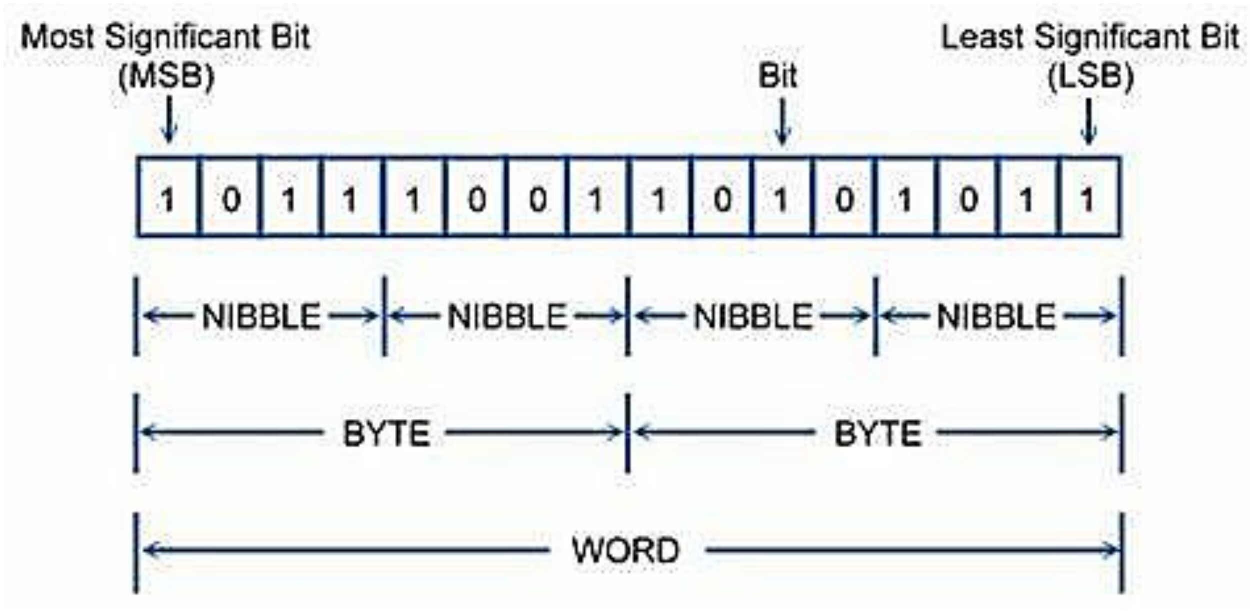
# *Unit-01*
# *Overview of C*

## BITS, BYTES and WORDS



### Bit

- Bit = From shortening of the word “binary digit”
- Smallest unit of computer memory.
- 1 or 0 are only values.
- Bit can represent only one value

### Bytes

- Byte = 8 bits
- Each keyword character is stored as a byte.
- Example = A -> 01000001

### Words

- Word = a sequence of N bits where N = 16, 32, 64 depending on the computer
- The number of bits a particular computer’s CPU can deal with in one go

## REPRESENTATION - CHARACTERS, INTEGER AND FRACTION

Everything represented by a computer is represented by binary sequences.

### Character Representation

All the characters on keyboard are given a character code. Two standards commonly use:

1. American Standard Code for Information Interchange (ASCII) - Giving a character set of 128 characters
2. Unicode - Giving a character set of 65,536 characters

## Integer and Fraction Representation

Use fixed number of bits to represent integer.

## Number System

- A number system in base r or radix r uses unique symbols for r digits. One or more digits are combined to get a number.
- The base of the number decides the valid digits that are used to make a number.
- In a number, the position of digit starts from the right-hand side of the number. The rightmost digit has position 0, the next digit on its left has position 1, and so on. The digits of a number have two kinds of values—

**Face value** and **Position value.**

The face value of a digit is the digit located at that position. For example, in decimal number 52, face value at position 0 is 2 and face value at position 1 is 5.

The position value of a digit is ($\text{base}^{\text{position}}$). For example, in decimal number 52, the position value of digit 2 is $10^0$ and the position value of digit 5 is $10^1$. Decimal numbers have a base of 10.

The number is calculated as the sum of, face value * $\text{base}^{\text{position}}$, of each of the digits. For decimal number 52, the number is $5*10^1 + 2*10^0 = 50 + 2 = 52$

In computers, we are concerned with four kinds of number systems, as follows—

- Decimal Number System —Base 10
- Binary Number System —Base 2
- Octal Number System —Base 8
- Hexadecimal Number System—Base 16

A number in a particular base is written as $(\text{number})_{\text{base}}$ of number For example, $(23)_{10}$ means that the number 23 is a decimal number, and $(345)_8$ shows that 345 is an octal number.

## Decimal Number System

- It consists of 10 digits—0, 1, 2, 3, 4, 5, 6, 7, 8 and 9.
- All numbers in this number system are represented as combination of digits 0—9. For example, 34, 5965 and 867321.

## Binary Number System

- The binary number system consists of two digits—0 and 1.

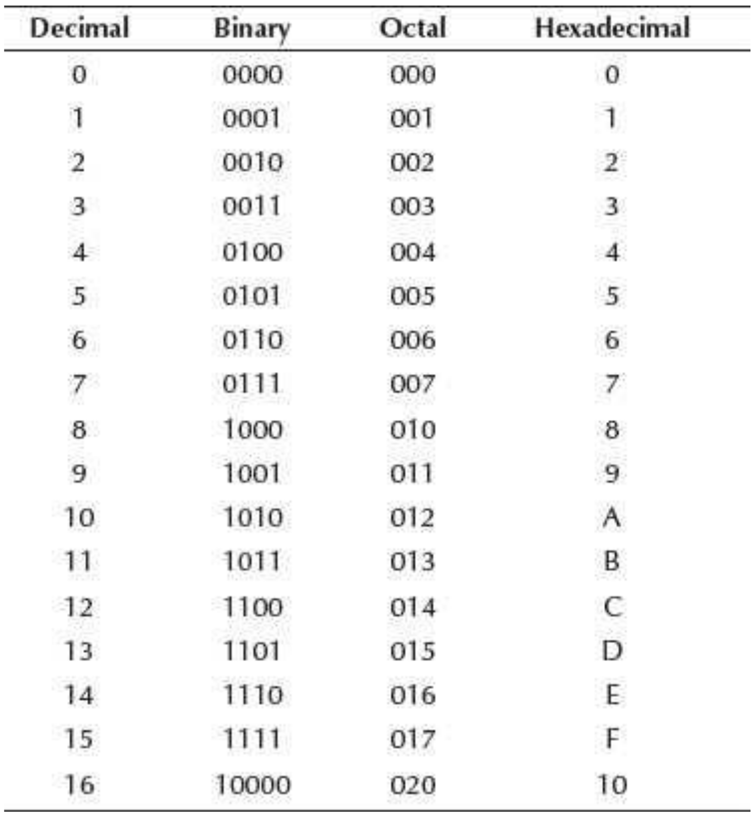
- All binary numbers are formed using combination of 0 and 1. For example, 1001, 11000011 and 10110101.

