Depending upon the printing capacity the printers are further classified in to 3 types.

Character Printer:

These printers print one character at a time and these are the slowest printers.

Line Printers:

These printers are capable of printing one line at a time.

Page Printers:

These printers are capable of printing one page at a time.

Plotters:

- Many applications require a graphical output apart from printed output. For example, pie-charts, bar-charts and graphs that is useful for representation of data.
- Plotters are output devices that produce good quality of drawings and graphs.
- They are expensive and slower than printers since they draw each line separately.
- There are two types of plotters namely drum plotters and flat-bed plotters.

Speakers:

- A speaker has become key sound output systems.
- A computer system must require a sound card.
- Speaker receive constantly change in electric current from sound cards.
- This current transferred to a magnet and there it converts to sound format.
- The speakers that are attached to a computer are similar to the one that are attached to the stereo.

III.CPU (Central Processing Unit): It is a Heart of a Computer. The part of the computer that executes the instructions (program) stored in memory. The CPU is made up of the Control Unit , Arithmetic/Logical Unit and Memory unit. CPU (Central Processing Unit) - controls the operation of the computer (i.e., the brains of the computer).

- Plastic, metal and mostly silicon.
- Composed of several million transistors.
- Enormously complicated wiring.

- Performs program control, arithmetic, and data movement.
- Locates and executes program instructions.



ALU+CU+Memory=CPU

Arithmetic/Logical Unit (ALU):

- Arithmetic/logic Unit - the component that performs arithmetic and logic operations (i.e., a calculator and decision maker).

Control Unit:

- The component of the CPU that controls the actions of the other components so that instructions are executed in the correct sequence.
- Control Unit - the component that coordinates and supervises all operations of the stored program

Memory Unit:

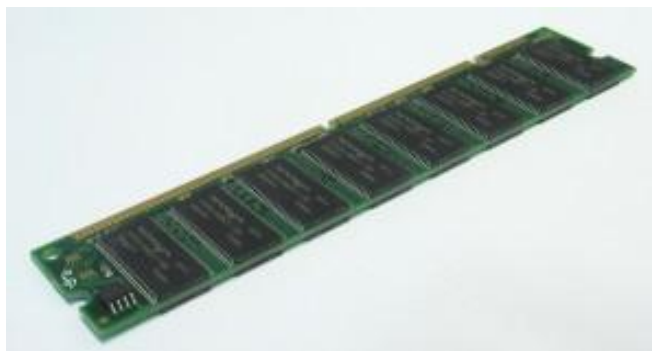
- Internal data storage in a computer (or) Memory means Stores information being processed by the CPU. It also saves the data for the later use.

The various storage devices of a computer system are divided into two categories.

Primary Storage:

- Stores and provides very fast.
- This memory is generally used to hold the program being currently executed in the computer, the data being received from the intermediate and final.
- The primary memory is temporary/volatile in nature.
- The data is lost, when is switched off.
- In order to store the data the data has to be to the secondary memory.
- The cost of the primary storage is more compared to the secondary storage.
- Therefore most computers have limited primary storage capacity.

RAM (Random Access Memory): Read-Write memory. RAM memory is volatile - its contents are lost when power is turned off.



ROM (Read Only Memory): It contains certain programs that must always be present. ROM is non-volatile - its contents are permanent when power is off.

Secondary Storage:

- Secondary storage is used like an archive/files/library/collection.
- It stores several programs, documents, data bases etc.
 - Hard disks, Magnetic Tape, Floppy Disk, Optical (Video or laser disk), CD(Compact Disk), Magnetic Disk, Rewritable Optical Disk , DVD, PENDRIVE, IPOD.

1. Software:

- It represents the set of programs that govern the operation of a computer system and make the hardware to function.
- Software can be classified broadly into two categories, mainly
 - System Software
 - Application Software

System Software:

- The Software that controls internal computer operations is called system software.
- Ex: Some of the internal computer operations are reading data from input devices, transmitting processed information to the output devices, converting data to computer understandable format.
- Examples of system softwares:
 - Operating System
 - Language Translator(Compiler, Assembler, Interpreter)

Operating System: (OS)

- An operating system is a program which acts as an interface between a user and hardware i.e. all computer resources.
- Operating system itself decides “how to do” “what to do” and “when to do” about a particular operation.
- The primary goal of an operating system is to make the computer system convenient to use and secondary goal is to use computer hardware in an efficient manner.
- An operating system is an important component of a computer system which controls all other components of the computer system.

Language Translator:

- As programmer prefer to write their programs in one of the High Level Languages {HLL} because it is much easier to understand the code easily.
- However, the computer doesn't understand any language other than “Binary Language.” It becomes necessary to process a HLL program me so as to make it understandable to the computer.
- The system program which performs the above job is called language processors.
- Assembler, Interpreter and Compilers are some of the language processors.

Assembler:

- The code written by the user in order to perform a task is called source code.
- The source code can't be understood by computer. Computer can understand only machine language code i.e., 0's & 1's. The code understood by the computer is called object code.
- The software which converts source code to object code is called “Translator”.
- In order to convert an Assembly level language code in to object code we use a translator called “Assembler”.
- **Assembler:** It is a Translator which converts Assembly level language code to Machine language code.

Compiler or Interpreter

- The code in a source file stored on the disk must be translated into machine language. This is the job of the translator.
- Compiler and interpreters are translators. Compiler compiles the entire program at once where as the interpreter interprets the program line by line.
- The translator reads the program and writes the resulting object module to a file that can then be combined with other precompiled units to form the final program.

Application Software:

- An Application Software is a set of programmers necessary to carry out operations for a specified application.

- These are the programmers written by programmers to enable computer perform specific task.
- Application software can be further sub-divided in to 3 categories.
 - Packages
 - Utilities
 - Business Software

Packages:

- Packages are general purpose application software. This software's has been written to do almost every task that is commonly used in various organizations. Some major packages are Word Processing System, Spread sheets, graphics, and multimedia.

Utilities:

- These are application programs that help the computer by performing housekeeping functions like backup, scanning virus, disk de-fragmentation, etc.

- These are helpful programs that ensure the smooth functioning of the computer.

Business Software:

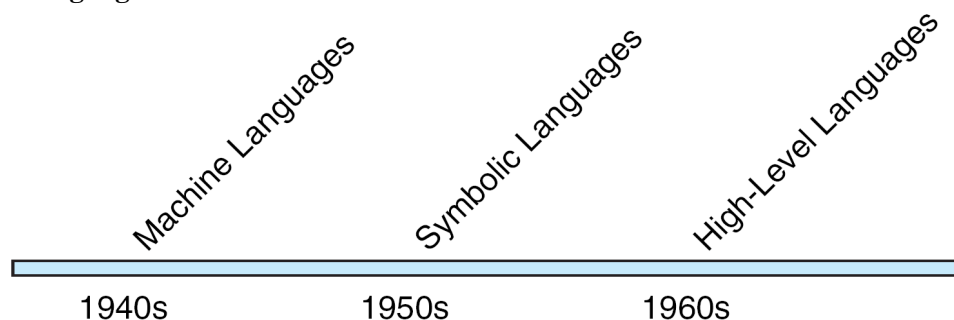
- This type of software is developed based on requirements of a particular business.
- Examples are Inventory Management System, Railway Reservation System, and Hospital Management System.

Computer languages:**Programming Languages**

Language is a tool, which is used for communication. Programming language is used to communicate a user with computer.

Programming languages are broadly divided into three types

- 1) Machine Language
- 2) Middle Level Language / Assembly Language
- 3) High Level Language



Machine Language:

the only language understood by a computer

```
0101 1000 0001 0000
0101 1011 0001 0000
0101 0000 0001 0000
```

Assembly/Symbolic language

```
MOVE  $$1,D1
MOVE  $$8,D0
ADD   D0,D1
```

High-level language

```
LUAS = PANJANG * LEBAR
IF LUAS > 50
    GOTO BESAR
ELSE
    GOTO KECIL
```

13

Machine Level Language:

- The computer internally understandable programming language is Machine Level Language.
- Machine Language is a collection of binary numbers.
- The binary system is based on two digits 1 & 0.
- Letters of the alphabet are also represented as numbers.
- Machine language binary number codes differ from CPU TO CPU.
- In the early days of computers like ENIAC, instructions written in 1's & 0's.

Disadvantages:

- There is no standard machine language.
- Very hard to program using machine language.
- One adv of machine language – its execution is very fast

Middle Level Language/ Assembly Language

- Assembly Language is a collection of symbols/abbreviations/mnemonic codes.
Ex: ADD, SUB, MUL, DIV, CMP etc.
- Codes are more easily memorized.
- Assembler is a systems program, which is used to translate Middle Language to Machine Language.

- Assembly Language is micro processor dependent language.

Advantages:

- It is more standardized.
- It is easy to correct errors.

Disadvantages:

- Its size is too long
- Its programs are still complex.
- It is machine dependent.

High Level Programming Languages

- It is simple English like language
- It is collection of alphabets, digits & special symbols.
- It is machine independent language.
- It is easy to learn & solve complex problems efficiently.
- Compiler is system software, which is used to translate High level language to machine level language.
- Interpreter is also system software, which is used to translate High level Language to machine language.

High Level Languages are broadly divided three types

Procedure Oriented Languages.

Object Oriented Languages.

Natural / fifth generation languages.

- The major difference between Compiler & interpreter are compiler translate whole program at a time where as interpreter translate line by line.
- Ex: C – compiler PASCAL - interpreter

ALGORITHM

1. Step by step Procedure to solve a Particular problem is called **ALGORITHM**
2. Algorithm can also be called as Pseudo - Code
3. Algorithm can be defined as a well-defined computational procedure that takes single value or a set of values as inputs and produces a single or multiple values as output.
4. Thus it is a sequence of steps which transforms an input into an output.
5. A single problem may have multiple algorithms. Also, an algorithm is always a language-free computational steps.
6. In order to compare algorithms, we must have some criteria to measure the efficiency of our algorithms. Suppose an algorithm has m input data. The time and space used by the algorithm are the two main measures for the efficiency of the algorithm.
7. The time is measured by counting the number of key operations— in sorting and searching algorithms,
8. The space is measured by counting the maximum of memory needed by the algorithm.

Algorithm is made up of the following basic logic structures.

- 1. Sequence** – The sequence logic is used for performing instructions one after another in sequence.
- 2. Selection** (If...then...else (or) if..then) – The selection logic is also called as the decision making logic, as it is used for making decisions and selecting proper path out of the two or more alternative paths in the program logic.
- 3. Iteration** (Do..while (or) Repeat.. Until) – The iteration logic is used to produce loops when one or more instructions may be executed several times depending on some condition.

Write a algorithm to find out number is odd or even?

```
step  1 : Start
      2 : Read a number
      3 : Compute  $rem \leftarrow \text{number} \bmod 2$ 
      4 : if  $rem \leftarrow 0$  then
            Display "number is even"
      else
            Display "number is odd"
      endif
      5 : Stop
```

Characteristics of an Algorithm:

Each and every Algorithm should satisfy the following Criteria:

1. Finiteness
2. Definiteness
3. Input
4. Output
5. Effectiveness

1) 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:- An algorithm should produce 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.

Advantages of Algorithm:







- Converting a Pseudo code to a programming language is much easier as compared to converting a flowchart or decision table.
- As compared to a flowchart, it is easier to modify the Pseudo code of program logic when program modifications are necessary.
- Writing of pseudo code involves much less time and effort than drawing an equivalent flowchart.