**Octal Number System**

- The octal number system consists of eight digits—0 to 7.
- All octal numbers are represented using these eight digits. For example, 273, 103, 2375, etc.

**Hexadecimal Number System**

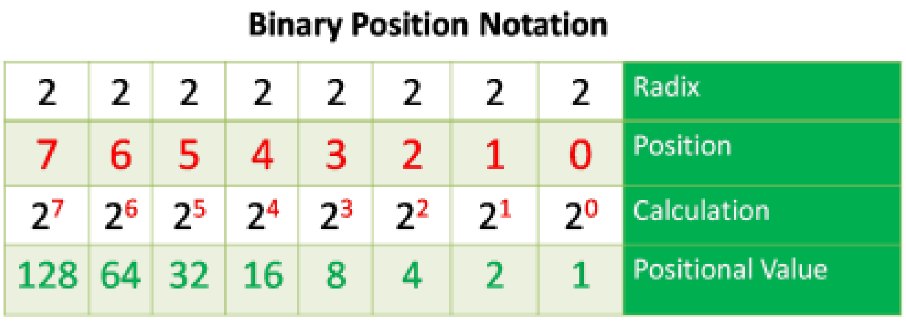
- The hexadecimal number system consists of sixteen digits—0 to 9, A, B, C, D, E, F, where (A is for 10, B is for 11, C-12, D-13, E-14, F-15).
- All hexadecimal numbers are represented using these 16 digits. For example, 3FA, 87B, 113, etc.



| Decimal | Binary | Octal | Hexadecimal |
|---|---|---|---|
| 0 | 0000 | 000 | 0 |
| 1 | 0001 | 001 | 1 |
| 2 | 0010 | 002 | 2 |
| 3 | 0011 | 003 | 3 |
| 4 | 0100 | 004 | 4 |
| 5 | 0101 | 005 | 5 |
| 6 | 0110 | 006 | 6 |
| 7 | 0111 | 007 | 7 |
| 8 | 1000 | 010 | 8 |
| 9 | 1001 | 011 | 9 |
| 10 | 1010 | 012 | A |
| 11 | 1011 | 013 | B |
| 12 | 1100 | 014 | C |
| 13 | 1101 | 015 | D |
| 14 | 1110 | 016 | E |
| 15 | 1111 | 017 | F |
| 16 | 10000 | 020 | 10 |

**Decimal, binary, octal and hexadecimal equivalents**

**Positional Notations**



**Binary Position Notation**

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | Radix |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | Position |
| $2^7$ | $2^6$ | $2^5$ | $2^4$ | $2^3$ | $2^2$ | $2^1$ | $2^0$ | Calculation |
| 128 | 64 | 32 | 16 | 8 | 4 | 2 | 1 | Positional Value |

**Decimal Position Notation**

| | | | | | |
|---|---|---|---|---|---|
| 10 | 10 | 10 | 10 | 10 | Radix |
| 4 | 3 | 2 | 1 | 0 | Position |
| $10^4$ | $10^3$ | $10^2$ | $10^1$ | $10^0$ | Calculation |
| 10,000 | 1,000 | 100 | 10 | 1 | Positional Value |

**Octal Position Notation**

| | | | | |
|---|---|---|---|---|
| 8 | 8 | 8 | 8 | Radix |
| 3 | 2 | 1 | 0 | Position |
| $8^3$ | $8^2$ | $8^1$ | $8^0$ | Calculation |
| 512 | 64 | 8 | 1 | Positional Value |

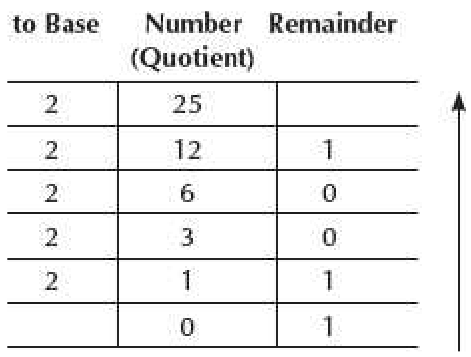
**Hexadecimal Position Notation**

| | | | | |
|---|---|---|---|---|
| 16 | 16 | 16 | 16 | Radix |
| 3 | 2 | 1 | 0 | Position |
| $16^3$ | $16^2$ | $16^1$ | $16^0$ | Calculation |
| 4096 | 256 | 16 | 1 | Positional Value |

## CONVERTING DECIMAL *INTEGER* TO BINARY, OCTAL, HEXADECIMAL

Decimal integer is converted to any other base, by using Division operation

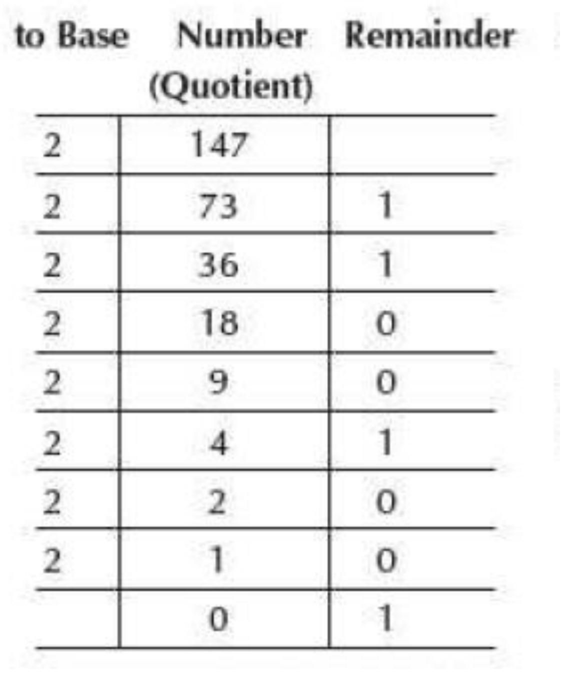
- Technique to convert into binary (Base 2)
  - Make a table. Write the number in center and toBase on the left side.
  - Divide number by two, keep track of the remainder
  - First remainder is bit 0, Second remainder is bit 1 etc.
  - Continue dividing till quotient is 0
  - Write the digits in remainder from downwards to upwards



| to Base | Number (Quotient) | Remainder |
|---|---|---|
| 2 | 25 | |
| 2 | 12 | 1 |
| 2 | 6 | 0 |
| 2 | 3 | 0 |
| 2 | 1 | 1 |
| | 0 | 1 |