Limitations of Algorithm:

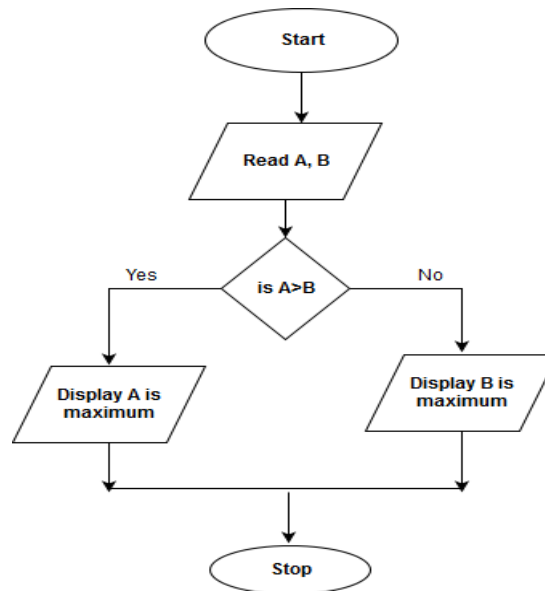
- A graphic representation of a program is not available in Algorithm.
- There are no standard rules to follow in pseudo code.

FLOWCHART

- A flowchart is the pictorial representation of an algorithm.
- The various steps in an algorithm are drawn in the form of prescribed symbols.
- The flow of the algorithm is maintained in a flowchart.
- The various symbols used in a flowchart are as below.

Symbol	Purpose	Description
	Flow line	Used to indicate the flow of logic by connecting symbols.
	Terminal(Stop/Start)	Used to represent start and end of flowchart.
	Input/Output	Used for input and output operation.
	Processing	Used for arithmetic operations and data-manipulations.
	Decision	Used to represent the operation in which there are two alternatives, true and false.
	On-page Connector	Used to join different flow line

Symbol	Purpose	Description
	Off-page Connector	Used to connect flowchart portion on different page.
	Predefined Process/Function	Used to represent a group of statements performing one processing task.

Example:**Advantages of flow charts:**

- It is machine-independent; hence we can use it for any computer system.
- The logic of the program is easily understood through it.
- Any logical errors can be easily checked and corrected through it.
- It is especially used when repeated calculations are involved.
- Program flowcharts serve as a good program documentation, which is needed for various purposes.
- The flowchart helps in debugging process.
- 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 flow charts:

- Flow charts are time consuming and difficult to draw with proper symbols and spacing,
- Especially for large complex programs.
- Owing to the symbol-string nature of flow charting, any changes or modifications in the
- Program logic will usually require a completely new flow chart.

Advantages of flow charts:

- It is machine-independent; hence we can use it for any computer system.
- The logic of the program is easily understood through it.
- Any logical errors can be easily checked and corrected through it.
- It is especially used when repeated calculations are involved.
- Program flowcharts serve as a good program documentation, which is needed for various purposes.

- The flowchart helps in debugging process.
- 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 flow charts:

- Flow charts are time consuming and difficult to draw with proper symbols and spacing,
- Especially for large complex programs.
- Owing to the symbol-string nature of flow charting, any changes or modifications in the
- Program logic will usually require a completely new flow chart.

SOFTWARE DEVELOPMENT METHOD

Introduction:-

A Disciplined approach is essential to create programs that are easy to read, understand and maintains.

- Programming is a problem-solving activity.
- Programmers use a methodology to solve the program and that methodology is referred to as software development method.

Steps in Software Development Life Cycle:-

The software development method includes the following steps.

- i) Problem specification
- ii) Analysis
- iii) Design
- iv) Implementation
- v) Testing
- vi) Maintenance

Problem specification:-

- It specifies the state of the problem clearly and without any ambiguity.
- This step helps us to gain a clear understanding of what is required for solving the existing problem and to find its solution.

Analysis:-

- This step involves identifying three important elements of a problem namely
 - Problem input i.e. the data that we have to work or process.
 - Problem output i.e. the result that we have to work or process.
 - Any additional requirements or constraints on the solution.
- At this stage we should also determine the required formal in which the results should be displayed.
- We should also develop a list of problem variables and their relationships.
- This process of extracting essential variables and their relationships is known as abstraction.

Design:-

- Designing makes to solve the problem by developing a list of steps called algorithm.
- Writing the algorithm is the most important part of problem solving.
- We generally use top-down design(also called divide and conquer) approach i.e. divide the problems into sub problems.
- Most computer algorithms contain three basic steps.
 - Get the data
 - Perform the computation
 - Display the results

- Once we know the algorithm we can perform algorithm refinement.
- Algorithm refinement is the development of a list of steps to solve a particular step in the original algorithm.
- Then we can represent algorithm in a pictorial representation by using flow chart.

Coding (or) implementation:-

- Implementing the algorithm involves writing a program in a particular language.
- We must convert each algorithm step into one or more statements in a programming language.

Testing:-

- Testing & verifying the program requires testing the completed program to verify that it works as desired.
- In testing we run the program several times using different sets of data to make sure that it works correctly for every situation.

Maintenance:-

Maintaining and updating the program involves modifying a program to keep it up to date as per company policy changes.

CASE STUDY:-**Analysis of Software Development Life Cycle:-**

Problem:- Converting miles to Kilometers

Step 1:- Problem Specification

In a particular surveying study job, some maps provide distance in kilometers, some in miles. Now the problem is to convert all the distances into kilometers i.e., we have to convert miles to kilometers.

Step 2:- Analysis

Data Requirement

- Problem Input :- miles (Distance in miles)
- Problem Output :- kms (distance in kilometers)
- Relevant Formula:- $1\text{km} = 1.609 * \text{miles}$

Step 3:- Design**Algorithm**

Step 1 :- start

Step 2:- input miles

Step 3:- convert miles to kilometers

Step 4:- display distance in kms

Step 5 :- stop

Algorithm Refinement:-

Step 1:- start

Step 2:- input miles

Step 3:- convert miles to kilometers

Step 3(a):- distance in kilometers is 1.609 times distance in miles.

Step 4:- display distance in kms

Step 5:- stop

Flow chart

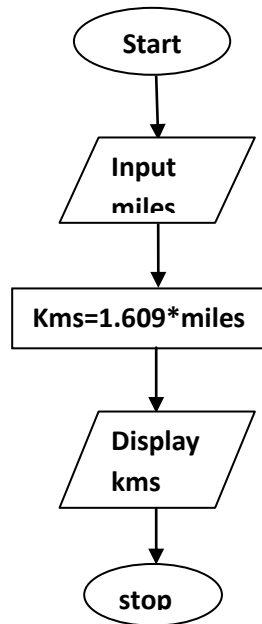


Fig: Flow chart for converting miles to kilometers

Step 4:- Implementation

Now we will implement this problem in a c language

```
// to convert miles to kilometers
#include<stdio.h>
#include<conio.h>
void main() {
    float miles,kms;
    clrscr();
    printf("Enter distance in kms");
    scanf("%f",&miles);
    kms=1.609*miles;
    printf("Equivalent distance in kilometers=%f",kms);
}
```

Step 5:- Testing

Test 1:-

Input :- Enter distance in kms=100.0

Output :- Equivalent distance in kilometers=160.9

Test 2:-

Input :- Enter distance in kms=23.2

Output :- Equivalent distance in kilometers=37.329

TYPES, OPERATORS AND EXPRESSIONS

Introduction to C programming:

History of C:

- C is a structured/procedure oriented, high-level and machine independent programming language.
- The root of all modern languages is ALGOL, introduced in the early 1960.
- In 1967, Martin Richards developed a language called BCPL.
- In 1970, ken thompson created a language using many features of BCPL and called it simply B.

- In 1972, Dennis Ritchie developed a language from the best features of ALGOL, BCPL & B and called it simply C.
- Standardized in 1989 by ANSI (American National Standards Institute) known as ANSI C
- International standard (ISO) in 1990 which was adopted by ANSI and is known as C89
- As part of the normal evolution process the standard was updated in 1995 (C95) and 1999 (C99)

Characteristics / Features of C

- C is General purpose, structured programming language and case sensitive programming language.
- C is highly portable i.e., it can be run in different OS.
- C is robust language, whose rich set of built in function and operations can be used to solve complex problems.
- C has ability to extend itself. We can continuously add our own functions to the existing system.
- C is well suited for writing both System S/W and Application S/W
- C program can be run on different OS with little or no modifications.
- C is a high level language i.e., it supports low level & middle level language features.
- C language allows reference to a memory location with the help of pointers, which holds address of memory location.
- C language allows dynamic memory allocation and manipulating data at bit level.
- C programs are fast & efficient and it has rich set of operators.
- C is used to develop System programs like OS, Compiler & Assembler etc.

Uses of C language:

The C language is used for developing system applications that forms a major portion of operating systems such as Windows, UNIX and Linux.

Ex:

- Database systems
- Graphics packages & Word processors
- Spreadsheets & Interpreters
- Operating system development
- Compilers and Assemblers

Structure of a C Program:

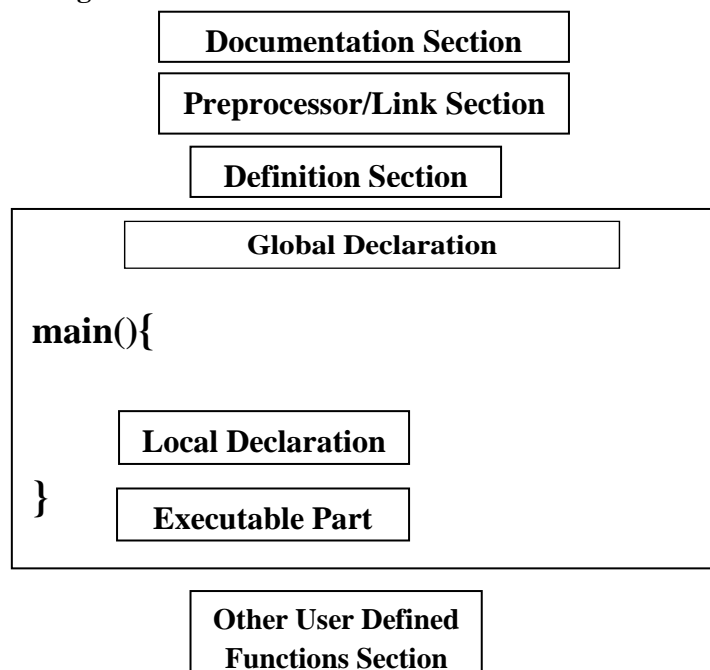


Fig: Structure of a C Program:

Documentation Section:

- It consists of set of comment lines used to specify the name of the program, the author etc.
- Comments are begin with /* and end with */, these are not executable, the compiler is ignored anything in between /* */

Ex: /* welcome to C world */ or// welcome to C world

Preprocessor Section:

- One major part of the C program is preprocessor.
- The preprocessor directives are commands that give instructions to C preprocessor.
- Whose job is modifying the text of the C program before it compiled?
- A preprocessor begin with #.
- Two most common preprocessor directives are #include & #define.
- Every C compiler contains collection of predefined function & symbols.
- Pre defined functions & symbols are organized in the form of header files whose name ends with .h.
Ex: stdio.h, conio.h etc.
- The #include directive causes the preprocessor to insert definitions from a standard header file.
Ex: #include<stdio.h>
- stdio.h having library functions like printf(), scanf() etc.

Definition Section:

- The definition section defines all symbolic constants.
Ex: #define PI 3.142.

Global Declaration:

- The variables that are used in more than one function throughout the program are called global variables and are declared out side of all the functions.

main() function:

- Every C program must have one main() function, which specify the starting of C program.
- Every C program execution begin from main() only.
- main() function is entry point of the program.
- C program allows any number other functions, which are called user defined functions.

Every C function contains two parts.

Declaration part: this part is used to declare all the variables that are used in the executable part of the program and called **local variables**.

Executable part: It contains at least one valid C statement.

- Every function execution begins with opening brace { and end with closing brace }.