The binary equivalent of number $(25)_{10}$ is $(11001)_2$

**Example: Convert 147 from Base 10 to Base 2**



| to Base | Number (Quotient) | Remainder |
|---|---|---|
| 2 | 147 | |
| 2 | 73 | 1 |
| 2 | 36 | 1 |
| 2 | 18 | 0 |
| 2 | 9 | 0 |
| 2 | 4 | 1 |
| 2 | 2 | 0 |
| 2 | 1 | 0 |
| | 0 | 1 |

The Binary equivalent of number $(147)_{10}$ is $(10010011)_2$

1. Example: Convert 94 from Base 10 to Base 2
2. Example: Convert 396 from Base 10 to Base 2

**CONVERTING DECIMAL FRACTION TO BINARY, OCTAL, HEXADECIMAL**

Decimal fraction is converted to any other base, by using Multiplication operation

Steps for conversion of a decimal fraction to any other base are –

1. Multiply the fractional number with the to Base, to get a resulting number.
2. The resulting number has two parts, non-fractional part and fractional part.
3. Record the non-fractional part of the resulting number.

4. Repeat the above steps at least four times.

5. Write the digits in the non-fractional part starting from upwards to downwards.

**Example: Convert 0.2345 from Base 10 to Base 2.**

- 0.2345

  $0.2345 * 2 = 0.4690$

  $0.4690 * 2 = 0.9380$

  $0.9380 * 2 = 1.8760$

  $0.8760 * 2 = 1.7520$

  $0.7520 * 2 = 1.5040$

  $0.5040 * 2 = 1.0080$ $\downarrow$

$(0.2345)_{10} = (0.001111)_2$

**Convert 0.865 from Base 10 to Base 2**

- 0.865

  $0.865 * 2 = 1.730$

  $0.730 * 2 = 1.460$

  $0.460 * 2 = 0.920$

  $0.920 * 2 = 1.840$

  $0.840 * 2 = 1.680$

  $0.680 * 2 = 1.360$ $\downarrow$

$(0.865)_{10} = (0.110111)_2$

1. **$(0.4673)_{10}$**
2. **$(34.4673)_{10}$**

## CONVERTING BINARY TO DECIMAL

Binary is converted to any other base, by using Multiplication operation

We need face value and position value for conversion

Technique to convert into Decimal (Base 10)

- Find sum of the *Face value * (from Base)* $^{Position}$ for each digit in number
- Multiply each digit by $2^n$, where n is the "position" of the digit

In non-fractional number the rightmost digit has position 0 and position increases as we go towards left

In fractional number the first digit to the right of the decimal point has position -1 and it decreases as we go towards the right

Example 1: $(1011)_2$

$1011_2 =$

$= (1 * 8) + (0 * 4) + (1 * 2) + (1 * 1)$

$= 8 + 0 + 2 + 1$

$= (11)_{10}$

Example 2: $(.1101)_2$

$.1101_2 =$

$= (1/2) + (1/4) + (0) + (1/16)$

$= 13/16$

$= (0.8125)_{10}$

1. $(101011)_2$
2. $(1011.1001)_2$
3. $(675)_{10}$
4. $(150.64)_{10}$
5. $(110000111)_2$
6. $(11001.1101)_2$

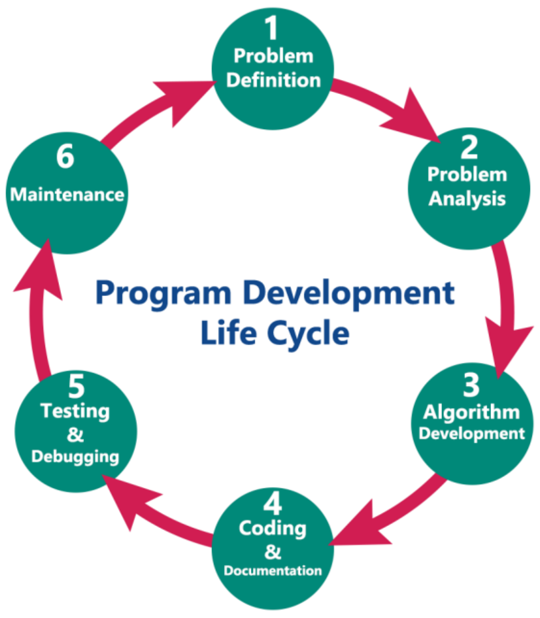
## Program Development Cycle

The program development process is part of the software lifecycle, characterized by the following stages:

- Requirements analysis
- Design
- Coding
- Testing
- Implementation and support
- Documentation

Program development life cycle contains 6 phases, they are as follows –

1. Problem Definition
2. Problem Analysis
3. Algorithm Development
4. Coding & Documentation
5. Testing & Debugging
6. Maintenance



Problem Definition:

- In this phase, we define the problem statement and we decide the boundaries of the problem.
- In this phase we need to understand the problem statement, what is our requirement, what should be the output of the problem solution.
- These are defined in this first phase of the program development life cycle.

Problem Analysis:

- In phase 2, we determine the requirements like variables, functions, etc. to solve the problem.
- That means we gather the required resources to solve the problem defined in the problem definition phase.
- We also determine the constraints of the solution.

Algorithm Development:

- During this phase, we develop a step-by-step procedure to solve the problem using the specification given in the previous phase.
- This phase is very important for program development.
- That means we write the solution in step-by-step statements.

Coding & Documentation:

- This phase uses a programming language to write or implement actual programming instructions for the steps defined in the previous phase.
- In this phase, we construct actual program.
- That means we write the program to solve the given problem using programming languages like C, C++, Java etc.,

Testing & Debugging:

- During this phase, we check whether the code written in previous step is solving the specified problem or not.
- That means we test the program whether it is solving the problem for various input data values or not.
- We also test that whether it is providing the desired output or not.

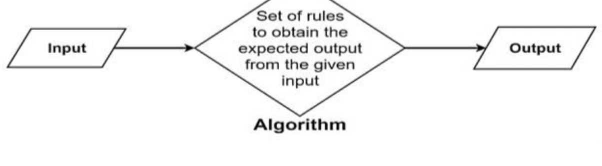
Maintenance:

- During this phase, the program is actively used by the users.
- If any enhancements found in this phase, all the phases are to be repeated again to make the enhancements.
- That means in this phase, the solution (program) is used by the end user.
- If the user encounters any problem or wants any enhancement, then we need to repeat all the phases from the starting, so that the encountered problem is solved or enhancement is added.

**Basics of Algorithm**

**Algorithm –**

- A set of finite rules or instructions to be followed in calculations or other problem-solving operations.
- A procedure for solving a mathematical problem in a finite number of steps that frequently involves recursive operations.
- Algorithm is an ordered sequence of finite, well defined, clear-cut instructions for completing a task. It is English like representation.


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

What is the need for algorithms?

- Algorithms are necessary for solving complex problems efficiently and effectively.
- They help to automate processes and make them more reliable, faster, and easier to perform.
- Algorithms also enable computers to perform tasks that would be difficult or impossible for humans to do manually.
- It is used in various fields such as mathematics, computer science, engineering, finance, and many others to optimize processes, analyze data, make predictions, and provide solutions to problems.

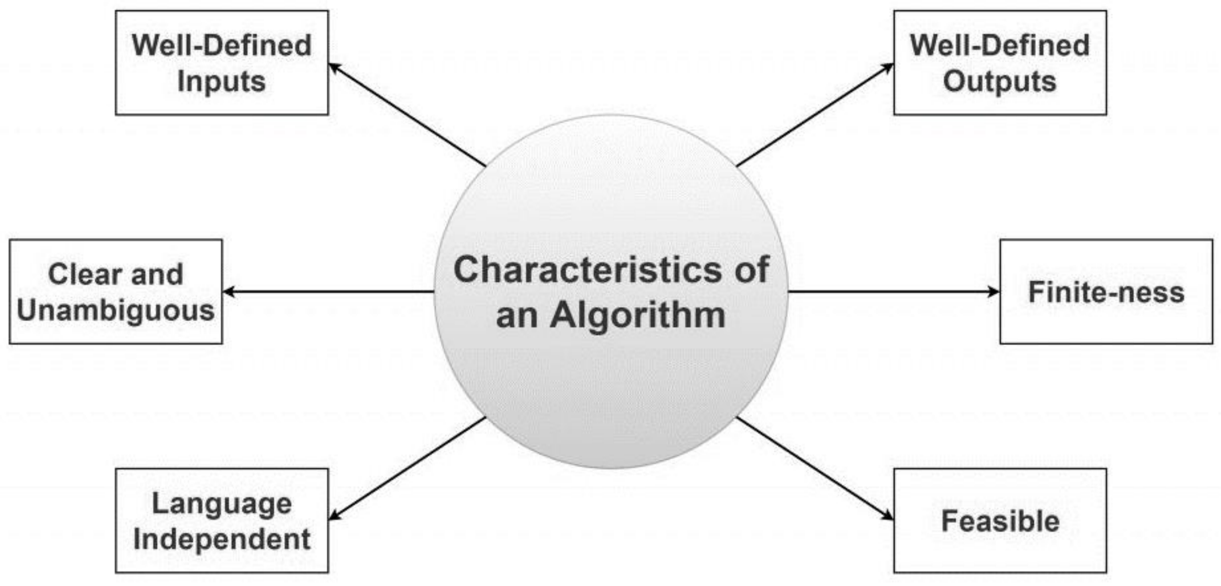
Algorithm for addition of two numbers –

- Step 1: Start.
- Step 2: Declare variables a, b and sum.
- Step 3: Read values a and b.
- Step 4: Add a and b and assign the result to sum.
- sum = a + b
- Step 5: Display sum.
- Step 6: Stop.

Algorithm to display your name and college name –

- Step 1: Start.
- Step 2: Declare variables name and college_name.
- Step 3: Read values name and college_name.
- Step 4: Display name and college_name.
- Step 5: Stop.

What are the Characteristics of an Algorithm?



Properties of Algorithm –

- It should terminate after a finite time.
- It should produce at least one output. It should take zero or more input.
- It should be deterministic means giving the same output for the same input case. Every step in the algorithm must be effective i.e., every step should do some work.

Advantages of Algorithms–

- It is easy to understand.
- An algorithm is a step-wise representation of a solution to a given problem.
- In Algorithm the problem is broken down into smaller pieces or steps hence, it is easier for the programmer to convert it into an actual program.

Disadvantages of Algorithms–

- Writing an algorithm takes a long time so it is time-consuming.
- Understanding complex logic through algorithms can be very difficult.
- Branching and Looping statements are difficult to show in Algorithms.

**Examples:**

- Algorithm to find greatest among 3 numbers
  - Algorithm 1:
    1. Start.
    2. Read three numbers A, B, C.
    3. Compare A and B.
    - If A is greater perform step 4, else step 5.
    4. Compare A and C.
    - If A is greater, output “A is greatest” else output “C is greatest”.
    - Perform Step 6.
    5. Compare B and C.
    - If B is greater, output “B is greatest” else output “C is greatest”.
    6. Stop.
  - Algorithm 2:
    1. Start.
    2. Read the three numbers A, B, C.
    3. Compare A and B.
    - If A is greater, store A in MAX, else store B in MAX.
    4. Compare MAX and C.
    - If MAX is greater, output “MAX is greatest” else output “C is greatest”.
    5. Stop.

First algorithm has a greater number of comparisons, whereas in the second algorithm an additional variable MAX is used.