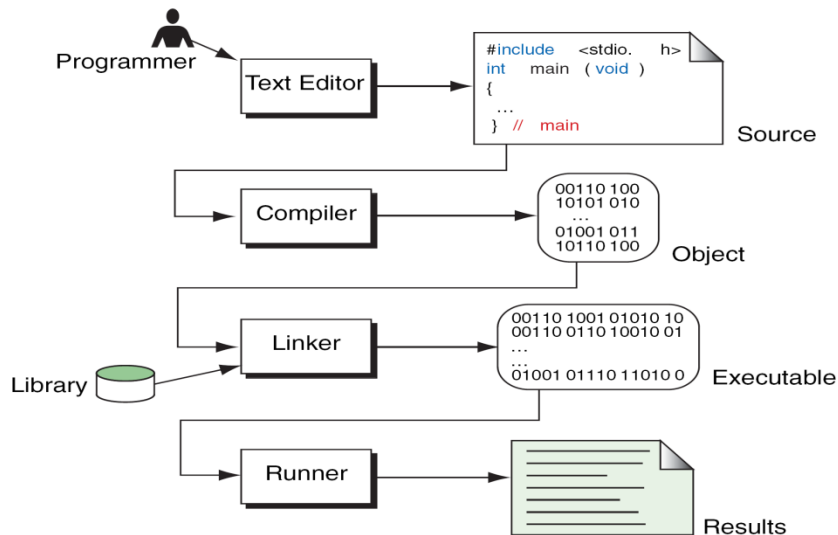
Note: The closing brace(}) of the main function indicate end of the program.

C program is a collection of functions.

Sample C program**sample.c**

```
#include<stdio.h>
void main()
{
    printf("Welcome to C");
}
```

Creating and Running Programs:



1)Creating the program:

- Creating the program means entering editing the program by using standard 'C ' editor and save the program with .c as an extension.
For Ex: sample.c.
- The popular C editors are turbo c, borland c, ANSI C etc.

2)Compiling the program:

- This is the process of converting the high level language to machine level language.
- The above process is performed, only when the program is syntactically & semantically correct .
- The mistakes in a program are called errors.

In C program errors are broadly divided into two types

1) Syntax errors 2) Logical errors

Syntax: It is a set of rules & regulations of programming language. (Or) grammar of the programming language.

Syntax Error:

- The grammatical errors in the program are called syntax errors.
- Syntax errors are easy to correct, because of C editor provide brief description about the error along with line number.
- During the compilation, C compiler scans the entire program to detect syntax errors.
- Once the program is successfully compiled, it will generate three different files.
- These are **.bak, .obj, .exe**.
- .bak are backup file, which are used for recover the source code.
- .obj are object file, it is collection of machine instructions that is output of compiler.

Linking the program with system library.

- The linker is a systems program, it combines user object file and library object files and also perform cross references.
- The linker output is .exe file and store on the disk.
- .exe is executable file: it is collection of machine instructions and are ready to execute under CPU.

Executing the program

- Loader is a systems program, it loads .exe file from disk to RAM and informs to the CPU beginning of the execution.
- Semantics: semantics are nothing but meaning of the identifier.
- C program execution always begin from main() function

Programming Rules

- All the statements in C program should be written in lower case letters. Upper case letters are only used for symbolic constants.
- The program statements can be write anywhere between two braces({, }) following the declaration part.
- The programmer can also write one or more statements in one line separating them with a semicolon.
- C is a free form language.
- C is a case sensitive language

C Tokens:

- C tokens are the basic buildings blocks in C language which are constructed together to write a C program.
- Smallest individual units in C program are known as C Tokens. C has 6 types of tokens

1. Keywords 2. Identifiers 3.Constants 4.Operators 5.Special Operators 6.Variables

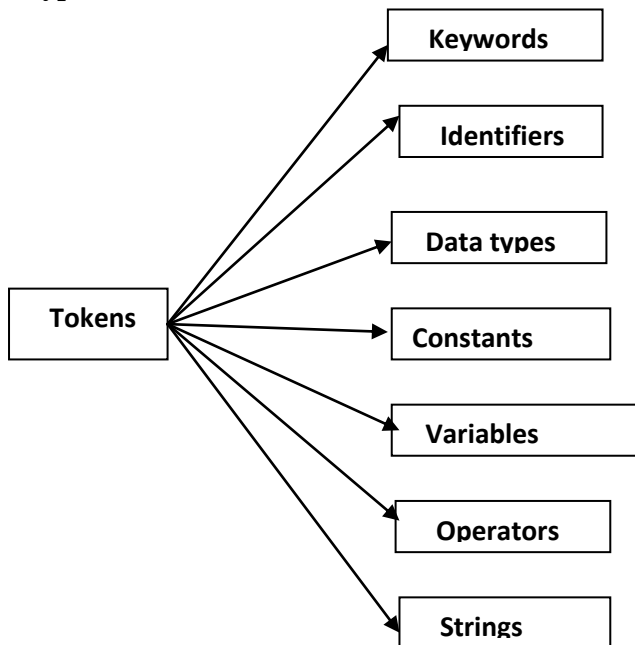
1. Data Types

Fig: Tokens in C

C tokens example program:

```
int main()
{
    int x, y, total;
    x = 10, y = 20;
    total = x + y;
    printf ("Total = %d \n", total);
}
```

where,

- main – identifier
- {, }, (,) – delimiter
- int – keyword
- x, y, total – identifier
- main, {, }, (,), int, x, y, total – tokens

1. Keywords:

- There are certain reserved words called token, that have standard & predefined meaning in C language.

- Whose meaning can't be changed?
- These are building blocks for C program statements.
- Each keyword is meant to perform a specific function in a C program.
- Since keywords are referred names for compiler, they can't be used as variable name.
- C having 32 keywords.
- All keywords must written in lower case.

For ex: int, for, while, if, else, struct, union, return etc.

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

2. Identifiers:

Identifiers are names given to various program elements such as variables, functions and arrays etc.

Rules for naming an identifier

1. Identifier consists of letters, digits & special symbol.
2. It allows one and only special character i.e under score (_).
3. Upper case are differ to lower case.
4. First character should be an alphabet or underscore.
5. Succeeding characters might be digits or letter.
6. Punctuation and special characters aren't allowed except underscore.
7. Identifiers should not be keywords.
8. An identifier can be any length, preferred size is 31 characters

3. Variable Names

- ✓ "A Variable is the name given to memory location to store data value in it." Unlike constant, the value of a variable can be changed.
- ✓ A variable name must be chosen in such a way that it improves the readability of a program.

Declaring & initializing C variable:

Syntax: <data type> <variable-name>;

Ex: - int a, b, c;

- In the above declaration a, b and c are integer variables which can store some integer data.

Ex:- float c, d, e;

- In the above declaration c, d and e are float variables which can store some real data

Initialization of a variable:

Static Initialization:

- int a,b,c; // a, b, c are declared
- a=10; b=20; c=30; // a,b,c are initialized with some values, this kind of initialization is called as static initialization.

Dynamic Initialization:

- a,b,c can also be initialized dynamically(at run time).
- scanf("%d%d%d",&a,&b,&c);

Rules for Constructing a Variable:

- A variable name is a combination of alphabets, digits and underscore.
- Other than the underscore no special characters is allowed.
- The first character in the variable name must be an alphabet.

- The variable name should not be of keyword.

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

1. Local variable
2. Global variable

1. Local Variable

- 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 variables are having scope within the main function only. These are not visible to test function.
- Likewise, a and b variables are having scope within the test function only. These are not visible to main function.

2. Global variable

- The scope of global variables will be throughout the program. These variables can be accessed from anywhere in the program.
- This variable is defined outside the main function. So that, this variable is visible to main function and all other sub functions.

1. Example program for local variable in C:

```
#include<stdio.h>
void test ();
int main()
{
    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' undeclared
error */
    printf("\nvalues : m = %d and n = %d", m, n);
    test();
}
void test()
{
    int a = 50, b = 80;
    // a, b are local variables of test function
    /*a and b variables are having scope within this test function
only. These are not visible to main function.*/
    /* If you try to access m and n in this function,you will get 'm'
undeclared and 'n' undeclared
error */
    printf("\nvalues : a = %d and b = %d", a, b);
}
```

Output:

values : m = 22 and n = 44
values : a = 50 and b = 80

2. Example program for global variable in C:

```
#include<stdio.h>
void test();
int m = 22, n = 44;
int a = 50, b = 80;
int main(){
    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);
}
```

Output:

```
All variables are accessed from main function
values : m = 22 : n = 44 : a = 50 : b = 80
All variables are accessed from test function
values : m = 22 : n = 44 : a = 50 : b = 80
```

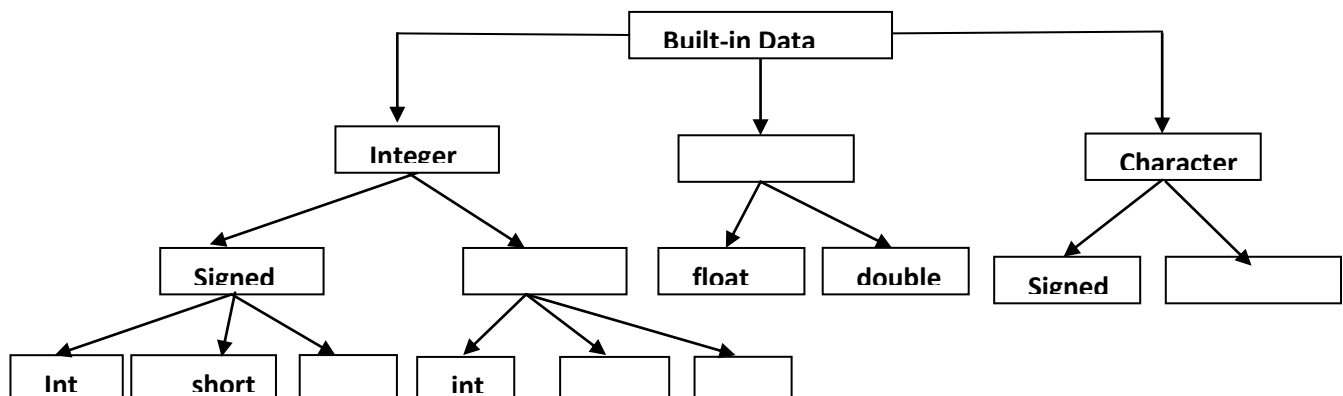
3.Data Types

- C data type refers to the kind of data or type of data involved during computation.
- Data types are used to define a variable before to use in a program, Each variable in C program associates with any one of the datatype.
- To decide the Size&type of variable.

C – data types:

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

Types	Data Types
Basic data types	int, char, float, double
Derived data type	pointer, array, structure, union
User defined data type	Typedef,enum
Void data type	void



S.No	C Data types	Format Specifier	Storage Size (bytes)	Range
1	Char/Signed char	%c	1	-128 to 127
2.	unsigned char	%c	1	0 to 255
3.	Int/signed int	%d	2	-32768 to +32767
4.	short int/signed short int	%d	2	-128 to +127
5.	long int/signed long int	%ld	4	-2147483648 to +2147483647
6.	unsigned int	%d(or)%u	2	0 to 65535
7.	unsigned short int	%d(or)%u	2	0 to 65535
8.	unsigned long int	%lu	4	0 to 4294967295
9.	Float	%f	4	3.4×10^{-38} to $3.4 \times 10^{+38}$
10.	Double	%lf	8	1.7×10^{-308} to $1.7 \times 10^{+308}$
11.	long double	%lf	10	3.4×10^{-4932} to $3.4 \times 10^{+4932}$

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

Note:

- ❖ We can't store decimal values using int data type.
- ❖ If we use int data type to store decimal values, decimal values will be truncated and we will get only whole number.
- ❖ In this case, float data type can be used to store decimal values in a variable.

Character data type:

- Character data type allows a variable to store only one character.
- Storage size of character data type is 1byte. 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.

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 bytes. 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.

Modifiers in C:

- The amount of memory space to be allocated for a variable is derived by modifiers.
- Modifiers are prefixed with basic data types to modify (either increase or decrease) the amount of storage space allocated to a variable.
- There are 4 modifiers available in C language. They are,
 1. short
 2. long
 3. signed
 4. unsigned

2. Derived data types in C:

- Array, pointer, structure and union are called derived data type in C language.
- Ex: `int a[10];`
`char name[20];`

3. User defined data types in C:

C supports two keywords to define user defined data types. They are

1. typedef 2. enum**1. typedef :**

- C supports a feature known as type definition that allows users to define an identifier that would represent an existing data type, the user defined data type identifier can later be used to declare variables.

Syntax: typedef type identifier;

- For Example, the declaration,

typedef int basic;

- makes the name **basic** a synonym of `int`. Now the type **basic** can be used in declarations, casts, etc, like,

basic num1, num2;

- Which will be treated by the C compiler as the declaration of `num1, num2` as `int` variables. “typedef” is more useful with structures and pointers.

2. enum:

- Enumerated datatype which can be used to declare variables that can have one of the values enclosed within the braces known as enumerated constants.
- Enumeration data type consists of named integer constants as a list.
- It starts with 0 (zero) by default and value is incremented by 1 for the sequential identifiers in the list.

Syntax: `enum identifier {value1, value2, ..., valuen};`

Ex: `enum day {Monday, Tuesday, ..., Sunday};`

example2:

```
enum month { Jan, Feb, Mar }; or
```

```
/* Jan, Feb and Mar variables will be assigned to 0, 1 and 2 respectively by default */
```

```
enum month { Jan = 1, Feb, Mar };
```

```
/* Feb and Mar variables will be assigned to 2 and 3 respectively by default */
```

```
enum month { Jan = 20, Feb, Mar };
```

```
/* Jan is assigned to 20. Feb and Mar variables will be assigned to 21 and 22 respectively by default */
```

Ex.: `enum color { RED, Green, BLUE }`

Enumerations provide a convenient way to associate constant values with names.

- The above enum functionality can also be implemented by “#define” preprocessor directive as given below. Above enum example is same as given below.

```
#define Jan 20;
#define Feb 21;
#define Mar 22;
```

C – enum example program:

```
#include <stdio.h>
int main()
{
enum MONTH { Jan = 0, Feb, Mar };
enum MONTH month = Mar;
if(month == 0)
printf("Value of Jan");
else if(month == 1)
printf("Month is Feb");
if(month == 2)
printf("Month is Mar");
}
```

Output:

Month is March

5.Void data type in C:

- Void is an empty data type that has no value.
- This can be used in functions and pointers.
- It is a special data type used for
 - To specify that a function doesn't returns any value.
 - To specify a function takes no arguments.
 - To create generic pointers.

Eg: 1. void print (void)
2. void *ptr

Type Conversion:

- Typecasting concept in C language is used to modify a variable from one data type to another data type.
- New data type should be mentioned before the variable name or value in brackets which to be typecast.
- In an expression that involves two different data types ,such as multiplying an integer and a floating point number to perform these evaluations ,one of the types must be converted.

Two types of conversions:

1.Implicit Type Conversion**2.Explicit Type Conversion****1Automatic (or Implicit) type conversion:**

- When the types of the two operands in a binary expression are different automatically converts one type to another .This is known as implicit type conversion .
- This type of conversion is useful and relied upon to perform integral promotions, integral conversions, floating point conversions, floating-integral conversions, arithmetic conversions, pointer conversions.

```
int a = 5.6;
float b = 7;
```

In the example above, in the first case an expression of type float is given and automatically interpreted as an integer. In the second case (more subtle), an integer is given and automatically interpreted as a float.

2. Explicit type conversion (Type Casting)

- Explicit type conversion uses the unary cast operator to convert data from one type to another. To cast data from one type to another, we specify the new type in parentheses before the value we want converted.
- Type casting is a way to convert a variable from one data type to another data type. For example, if you want to store a long value into a simple integer then you can type cast long to int. You can convert values from one type to another explicitly using the cast operator as follows:

(type_name) expression

Consider the following example where the cast operator causes the division of one integer variable by another to be performed as a floating-point operation:

```
#include <stdio.h>
main()
{
    int sum = 17, count = 5;
    double mean;
    mean = (double) sum / count;
    printf("Value of mean : %f\n", mean );
}
```

It produces the following result:

```
Value of mean : 3.400000
```

It should be noted here that the cast operator has precedence over division, so the value of sum is first converted to type double and finally it gets divided by count yielding a double value.

Type conversions can be implicit which is performed by the compiler automatically, or it can be specified explicitly through the use of the cast operator. It is considered good programming practice to use the cast operator whenever type conversions are necessary.

Integer Promotion

Integer promotion is the process by which values of integer type "smaller" than int or unsigned int are converted either to int or unsigned int. Consider an example of adding a character in an int:

```
#include <stdio.h>
main() {
    int i = 17;
    char c = 'c'; /* ascii value is 99 */
    int sum;
    sum = i + c;
    printf("Value of sum : %d\n", sum );
}
```

When the above code is compiled and executed, it produces the following result:

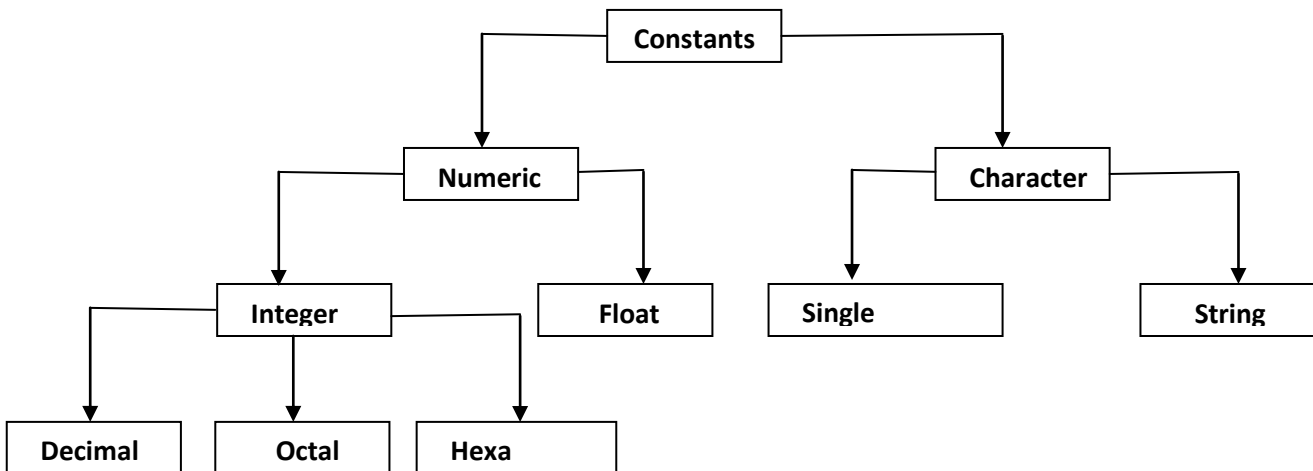
Value of sum : 116

Here, value of sum is coming as 116 because compiler is doing integer promotion and converting the value of 'c' to ascii before performing actual addition operation.

5.Constants:

Constants refer to fixed values that do not change during execution of program

Constants are broadly divided into following sub types



Integer Constants:

These refers to integers/whole numbers consisting of sequence of digits.

There are three types of integer constants

1. Decimal 2.Octal 3.Hexa Decimal

Rules for defining integer constants.

- 1.An integer constant must have at least one digit.
- 2.It should not contain any special symbols & white spaces.
- 3.It Should not contain any decimal point or exponent.
- 4.The integer can't exceed the range allowed the particular machine.

Decimal Constant

- Decimal integer constant range from 0 to 9 & preceded by optional sign.
- Decimal integer constant first digit must not be zero.

Valid invalid

+129 1,29

-95 1 54

Octal Constants:

- Octal integer constants are in the range from 0 to 7.
- Sign is optional.
- Every octal integer constant preceded with 0(zero).

Valid invalid

037 081

0 541

036 01,35

Hexadecimal constants

- Hexadecimal integer constants is a combination of digits from
- 0 to 9 & alphabets A to F represents from 10 to 15.

- Every hexadecimal number precede with 0X.

valid	invalid
0XA5	075
0XABC	0XAGE
0X9FA	0X7,AF

Real/Float Constants

- These constants refer to the numbers containing fractional parts.
- These are also known as floating-point constants.

Two ways of representing real constants

1. Decimal form 2. Exponential form.

In decimal form decimal & fractional part are separated by .(dot)

Valid	invalid
0.056	0.78.78
-6.453	-89.34 90

- Exponential form consists of two parts mantissa & exponent.
- Exponent & mantissa are separated by e or E.
- Exponent define to shift decimal point to the right if exponent is positive or to the left if exponent is negative.
- If decimal point is not included with in the number assumed to be positioned to the right of last digit.

Rules for constructing real/float constants

1. Mantissa & Exponent can be either positive or negative.
2. Special symbols are not allowed except .(dot)
3. Exponent must be an integer.
4. Exponent & Mantissa must have at least one digit each.

For Ex: 4 X 10⁴ can be represented as floating point constant as 40000. 4E4/4E+4 400E2

For Ex: 3.04 X 10⁻⁵ can be represented as floating point

constant as

3.04E-5 30.4 E -6
0.00304E-2

Invalid numbers

E+10 3.45e8.9

Character Constants

- These refers to single character enclosed with in single quote marks.
- A character constant is a single alphabet, a single digit or a single special symbol enclosed within single quotes.
- The maximum length of a character constant is 1 character.

For Ex: **valid** **invalid**

'A'	"A"
'2'	'abc'
'\$'	"\$"

String Constants

These refers to group of characters enclosed with in double quote.

For EX: **valid** **invalid**

"Hello"	'hello'
"A"	'A'
"2"	'2'

S.no	Constant type	data type	Example
------	---------------	-----------	---------

1	Integer constants	int unsigned int long int long long int	53, 762, -478 etc 5000u, 1000U etc 483,647 2,147,483,680
2	Real or Floating point constants	float double	10.456789 600.123456789
3	Octal constant	int	013 /* starts with 0 */
4	Hexadecimal constant	int	0x90 /* starts with 0x */
5	character constants	char	'A' , 'B' , 'C'
6	string constants	char	"ABCD" , "Hai"

Defining Constants

There are two simple ways in C to define constants –

- Using **#define** preprocessor.
- Using **const** keyword.

1. Example program using const keyword in C:

```
#include <stdio.h>
void main()
{
    const int 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);
}
```

Output:

```
value of height : 100
value of number : 3.140000
value of letter : A
value of letter_sequence : ABC
value of backslash_char : ?
```

2. Example program using #define preprocessor directive in C:

```
#include <stdio.h>
#define height 100
#define number 3.14
#define letter 'A'
#define letter_sequence "ABC"
#define backslash_char '\\?'
void main(){
    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);
}

```

Output:

```

value of height : 100
value of number : 3.140000
value of letter : A
value of letter_sequence : ABC
value of backslash_char : ?

```

Backslash Character Constants in C:

- There are some characters which have special meaning in C language.
- They should be preceded by backslash symbol to make use of special function of them.
- Given below is the list of special characters and their purpose.

Backslash character	Meaning
\b	Backspace
\f	Form feed
\n	New line
\r	Carriage return
\t	Horizontal tab
\"	Double quote
\'	Single quote
\\	Backslash
\v	Vertical tab
\a	Alert or bell
\?	Question mark
\N	Octal constant (N is an octal constant)
\XN	Hexadecimal constant (N – hex.dcm1 cnst)

7.Operators:

- An operator is a symbol that tells the computer to perform certain mathematical or logical manipulation on data stored in variables. The variables that are operated as operands.
- These C operators join individual constants and variables to form expressions.
- Operators, functions, constants and variables are combined together to form expressions.
- Consider the expression $A + B * 5$. where, +, * are operators, A, B are variables, 5 is constant and $A + B * 5$ is an expression.

Types of C operators:

C language offers many types of operators. They are,

1. Arithmetic operators
2. Increment and Decrement(unary) Operators
3. Assignment operators
4. Relational operators
5. Logical operators

6. Bit wise operators
7. Conditional operators (ternary operators)
8. Special Operators

1. Arithmetic operators:

The arithmetic operators that we come across in 'C' language are +, -, *, / and %. All of these operators are called 'binary' operators as they operate on two operands at a time. Each operand can be an *int* or *float* or *char*.