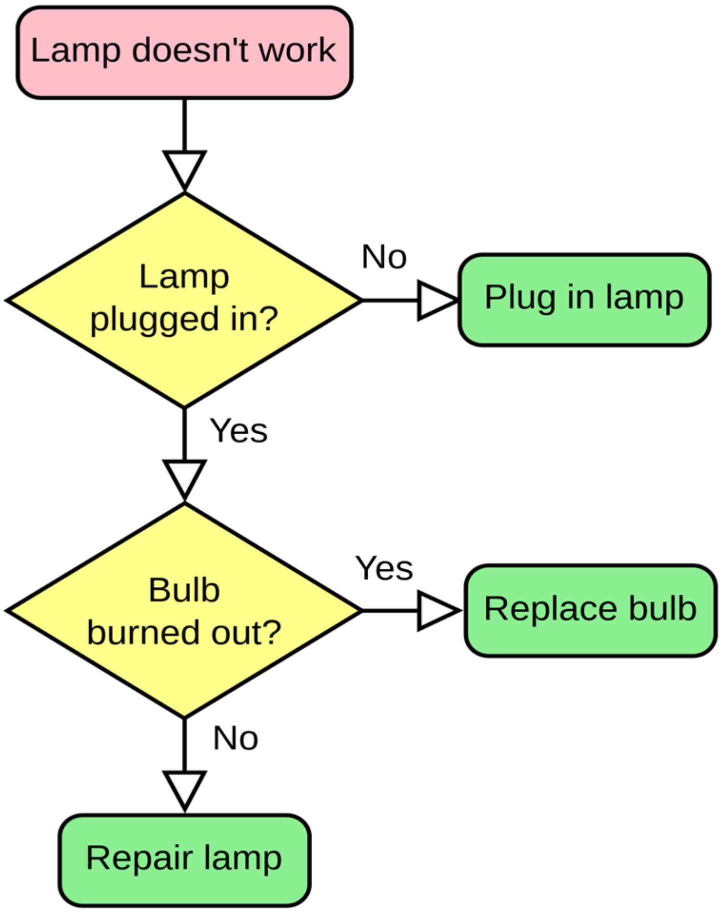
## Control Structures

- Control structures specify the statements to be executed and the order of execution of statements.
- Three kinds of control structures –
    - Sequential
    - Selection
    - Iterative (loop)
- Flowchart and Pseudo code use control structures for representation.

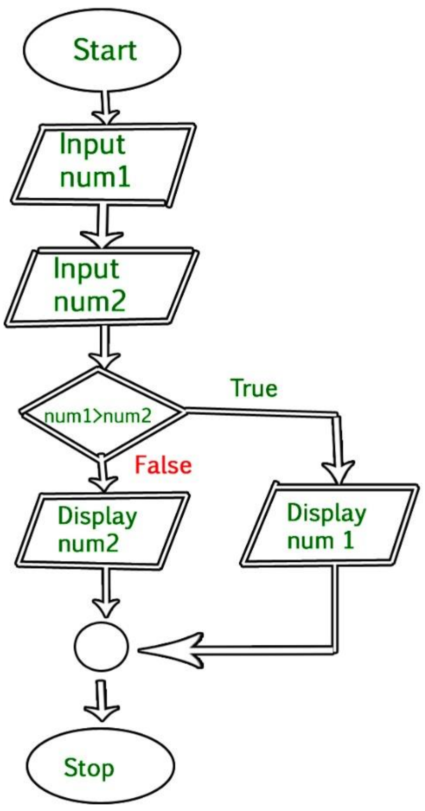
## Flowchart

### What is a Flowchart?

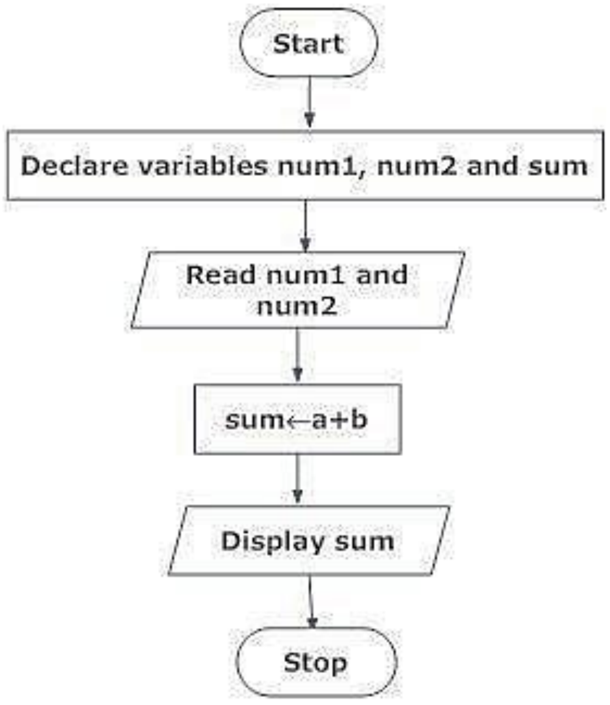
- Flowchart is a graphical representation of an algorithm. Programmers often use it as a program-planning tool to solve a problem.
- It makes use of symbols which are connected among them to indicate the flow of information and processing.
- The process of drawing a flowchart for an algorithm is known as "flowcharting".
- Example:

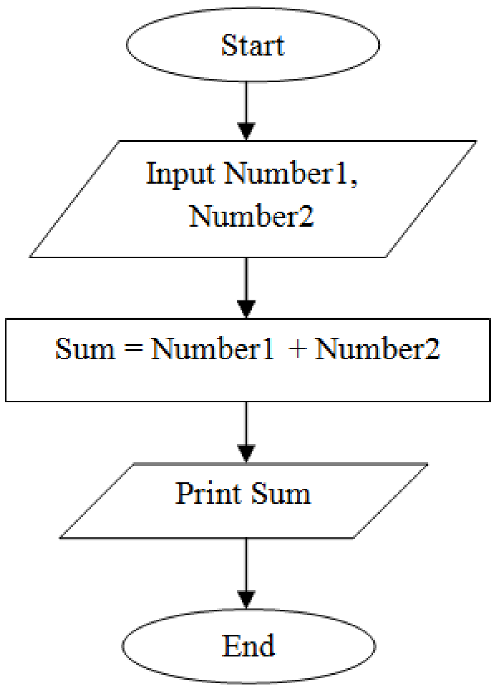


- Example:
    - Draw a flowchart to input two numbers from the user and display the largest of two numbers.