Arithmetic operators

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

Ex:

```
int x, y, z;
```

```
z=x+y;
```

```
z=x-y;
```

```
z=x*y;
```

```
z=x/y;
```

If both operands are integers, then the expression called an integer expression and the operation is called integer arithmetic. Integer arithmetic always yields an integer value.

Ex: for a=14, b=4,

a+b=18

a-b=10

a*b=56

a/b=3(decimal part truncated)

a % b=2(remainder of division)

During integer division, if the both the operands are of the same sign; the result is truncated towards to zero. If one of them is negative; the direction of truncation is implementation depended. Ex: 6/7=0 and -6/-7=0 but -6/7=0 or 1(machine dependent).

Similarly in modular division, the sign of the result is always the sign of the first operand (the dividend),

Ex: -14%3= -2;

-14%-3= -2;

14%-3= 2;

14%3=2;

If both operands are real, then the expression is called a real expression and the operation is called real arithmetic. A real operand may be either in decimal or exponential notation. Real arithmetic always yields a real value. The modulus (%) operator cannot be used for real operands.

If one of the operand is real and other is integer then the expression is called mixed-mode arithmetic expression. Here only the real operation is performed and the result is always in real form.

/ Write a c program that illustrates on arithmetic operators*/*

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

```
#include<conio.h>
```

```
void main( )
```

```
{
```



```

int a=10,b=5;
clrscr( );
printf("operation\t result\n");
printf("a+b=\t%d",a+b);
printf("a-b=\t%d",a-b);
printf("a*b=\t%d",a*b);
printf("a/b=\t%d",a/b);
printf("a%%b=\t%d",a+b);
getch( );
}

```

Output:

```

Operation  result
a+b=  15
a-b=   5
a*b= 150
a/b=   0

```

2.Increment and Decrement operators:

The increment (++) and Decrement (--) operators are add one and subtract one. These are unary operators since they operate on only one operand. The operand has to be a variable. The increment or decrement of the value either before or after the value of the variable is used. If the operator appears before the variable, it is called a prefix operator. If the operator appears after the variable, it is called a postfix operator.

Operator	Meaning
a ++	Post increment
++a	Pre increment
a --	Post decrement
--a	Pre decrement

a++ and ++a is the same when the statements are independent like

```

a=5;                a=5;
a++;                ++a;

```

In the both cases a value will be 6.

When the prefix ++ (or--) is used in an expression, the variable is incremented (or decremented) first and then the expression is evaluated using with the new value of the variable. Whereas the postfix ++ (or --) is used in an expression, the expression is evaluated first using with the original values of the variables and then the variable is incremented (or decremented) by one.

Consider the following:

```

a=5;                b=a++;

```

In this case the value of a would be 6 and b would be 5.If we write the above statement as

```

a=5;                b=++a;

```

In this case the value of a would be 6 and b would be 6.

/* program to demonstrate post increment and decrement*/

```

#include<stdio.h>
#include<conio.h>
void main( ) {
    int a, b, x, y;
    clrscr( );

```

```

a=4;b=5;x=10;y=20;
printf("value of a :%d\n",a);
printf("value of a++ :%d\n",a++);

printf("value of b :%d\n",b);
printf("value of b-- :%d\n",b--);
printf("value of b :%d\n",b);
printf("value of x*y++ :%d\n",x*y++);
getch( );
}

```

Output:

```

value of a :4
value of a++ :4
new value of a :5
value of b :5
value of b-- :5
value of b :4
value of x*y++ :200

```

/* program to demonstrate pre increment and decremenrt*/

```

#include<stdio.h>
#include<conio.h>
void main( )
{
    int a,b,x,y;
    clrscr( );
    a=4;b=5;x=10;y=20;
    printf("value of a :%d\n",a);
    printf("value of ++a :%d\n",++a);
    printf("new value of a :%d\n",a);
    printf("value of b :%d\n",b);
    printf("value of --b :%d\n",--b);
    printf("value of b :%d\n",b);
    printf("value of x*++y :%d\n",x*++y);
    getch( );
}

```

□ Output:

```

value of a :4
value of ++a :5
new value of a :5
value of b :5
value of --b :4
value of b :4
value of x*++y :210

```

2.Assignment Operators:

it is used to assign the result of an expression to a variable. Values can be assigned to variables using the assignment operator '=' as follows:

variable_name=constant;

Ex: balance=1278;

Yes='x';

C permits multiple assignments in one line.

Ex: balance=1278; Yes='x'; are valid statements.

An assignment statement implies that the value of the variable on the left of the 'equal sign' is set equal to the value of the quantity (or the expression) on the right.

The statement `year=year+1;` means that the 'new value' of year is equal to the 'old value' of year plus 1.

It is also possible to assign a value to a variable at the time the variable is declared. This takes the below form:

`int x=10;`

In addition C has a set of 'shorthand' assignment operators.

Syntax: V op= exp

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

The following are the shorthand assignment operators.

+= add assignment operator
 -= minus assignment operator
 *= multiply assignment operator
 /= divide assignment operator
 %= modulus assignment operator

Ex:

x+ = y is equivalent to x= x + y
 x- = y is equivalent to x= x - y
 x*=y is equivalent to x=x*y
 x/=y is equivalent to x=x/y
 x%=y is equivalent to x=x%y

4. Relational operators:

The relational and equality operators are used to test or compare values between two operands. The relational and equality operators produce an integer result to express the condition of the comparison. If the condition is false then the integer result is 0. If the condition is true then the result is non-zero.

The following table shows all the relational operators supported by C. Assume variable A holds 10 and variable B holds 20 then –

Operator	Description	Example
==	Checks if the values of two operands are equal or not. If yes, then the condition becomes true.	(A == B) is not true.
!=	Checks if the values of two operands are equal or not. If the values are not equal, then the condition becomes true.	(A != B) is true.
>	Checks if the value of left operand is greater than the	(A > B) is not true.

	value of right operand. If yes, then the condition becomes true.	
<	Checks if the value of left operand is less than the value of right operand. If yes, then the condition becomes true.	(A < B) is true.
>=	Checks if the value of left operand is greater than or equal to the value of right operand. If yes, then the condition becomes true.	(A >= B) is not true.
<=	Checks if the value of left operand is less than or equal to the value of right operand. If yes, then the condition becomes true.	(A <= B) is true.

Ex:

1) 10<20 result is TRUE (i.e.1)

2) 20<10 result is FLASE(i.e.0)

3) 10<=20 result is TRUE(i.e.1)

4) 20>=10 result is TRUE(i.e.1)

4) 10 == 10 result is TRUE(i.e.1)

5) 10!=10 result is FALSE(i.e.0)

*/*Program: Write a c program to use various relational operators and display their return values.*/*

```
#include<stdio.h>
#include<conio.h>
void main( )
{
clrscr( );
printf("\n condition: Return values\n");
```

*/*Program: Write a c program to find largest number among two numbers*/*

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

{
int a=10,b=20;
clrscr( );
if(a>b)
```

```
printf("\n 10!=10   :%5d",10!=10);
greater\n");
printf("\n 10 ==10   :%5d",10 ==10);
printf("\n 10 > =10   :%5d",10 > =10);
greater\n");
printf("\n 10 < =10   :%5d",10 < =10);
printf("\n 10!=9    :%5d",10!=9);
printf("\n 10>10    :%5d",10>10);
printf("\n 10<10    :%5d",10<10);
getch ( );
}
```

```
printf("a is
else
printf(" b is
getch( );
}
```

Output: b is greater

Output:

Condition : Return values

10!=10 : 0

10 ==10 : 1

10 > =10 : 1

```

10 < =10      :   1
10 !=9        :   1
10 > 10       :   0
10 < 10       :   0

```

5.Logical operators:

Logical operators are used to combine two or more relations. The logical operators are called Boolean operators. Because the tests between values are reduced to either true or false, with zero being false and one being true.

Operator	Meaning
&&	Logical AND
	Logical OR
!	Logical NOT

(NOTE: Expression is nothing but, it is a combination of operators and operands (i.e, constants, variables etc.) which will give you some value as its result)

The expressions can be connected like the following

(expression 1) &&|| (expression 2)

The below table demonstrates ‘&&’ and ‘||’ Operators:

Operands		Results	
Exp 1	Exp 2	Exp 1 && Exp 2	Exp 1 Exp 2
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	1

From the above table following rules can be followed for logical operators:

- The logical AND (&&) operator provides true result when both expressions are true otherwise 0.
- The logical OR (||) operator provides true result when one of the expression is true otherwise 0.
- The logical NOT (!) operator provides 0 if the condition is true otherwise 1

*/*program: Write a c program that illustrates the use of logical operators*/*

```

#include<stdio.h>
#include<conio.h>
void main( )
{
    clrscr( );
    Printf("\n condition: Return values\n");
    Printf("\n 5>3&&5<10   :%5d", 5>3&&5<10);
    Printf("\n 8>5 ||8<2    :%5d", 8>5 ||8<2);
    Printf("\n !(8==8)    :%5d", !(8==8));
    getch ( );
}

```

Output:

Condition: Return values

```
5>3&&5<10:      1
8>5 ||8<2       : 1
!(8==8)         : 0
```

*/*Program: Write a c program to find largest number among three numbers*/ #include<stdio.h>*

```
#include<conio.h>
void main() {
int a=10,b=20,c=30;
clrscr( );
if(a>b&&a>c)
printf("a is greater\n"); else if(b>c)
printf(" b is greater\n");
else
printf(" c is greater\n");
getch( );
}
```

Output:

c is greater

6.Bitwise Operators:

Bitwise operator works on bits and perform bit-by-bit operation. The truth tables for &, |, and ^ is as follows –

p	q	p & q	p q	p ^ q
0	0	0	0	0
0	1	0	1	1
1	1	1	1	0
1	0	0	1	1

Assume A = 60 and B = 13 in binary format, they will be as follows –

A = 0011 1100

B = 0000 1101

A&B = 0000 1100

A|B = 0011 1101

A^B = 0011 0001

~A = 1100 0011

The following table lists the bitwise operators supported by C. Assume variable 'A' holds 60 and variable 'B' holds 13, then –

Operator	Description	Example
&	Binary AND Operator copies a bit to the result if it exists in both operands.	$(A \& B) = 12$, i.e., 0000 1100
	Binary OR Operator copies a bit if it exists in either operand.	$(A B) = 61$, i.e., 0011 1101
^	Binary XOR Operator copies the bit if it is set in one operand but not both.	$(A \wedge B) = 49$, i.e., 0011 0001
~	Binary Ones Complement Operator is unary and has the effect of 'flipping' bits.	$(\sim A) = -61$, i.e., 1100 0011 in 2's complement form.
<<	Binary Left Shift Operator. The left operands value is moved left by the number of bits specified by the right operand.	$A \ll 2 = 240$ i.e., 1111 0000
>>	Binary Right Shift Operator. The left operands value is moved right by the number of bits specified by the right operand.	$A \gg 2 = 15$ i.e., 0000 1111

7. Conditional operator (or) ternary operator:

The conditional operator is a pair of operators (“?” and “:”) are sometimes called ternary operator since they take three operands, and it is condensed form of an *if-then-else* C statement.

The general form of Conditional Operator is:

$$exp\ 1? \ exp\ 2: \ exp\ 3;$$

The conditional operator works as follows: `expression1` is evaluated first. If it is non zero(true), then the expression 2 is evaluated. If expression 1 is false, expression 3 is evaluated. Note that only one of the expressions is evaluated.

Ex: y=(x>5? 3:4) is equivalent to

	if(x>5)
	then y=3;
	else y=4;

```

/* Program to demonstrate conditional operator */
#include<stdio.h>
#include<conio.h> void main( )
{
    clrscr( );
    3>2?printf("TRUE"):printf("FALSE");
    getch( );
}

```

Output:

TRUE

8. Special operators:

i) *Comma operator (,):* The comma (,) operator permits two different expressions to appear in situation where only one expression would ordinarily be used. The expressions are separated by comma operator.

Ex: $c = (a=10, b=20, a+b)$:

Here firstly value 10 is assigned to a followed by this 20 is assigned to b and then the result of a+b is assigned to c.

ii) *Sizeof operator*: The size of operator returns the number of bytes the operand occupies in memory. The operand may be a variable, a constant or a data type qualifier.

Ex: sizeof(int) is going to return 2

iii) *Address of operator (&)*: The address of operator (&) returns the address of the variable. The operand may be a variable, a constant.

Ex: m=&n;

Here address of n is assigned to m. This m is not a ordinary variable, it is a variable which holds the address of the other variable (i.e., pointer variable).

iv) *Value at address operator (*)*: The value at address operator (*) returns the value stored at a particular address. The 'value at address' operator is also called an 'indirection' operator.

Ex: x=*m;

The value at address of m is assigned to x. Here m is going to hold the address.

/ Program to demonstrate special operators*/*

```
#include<stdio.h>
#include<conio.h>
void main() {
int x=8,*m;
clrscr( );
m=&x;
printf("value of x :%d\n",x);
printf("value of x by using pointer :%d\n",*m); printf("address of x :
%u\n",&x);
printf("address of x using m : %u\n",m);
printf("address of m : %u\n",&m);
printf("sizeof(x) : %d\n",sizeof(x));
getch( );
}
```

Output:

```
value of x :8
value of x by using pointer :8 address of x : 2293572
address of x using m : 2293572 address of m : 2293568
sizeof(x) : 4
```

Precedence and Order of Evaluation

Operator precedence determines the grouping of terms in an expression and decides how an expression is evaluated. Certain operators have higher precedence than others; for example, the multiplication operator has a higher precedence than the addition operator.

For example, $x = 7 + 3 * 2$; here, x is assigned 13, not 20 because operator * has a higher precedence than +, so it first gets multiplied with $3*2$ and then adds into 7.

Here, operators with the highest precedence appear at the top of the table, those with the lowest appear at the bottom. Within an expression, higher precedence operators will be evaluated first.

Category	Operator	Associativity
Postfix	() [] -> . ++ --	Left to right
Unary	+ - ! ~ ++ -- (type)* & sizeof	Right to left
Multiplicative	* / %	Left to right
Additive	+ -	Left to right
Shift	<< >>	Left to right
Relational	< <= > >=	Left to right
Equality	== !=	Left to right
Bitwise AND	&	Left to right
Bitwise XOR	^	Left to right
Bitwise OR		Left to right
Logical AND	&&	Left to right
Logical OR		Left to right
Conditional	?:	Right to left
Assignment	= += -= *= /= %= >>= <<= &= ^= =	Right to left
Comma	,	Left to right

C Programming supports wide range of operators. While Solving the Expression we must follow some rules

While solving the expression [a + b *c], we should first perform Multiplication Operation and then Addition, similarly in order to solve such complicated expression you should have hands on Operator Precedence and Associativity of Operators.

Operator precedence & associativity table

Operator precedence & associativity are listed in the following table and this table is summarized in decreasing Order of priority i.e topmost operator has highest priority and bottommost operator has Lowest Priority.

Operator	Description	Associativity
() [] . -> ++ --	Parentheses (function call) (see Note 1) Brackets (array subscript) Member selection via object name Member selection via pointer Postfix increment/decrement (see Note 2)	left-to-right
++ -- + - ! ~ (type) * & sizeof	Prefix increment/decrement Unary plus/minus Logical negation/bitwise complement Cast (convert value to temporary value of type) Dereference Address (of operand) Determine size in bytes on this implementation	right-to-left
* / %	Multiplication/division/modulus	left-to-right
+ -	Addition/subtraction	left-to-right

<< >>	Bitwise shift left, Bitwise shift right	left-to-right
< <=	Relational less than/less than or equal to	left-to-right
> >=	Relational greater than/greater than or equal to	
== !=	Relational is equal to/is not equal to	left-to-right
&	Bitwise AND	left-to-right
^	Bitwise exclusive OR	left-to-right
	Bitwise inclusive OR	left-to-right
&&	Logical AND	left-to-right
	Logical OR	left-to-right
?:	Ternary conditional	right-to-left
=	Assignment	right-to-left
+= -=	Addition/subtraction assignment	
*= /=	Multiplication/division assignment	
%= &=	Modulus/bitwise AND assignment	
^= =	Bitwise exclusive/inclusive OR assignment	
<<= >>=	Bitwise shift left/right assignment	
,	Comma (separate expressions)	left-to-right

Summary of operator precedence

1. **Comma Operator** Has **Lowest Precedence**.
2. **Unary Operators** are Operators having **Highest Precedence**.
3. **sizeof** is Operator not Function.
4. Operators sharing Common Block in the Above Table have Equal Priority or Precedence .
5. While Solving Expression , Equal Priority Operators are handled on the basis of FIFO [First in First Out]
] i.e Operator Coming First is handled First.

Important definitions

Unary Operator	A unary operation is an operation with only one operand, i.e. an operation with a single input . A unary operator is one which has only one operand. e.g. post / pre increment operator
Binary Operator	A Binary operator is one which has two operand. e.g. plus , Minus .
Associativity	Their associativity indicates in what order operators of equal precedence in an expression are applied
Precedence	Priority Of Operator

Ex : operator precedence & associativity

We have listed out some of the examples based on operator precedence & associativity. Examples will give you clear picture how to use operator precedence & associativity table chart while solving the expression

Ex 1 : Simple use of precedence chart

```
#include<stdio.h>
int main() {
    int num1 = 10, num2 = 20;
    int result;
```

```
result = num1 * 2 + num2;
printf("\nResult is : %d", result);
return (0);
}
```

Consider the simple expression used in above program and refer operator precedence & associativity table chart

result = num1 * 2 + num2;

In this case multiplication operator will have higher priority than addition and assignment operator so multiplication will be evaluated firstly.

result = num1 * 2 + num2;

result = 10 * 2 + 20;

result = 20 + 20;

result = 40;

Ex 2 : Use of associativity

In above example none of the operator has equal priority. In the below example some operators are having same priority.

result = num1 * 2 + num2 * 2 ;

In the above expression we have overall 3 operators i.e. 2 multiplication, 1 addition and 1 assignment operator.

Now refer operator precedence & associativity table (block 3) which clearly tells that associativity is from left to right so we will give priority to multiplication operator on the left side

result = num1 * 2 + num2 * 2 ;

result = 10 * 2 + 20 * 2 ;

result = 20 + 20 * 2 ;

result = 20 + 40 ;

result = 60 ;

Type Conversions:

When variables and constants of different types are combined in an expression then they are converted to same data type. The process of converting one predefined type into another is called type conversion.

Type conversion in c can be classified into the following two types:

Implicit Type Conversion

When the type conversion is performed automatically by the compiler without programmers intervention, such type of conversion is known as **implicit type conversion** or **type promotion**.

The compiler converts all operands into the data type of the largest operand.

The sequence of rules that are applied while evaluating expressions are given below:

All short and char are automatically converted to int, then,

1. If either of the operand is of type long double, then others will be converted to long double and result will be long double.
2. Else, if either of the operand is double, then others are converted to double.
3. Else, if either of the operand is float, then others are converted to float.
4. Else, if either of the operand is unsigned long int, then others will be converted to unsigned long int.
5. Else, if one of the operand is long int, and the other is unsigned int, then
 1. if a long int can represent all values of an unsigned int, the unsigned int is converted to long int.
 2. otherwise, both operands are converted to unsigned long int.

6. Else, if either operand is long int then other will be converted to long int.
7. Else, if either operand is unsigned int then others will be converted to unsigned int.

It should be noted that the final result of expression is converted to type of variable on left side of assignment operator before assigning value to it.

Also, conversion of float to int causes truncation of fractional part, conversion of double to float causes rounding of digits and the conversion of long int to int causes dropping of excess higher order bits.

Explicit Type Conversion

The type conversion performed by the programmer by posing the data type of the expression of specific type is known as explicit type conversion.

The explicit type conversion is also known as **type casting**.

Type casting in c is done in the following form:

(data_type)expression;

where, *data_type* is any valid c data type, and *expression* may be constant, variable or expression.

Ex,

```
1x=(int) a+b*d;
```

The following rules have to be followed while converting the expression from one type to another to avoid the loss of information:

- All integer types to be converted to float.
- All float types to be converted to double.
- All character types to be converted to integer.

Expressions

- In programming, an expression is any legal combination of symbols that represents a value.
- C Programming Provides its own rules of Expression, whether it is legal expression or illegal expression. For example, in the C language $x+5$ is a legal expression.
- Every expression consists of at least one operand and can have one or more operators.
- Operands are values and Operators are symbols that represent particular actions.

Valid C Programming Expression:

C Programming code gets compiled firstly before execution. In the different phases of compiler, c programming expression is checked for its validity.

Expressions	Validity
$a + b$	Expression is valid since it contain + operator which is binary operator
$++ a + b$	Invalid Expression

Priority and Expression:

In order to solve any expression we should have knowledge of C Programming Operators and their priorities.

Types of Expression:

In Programming, different varieties of expressions are given to the compiler. Expressions can be classified on the basis of Position of Operators in an expression –

Type	Explanation	Example
Infix	Expression in which Operator is in between Operands	$a + b$
Prefix	Expression in which Operator is written before Operands	$+ a b$
Postfix	Expression in which Operator is written after Operands	$a b +$

These expressions are solved using the stack.

Examples of Expression:

Now we will be looking into some of the C Programming Expressions, Expression can be created by combining the operators and operands. Each of the expression results into the some resultant output value. Consider few expressions in below table Expression Examples, Explanation

$n1 + n2$, This is an expression which is going to add two numbers and we can assign the result of addition to another variable.

$x = y$,

This is an expression which assigns the value of right hand side operand to left side variable

$v = u + a * t$,

We are multiplying two numbers and result is added to 'u' and total result is assigned to v

$x \leq y$,

This expression will return Boolean value because comparison operator will give us output either true or false

$++j$, This is expression having pre increment operator\, it is used to increment the value of j before using it in expression [/table]

Control Statements:

- Control statements are building blocks of C Programming.
- Control statements determine the flow of execution.
- Control statements are used to create special program features, such as logical tests, loops and branches.

These are 3 types of control statements.

1.Sequential Statements(To alter the flow of a program).

2.Conditional Control Statements(Test the logical conditions).

3.Loop Control Statements(control the flow of execution as per the selection).

4.Jump/Unconditional Statements.

1.Sequential Statements:

- The Sequential statements are executed one after another from top to bottom.
- In this section every statement is executed exactly once.

Ex: convert Celsius to Fahrenheit.

$F=(1.8c+32)$.

2.Conditional Control/Selection/Decision statements:

- The execution of a statement is depends on result of the condition/expression.
- The selection statements create multiple paths in program.
- There are 4 different selection statements.

1.if

2.if –else

3.Nested if

4.Switch-case

Decision control statements	Syntax	Description
If	if (condition) { Statements; }	In these type of statements, if condition is true, then respective block of code is executed.
if...else	if (condition) { Statement1; Statement2; } else { Statement3; Statement4; }	In these type of statements, group of statements are executed when condition is true. If condition is false, then else part statements are executed.
nested if	if (condition1){ Statement1; } else_if (condition2) { Statement2; } else Statement 3;	If condition 1 is false, then condition 2 is checked and statements are executed if it is true. If condition 2 also gets failure, then else part is executed.

1.if:

- The if Statement is a decision making statement.
- if is a keyword
- If the test condition is true then the statement block after if is executed otherwise it is not executed
- If the condition is false, then the if block is skipped and execution continues with the rest of the program.

Syntax:

```
if(condition) //no semicolon
{
    st1;
}
```

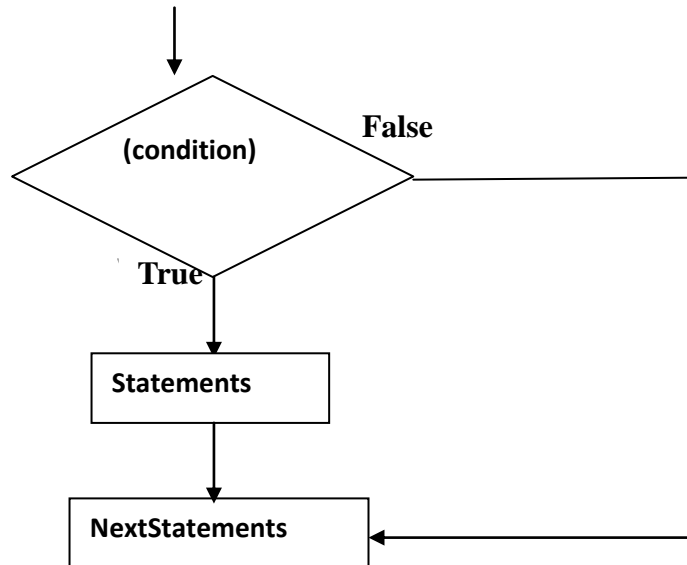
if block

```

    st2;
    ----- stn;
}
next statement;

```

Flow Chart:



Ex:

```

main()
{
    int a=10,b=20;
    if(a>b)
        printf("a>b");
    if(a<b)
        printf("b>a");
}

```

2)if-else Statement:

- The if/else statement is an extension of the if statement.
- If –else are keywords.
- If the condition in the if statement fails, the statements in the else block are executed.

Syntax:

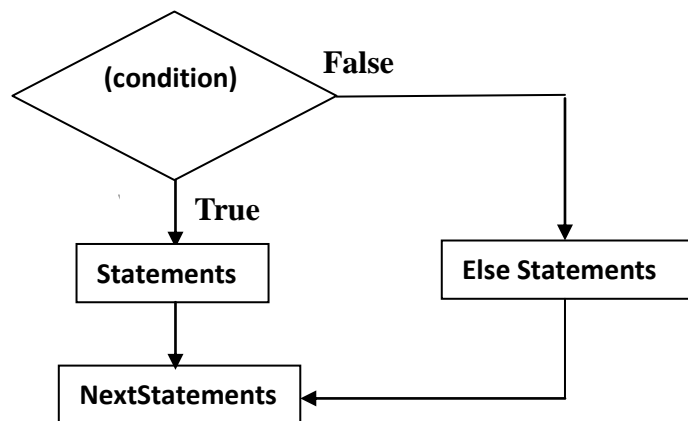
```

if(<conditional expression>)
{
    if block
    St1;
    ----
    ----
    Stn;
}
else
{
    else block
    St1;
    ----
    ----
    Stn;
}
Next statement;

```

if/else Flow Chart:





//C - program for concept of IF-ELSE Statement

```

#include<stdio.h>
#include<conio.h>
void main()
{
    int a = 10, b = 20;
    clrscr();
    if(a > b)
        printf("a > b");
    else
        printf("b > a");
    getch();
}

```

Nested if:

To define if-else or if in another if-else statement is called nested if or nesting.

(or)

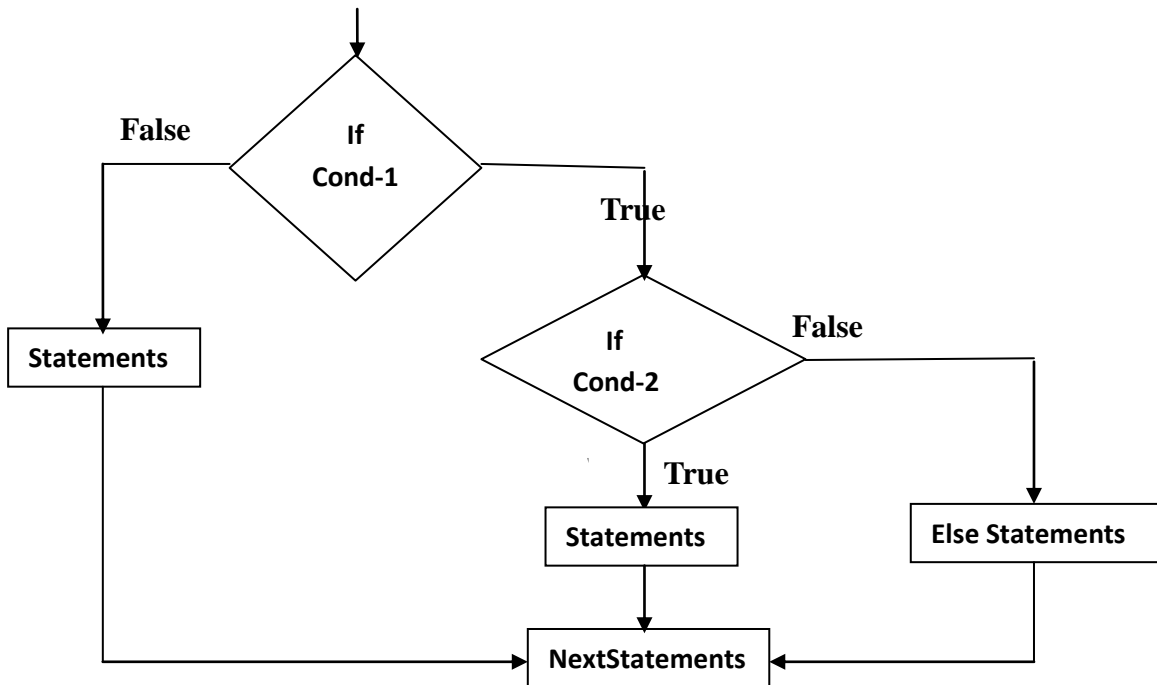
If more than one if else statement

Syntax:

```

if(<conditional expression>)
{
    st;
}
else
{
    if(expression2)
    {
        st1;
    }
    else
    {
        st;
    }
}
}
}
next statement;

```


**Ex:****main()**

```

{
int a=20,b=10,c=198;
if((a>b)&&(a>c))
{
printf("a is big");
}
else
{
if(b>c)
{
printf("b is big");
}
else
{
printf("c is big");
}
}
}

```

Ex2:

```

#include <stdio.h>
int main()
{
int m=40,n=20;
if (m>n) {
printf("m is greater than n");
}
else if(m<n) {
printf("m is less than n");
}
}

```

```
}  
else {  
    printf("m is equal to n");  
}  
}
```

Output:

m is greater than n

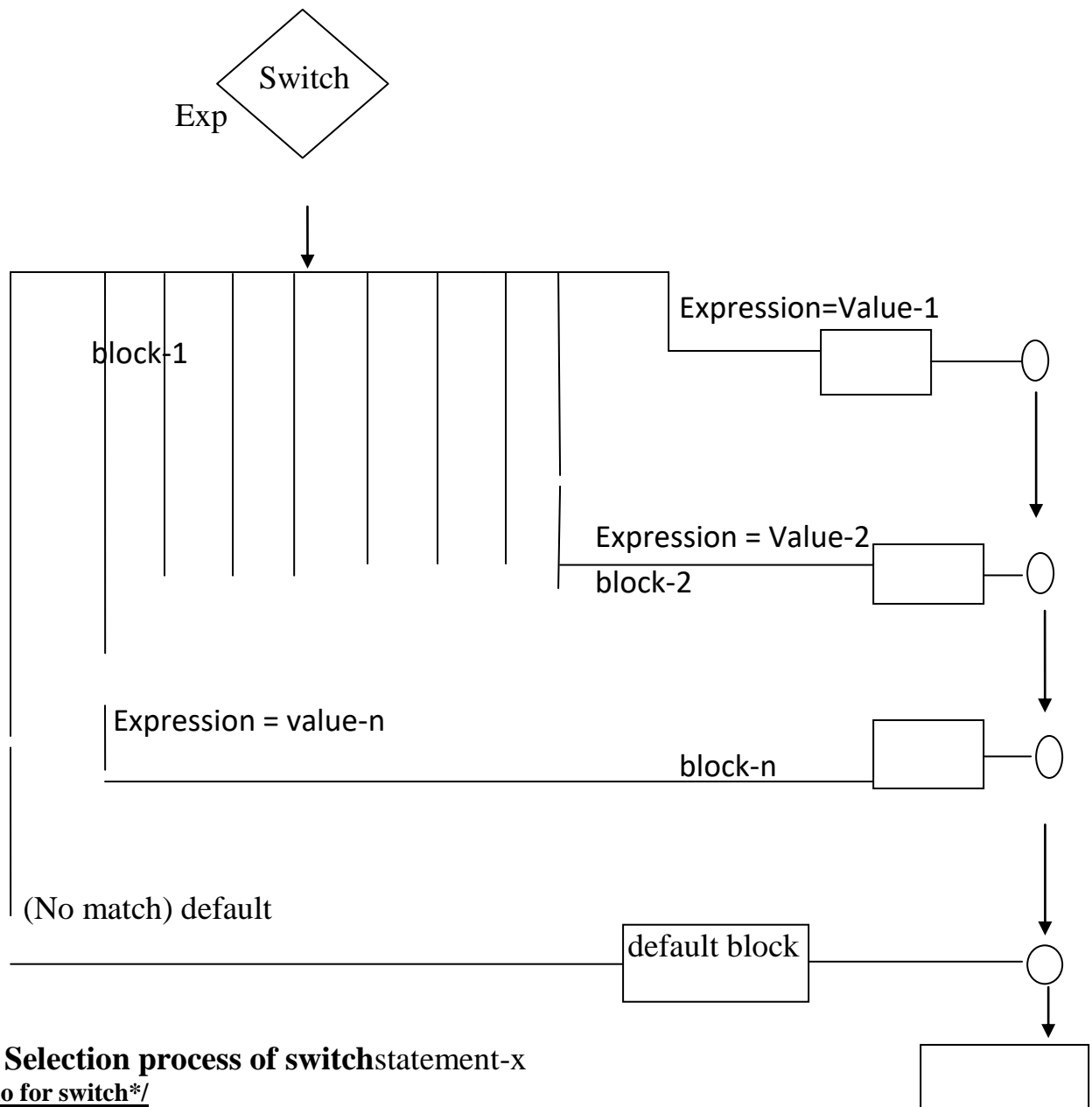
Switch Case Statement :

- The switch case statement also called a case statement is a multi-way branch with several choices.
- It is a keyword.
- Switch case statements are used to execute only specific case statements based on the switch expression.
- A switch is easier to implement than a series of if/else statements.
- Each label must equate to an integer constant and each must be unique.
- When the switch statement executes, it compares the value of the controlling expression to the values of each case label.

Syntax

```
switch (< expression>)  
{  
    case label 1: <statement 1>;  
    case label 2: <statement 2>;  
    ....  
    case label n: <statement n>;  
    default: <statement>;  
}
```

- Each case block and default block are terminated with keyword break is optional.
- Absence of break all the statements matched case are executed.
- break statement when used in a switch takes the control outside of the switch.
- In case : must
- No 2 case constants are identical.

Flowchart:**Selection process of switch statement-x**

/* Demo for switch*/

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

```
void main()
```

```
{
    int day; clrscr();
    printf("Enter Day:");
    scanf("%d", &day);
    switch(day)
    {
        case 1: printf("Monday");   break;
        case 2: printf("Tuesday");  break;
        case 3: printf("Wednesday"); break;
        case 4: printf("Thursday");  break;
        case 5: printf("Friday");    break;
        case 6: printf("Saturday");  break;
```

```

        case 7: printf("Sunday");    break;
        default: printf("Enter correct Day"); break;
    }
}

```

/* Demo for arithmetic operations using switch*/