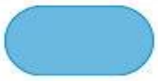
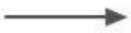
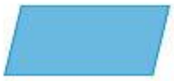
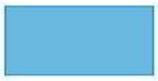
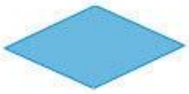


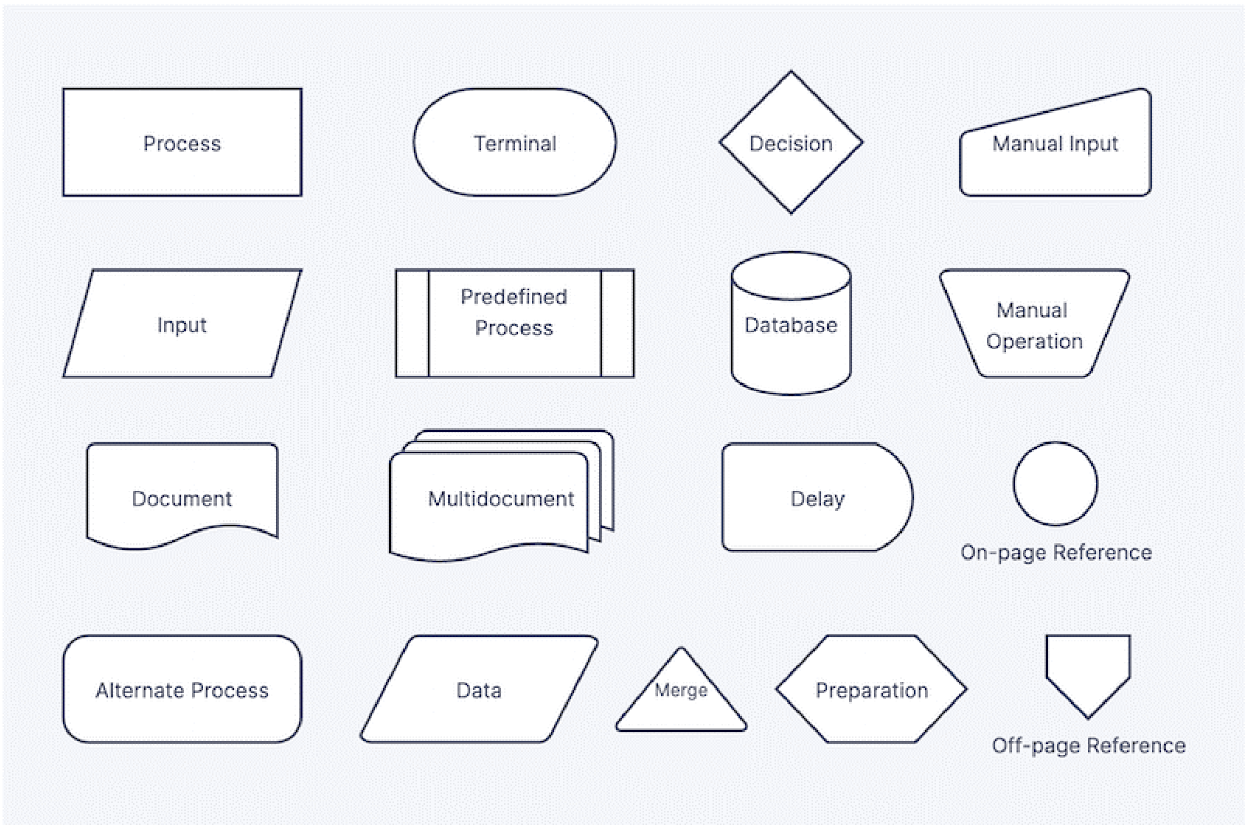
- Examples:





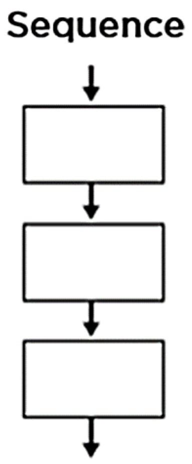
Basic Symbols used in Flowchart Designs:

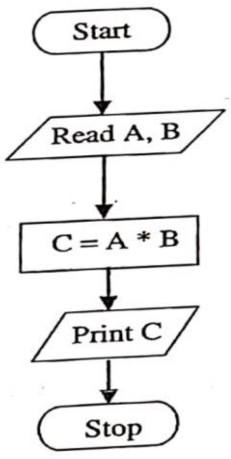
| Symbol | Name | Function |
|---|---|---|
|  | Start/end | An oval represents a start or end point |
|  | Arrows | A line is a connector that shows relationships between the representative shapes |
|  | Input/Output | A parallelogram represents input or output |
|  | Process | A rectagle represents a process |
|  | Decision | A diamond indicates a decision |



Rules For Creating Flowcharts –

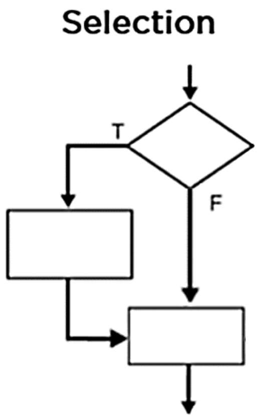
- A flowchart is a graphical representation of an algorithm.
- It should follow some rules while creating a flowchart
- Flowchart opening statement must be 'start' keyword.
- Flowchart ending statement must be 'end' keyword.
- The direction of flow in a flowchart must be from top to bottom and left to right.
- The relevant symbol must be used while drawing a flowchart.
- All symbols in the flowchart must be connected with an arrow line.
- The decision symbol in the flowchart is associated with the arrow line.

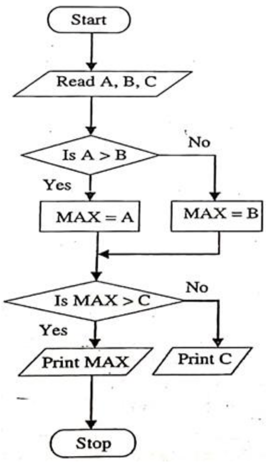




•

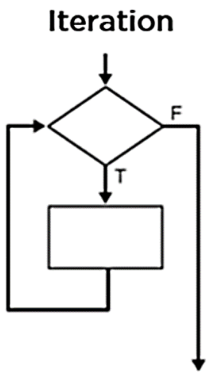
•

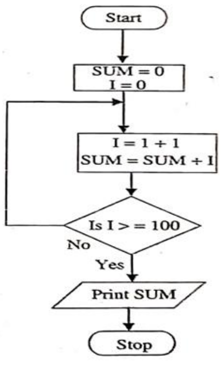




•

•





•

Advantages of Flowchart:

- Flowcharts are a better way of communicating the logic of the system.
- Flowcharts act as a guide for blueprint during program designed.
- Flowcharts help in debugging process.
- With the help of flowcharts programs can be easily analyzed.
- Flowcharts serve as a good proper documentation.
- Easy to trace errors in the software.
- Easy to understand.
- The flowchart can be reused for inconvenience in the future.
- It helps to provide correct logic.

Disadvantages of Flowchart:

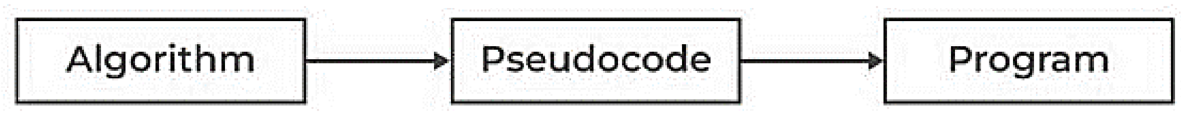
- It is difficult to draw flowcharts for large and complex programs.
- There is no standard to determine the amount of detail.
- Difficult to reproduce the flowcharts.
- It is very difficult to modify the Flowchart.
- Making a flowchart is costly.
- Some developer thinks that it is waste of time.
- It makes software development processes low.
- If changes are done in software, then the flowchart must be redrawn.

## Pseudo Code

- A Pseudocode is defined as a step-by-step description of an algorithm.
- Pseudocode does not use any programming language in its representation.
- It uses the simple English language text as it is used for human understanding rather than machine reading.
- Pseudocode is the intermediate state between an idea and its implementation (code) in a high-level language.

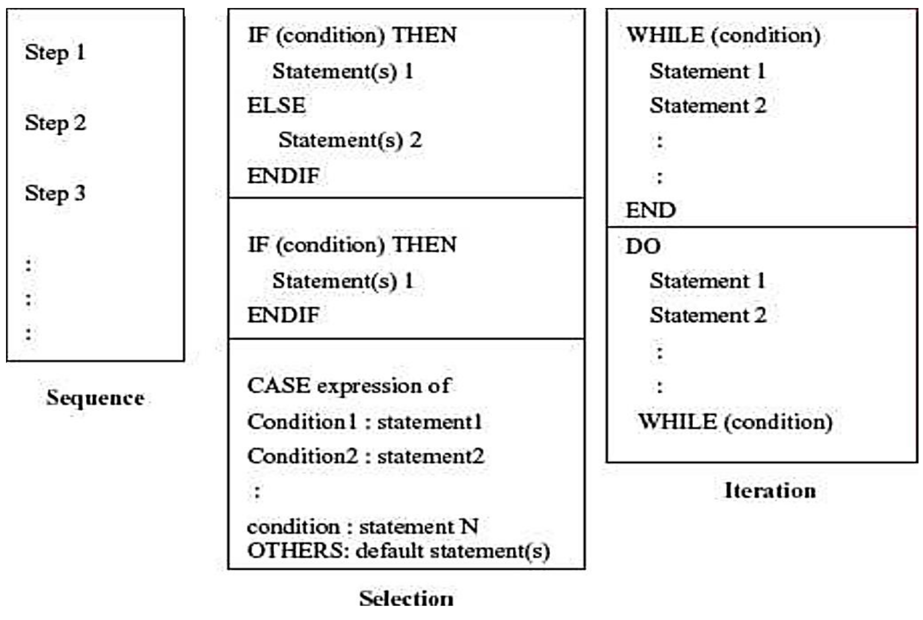
What is the need for Pseudocode?

- Pseudocode is an important part of designing an algorithm, it helps the programmer in planning the solution to the problem as well as the reader in understanding the approach to the problem.
- Pseudocode is an intermediate state between algorithm and program that plays supports the transition of the algorithm into the program.



How to write Pseudocode?

- Pseudo code is written using structured English.
- Some terms are commonly used to represent the various actions (for example: INPUT, OUTPUT, COMPUTE etc.)
- Control structures also used while writing a pseudo code.
- Before writing the pseudocode of any algorithm the following points must be kept in mind –
- Organize the sequence of tasks and write the pseudocode accordingly.
  - At first, establishes the main goal.
  - E.g. - This program will print largest number among two numbers.



```
Step 1

Step 2

Step 3

:
:
:
```

Sequence

```
IF (condition) THEN
    Statement(s) 1
ELSE
    Statement(s) 2
ENDIF
```

```
IF (condition) THEN
    Statement(s) 1
ENDIF
```

```
CASE expression of
Condition1 : statement1
Condition2 : statement2
:
condition : statement N
OTHERS: default statement(s)
```

Selection

```
WHILE (condition)
    Statement 1
    Statement 2
    :
    :
END
```

```
DO
    Statement 1
    Statement 2
    :
    :
  WHILE (condition)
```

Iteration

```
IF student's grade is greater than or equal to 40

        PRINT "passed"

ELSE

        PRINT "failed“
```

```
CASE of age

        0 to 17         DISPLAY "You can't vote."

        18 to 64 DISPLAY "You're in your working years."

        65 +   DISPLAY "You should be retired."

ENDCASE
```

```
COUNT assigned zero
    WHILE count < 5
        DISPLAY "I love computers!"        INCREMENT count
    END WHILE
```

```
COUNT assigned seven
    DO
        DISPLAY "COUNT day to go..!!"
        DECREMENT count
    WHILE count > zero
```

How to write Pseudocode?

- Rules –
    - Use standard programming structures such as if-else, for, while, and cases the way we use them in programming.
    - Indent the statements if-else, for, while loops as they are indented in a program, it helps to comprehend the decision control and execution mechanism.
    - It also improves readability to a great extent.
    - Use appropriate naming conventions.
        - The human tendency follows the approach of following what we see.
        - If a programmer goes through a pseudo code, his approach will be the same as per that, so the naming must be simple and distinct.
    - Reserved commands or keywords must be represented in capital letters.
        - Example: if you are writing IF…ELSE statements then make sure IF and ELSE be in capital letters.
    - Check whether all the sections of a pseudo code are complete, finite, and clear to understand and comprehend.
        - Also, explain everything that is going to happen in the actual code.
    - Don't write the pseudocode in a programming language.
        - It is necessary that the pseudocode is simple and easy to understand even for a layman or client, minimizing the use of technical terms.