```

#include<stdio.h>
#include<conio.h>
void main()
{
    int a,b; char ch; clrscr();
    printf("Enter a,b values:");
    scanf("%d%d",&a,&b);
    printf("Enter u r choice:"); getchar();
    scanf("%c",&ch);
    switch(ch)
    {
        case '+': printf("Addition=%d",a+b);
                  break;
        case '-': printf("Subtraction=%d",a-b);
                  break;
        case '*': printf("Multiplication=%d",a*b);
                  break;
        case '/': printf("Division=%d",a/b);
                  break;
        case '%': printf("Modulodiviion=%d",a%b);
                  break;
        default: printf("Enter correct operator");
                  break;
    }
}

```

Loop Control Statements:

Loop: Loop is a process of executing a set of statements one or more times.

The following are the loop statements available in 'C'.

- 1) While
- 2) Do –while
- 3) For loop

S.no	Loop Name	Syntax	Description
1	For	for (exp1; exp2; expr3) { statements; }	Where, exp1 – variable initialization (Example: i=0, j=2, k=3) exp2 – condition checking (Example: i>5, j<3, k=3) exp3 – increment/decrement (Example: ++i, j–, ++k)
2	While	while (condition) { statements; }	where, condition might be a>5, i<10
3	do while	do { statements; } while (condition);	where, condition might be a>5, i<10

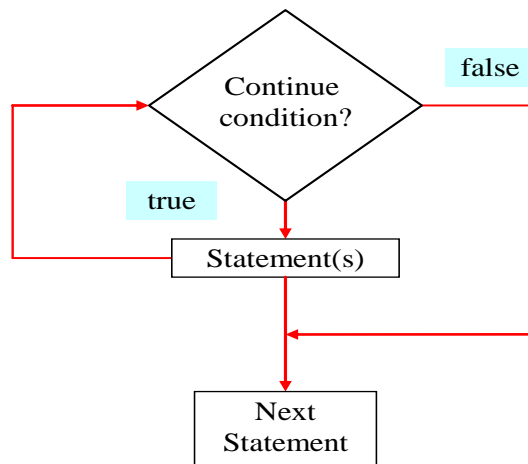
While Statement :

- while is a looping construct control statement that executes a block of code while a condition is true.
- The loop will never be executed if the testing expression evaluates to false.
- The loop condition must be a boolean expression.

- A while statement (pre test) checks at the beginning of a loop to see whether the next loop iteration should occur.
- It is a keyword, While is a entry control loop statement.

Syntax:

```
while (<loop condition>)
{
    <statements>;
}
```

**Example program (while loop) in C:**

In while loop control statement, loop is executed until condition becomes false.

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

```
int main()
```

```
{
```

```
    int i=3;
```

```
    while(i<10)
```

```
    {
```

```
        printf("%d\n",i);
```

```
        i++;
```

```
    }
```

```
}Output:
```

3	4	5	6	7	8	9
---	---	---	---	---	---	---

/* Reversing Number using while */

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

```
#include<conio.h>
```

```
void main()
```

```
{
```

```
    int n,rev=0,r;
```

```
    clrscr();
```

```
    printf("Enter n:");
```

```
    scanf("%d",&n);
```

```
    printf("Reverse Number=");
```

```
    while(n>0)
```

```
    {    r=n%10;
```

```
        rev=r;
```

```
        n=n/10;
```

```
        printf("%d",rev);
```

```
    } getch();
```

```
}
```

/* Factorial using while */

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

// Checking palindrome or not

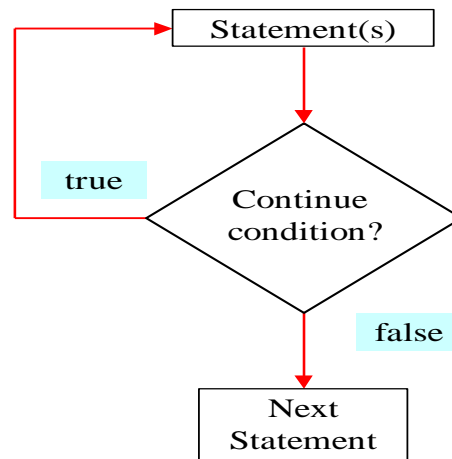
```
#include<stdio.h>
#include<conio.h>
void main()
{
    int n,r,rev=0,term;
    clrscr();
    printf("Enter n:");
    scanf("%d",&n);
    term=n;
    while(n>0)
    {
        r=n%10;
        rev=rev*10+r;
        n=n/10;
    }
    if(term==rev)
        printf("Palindrome");
    else
        printf("Not Palindrome");
    getch();
}
```

Do-while Loop:

- The do-while loop is similar to the while loop, except that the test is performed at the end of the loop instead of at the beginning.
- A do-while loop begins with the keyword do, followed by the statements that make up the body of the loop. When the loop condition becomes false, the loop is terminated and execution continues with the statement immediately following the loop.
- do-while is a Exit control loop statement.
- do is a keyword.

Syntax

```
do
{
    <loop body>
}while (<loop condition>);
```

Do-while Loop Flow Chart**Example program (do while loop) in C:**

In do..while loop control statement, while loop is executed irrespective of the condition for first time. Then 2nd time onwards, loop is executed until condition becomes false.

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

```
int main()
{
    int i=1;
    do
    {
        printf("Value of i is %d\n",i);
        i++;
    }while(i<=4 && i>=2);
}
```

Output:

```
Value of i is 1
Value of i is 2
Value of i is 3
Value of i is 4
```

Difference between while & do while loops in C:

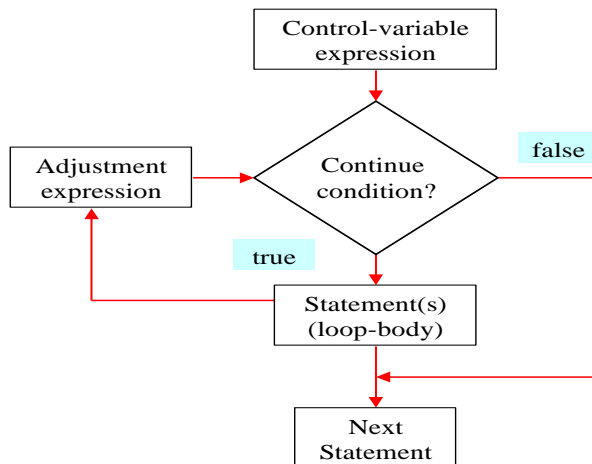
S.no	while	do while
1	Loop is executed only when condition is true.	Loop is executed for first time irrespective of the condition. After executing while loop for first time, then condition is checked.

For Loop:

- The for loop is a looping construct which can execute a set of instructions a specified number of times.
- It is a counter controlled loop.

Syntax

```
for (<initialization>; <loop condition>; <increment expression>)
{
    <loop body>
}
```

for Loop Flow Chart:**//Random Number Generations program**

```
#include<stdio.h>
#include<math.h>
void main()
{
    int i,n;
    clrscr();
    printf("Random Number Generations=");
    for(i=1;i<=10;i++)
    {
        printf("\t%4d",rand());
    }
    getch();
}
```

4.Jump Statements/Unconditional Statements:

The jump statements are used to transfer control from one place to another place randomly. There are 4 types.

- 1.break
- 2.continue
- 3.goto
- 4.return

1.Break Statement: The break statement transfers control out of the nearest enclosing loop (for, while, do or switch statement).

- Break statement is used to terminate the while loops, switch case loops and for loops from the subsequent execution.
- When **break** is encountered inside any loop, control automatically passes to the first statement after the loop. A **break** is usually associated with an **if**.
- It is a keyword.

- The label name is optional, and is usually only used when you wish to terminate the outermost loop in a series of nested loops.

Syntax:

break; // the unlabeled form

break <label>; // the labeled form

Example program for break statement in C:

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

    for(i=0;i<10;i++)
    {
        if(i==5)
        {
            printf("\nComing out of for loop when i = 5");
            break;
        }
        printf("%d ",i);
    }
}
```

Output:

```
0 1 2 3 4
Coming out of for loop when i = 5
```

/* Demo for break statement*/

```
#include <stdio.h>
main()
{
    int grade;
    printf ("Enter Input grade :");
    scanf("%d", &grade);
    switch (grade)
    {
        case 1:printf("Fall (F)\n");break;
        case 2:printf("Bad (D)\n");break;
        case 3:printf("Good (C)\n");break;
        case 4:printf("Very Good (B)\n");break;
        case 5:printf("Excellent (A)\n");break;
        default: printf("You have inputted false grade\n");
                break;
    }
}
```

Ex2:

```
main( )
{
    int i = 1 , j = 1 ;
    while ( i++ <= 100 )
    {
        while ( j++ <= 200 )
        {
            if ( j == 150 )
```

```
break ;
else
printf ( "%d %d\n", i, j ) ;
}
}
}
```

In this program when j equals 150, break takes the control outside the inner while only, since it is placed inside the inner while.

2.Continue Statement:

- Continue statement is used to continue the next iteration of for loop, while loop and do-while loops. So, the remaining statements are skipped within the loop for that particular iteration.
- It is a keyword.
- You use a continue statement when you do not want to execute the remaining statements in the loop, but you do not want to exit the loop itself.

Syntax:

continue; //the unlabeled form

continue <label>; // the labeled form

Example program for continue statement in C:

```
#include <stdio.h>
int main()
{
    int i;
    for(i=0;i<10;i++)
    {
        if(i==5 || i==6)
        {
            printf("\nSkipping %d from display using " \
                "continue statement \n",i);
            continue;
        }
        printf("%d ",i);
    }
}
```

Output:

```
0 1 2 3 4
Skipping 5 from display using continue statement
Skipping 6 from display using continue statement
7 8 9
```

```
/*c program for continue concept*/
#include<stdio.h>
void main()
{
    int i;
    clrscr();
    printf("Odd Numbers =");
    for (i = 1; i <= 15; ++i)
    {
        if (i % 2 == 0)
            continue;
        printf("\t%d",i);
    }
    getch();
}
```

}

When **continue** is encountered inside any loop, control automatically passes to the beginning of the loop. A **continue** is usually associated with an **if**. As an example, let's consider the following program.

```
main( )
{
int i, j ;
for ( i = 1 ; i <= 2 ; i++ )
{
for ( j = 1 ; j <= 2 ; j++ )
{
if ( i == j )
continue ;
printf ( "\n%d %d\n", i, j ) ;
}
}
}
```

The output of the above program would be...

```
1 2
2 1
```

Ex3:

```
#include <stdio.h>
#include <conio.h>
main() {
int n;
do {
printf ( " \nEnter the number :");
scanf("%d", &n );
if (n < 0) {
break;
}
if (n >10) {
printf("Skip the value\n");
continue;
}
printf ("The number is: %d",n);
} while (n!= 0);
}
```

3.goto statement:

- goto statements is used to transfer the normal flow of a program to the specified label in the program.
- **goto** statement transfers control to a label.
- goto is a keyword.
- The given label must reside in the same function and can appear before only one statement in the same function.
- goto is a unconditional branching statement used to transfer control of the program from one statement to another.

Syntax: goto name1;

```
.....
.....
```

```

    name1:
    statement;

```

Example program for goto statement in C:

```

#include <stdio.h>
int main()
{
    int i;
    for(i=0;i<10;i++)
    {
        if(i==5)
        {
            printf("\nWe are using goto statement when i = 5");
            goto HAI;
        }
        printf("%d ",i);
    }
    HAI : printf("\nNow, we are inside label name \"hai\" \n");
}

```

Output:

```

0 1 2 3 4
We are using goto statement when i = 5
Now, we are inside label name "hai"

```

```

}
Ex2:
#include <stdio.h>
int main(){
    int n = 0;
    loop:
    printf ("\n%d",n);
    n++;
    if (n<10){
        goto loop ;
    }
    return 0;
}

```

4.return statement:

- The return statement exits from the current method, and control flow returns to where the method was invoked.
- The return statement has two forms: one that returns a value, and one that doesn't.
- To return a value, simply put the value after the return keyword.
- The data type of the returned value must match the type of the method's declared return value.
- It is a keyword.
- A return statement terminates a function. All functions including main must have a return statement. Where there is no return statement at the end of the function, the system inserts one with a void return value.
- The return statement can return a value to the calling function. In case of main, it returns a value to the operating system rather than to another function. A return value of zero tells the operating system that the program executed successfully.

Syntax:

return expression; //return statement

/*Maximum of two numbers*/

#include <stdio.h>

int Max(int a, int b);

void main()

**{ clrscr();
 printf("%d", Max(5,10));
 getch();**

}

int Max(int a, int b)

**{ if (a > b)
 return a;
 else
 return b;
}**