- Examples –

```
READ values of A and B
COMPUTE C by multiplying A with B
PRINT the result C
STOP
```

(i) Find product of any two numbers

```
READ values of A, B, C
IF A is greater than B THEN
        ASSIGN A to MAX
ELSE
        ASSIGN B to MAX
IF MAX is greater than C THEN
        PRINT MAX is greatest
ELSE
        PRINT C is greatest
STOP
```

(ii) Find maximum of any three numbers

```
INITIALIZE SUM to zero
INIT IALIZE I to zero
DO WHILE (I less than 100)
    INCREMENT I
    ADD I to SUM and store in SUM
PRINT SUM
STOP
```

(iii) Find sum of first 100 integers

**Algorithm v/s Pseudocode**

| Algorithm | Pseudocode |
|---|---|
| An Algorithm is used to provide a solution to a particular problem in form of a well-defined step-based form. | A Pseudocode is a step-by-step description of an algorithm in code-like structure using plain English text. |
| An algorithm only uses simple English words | Pseudocode also uses reserved keywords like if-else, for, while, etc. |
| These are a sequence of steps of a solution to a problem | These are fake codes as the word pseudo means fake, using code like structure and plain English text |
| There are no rules to writing algorithms | There are certain rules for writing pseudocode |
| Algorithms can be considered pseudocode | Pseudocode cannot be considered an algorithm |
| It is difficult to understand and interpret | It is easy to understand and interpret |

**Flowchart v/s Pseudocode**

| Flowchart | Pseudocode |
|---|---|
| A Flowchart is pictorial representation of flow of an algorithm. | A Pseudocode is a step-by-step description of an algorithm in code like structure using plain English text. |
| A Flowchart uses standard symbols for input, output decisions and start stop statements. Only uses different shapes like box, circle and arrow. | Pseudocode uses reserved keywords like if-else, for, while, etc. |
| This is a way of visually representing data, these are nothing but the graphical representation of the algorithm for a better understanding of the code | These are fake codes as the word pseudo means fake, using code like structure but plain English text instead of programming language |
| Flowcharts are good for documentation | Pseudocode is better suited for the purpose of understanding |

## Programming

What is a Program?

- A program is a set of instructions that tell a computer what to do.
- A computer cannot do anything unless it has a program to tell it what to do.
- Programs are used to operate the components of a computer, solve problems or satisfy a want/need.
- Programs can also be called software.

What is a Programming?

- Just like we use Hindi or English to communicate with each other, we use a Programming language to communicate with the computer.
- Programming is a way to instruct the computer to perform various tasks.

What is a Programming Language?

- A programming language is a set of rules that provides a way of telling a computer what operations to perform.
- A programming language is a set of rules for communicating an algorithm.
- Series of instructions to a computer to accomplish a task.
  - Instructions must be written in a way the computer can understand.
  - Programming languages are used to write programs.

## C Programming

- About C Programming
  - Procedural Language
    - Instructions in a C program are executed step by step.
  - Portable
    - You can move C programs from one platform to another, and run it without any or minimal changes.
  - Speed
    - C programming is faster than most programming languages like Java, Python, etc.
  - General Purpose
    - C programming can be used to develop operating systems, embedded systems, databases, and so on.
  - C is a general purpose, procedural, high level programming language used in the development of computer software and applications, system programming, games, web development, and more.
  - C language was developed by Dennis M. Ritchie at the Bell Telephone Laboratories in 1972.

- It is a powerful and flexible language which was first developed for the programming of the UNIX operating System.

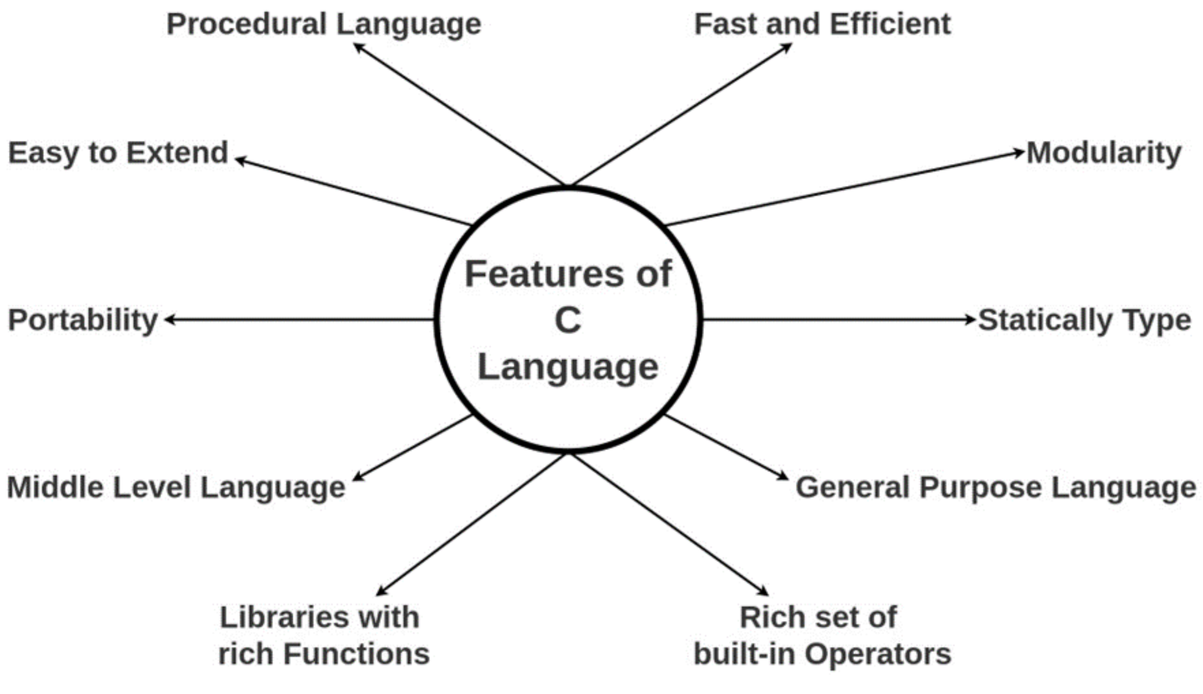
What is C?

- C is one of the most widely used programming language.
- C programming language is known for its simplicity and efficiency.
- It is the best choice to start with programming as it gives you a foundational understanding of programming.
- The main features of C language include lowlevel access to memory, a simple set of keywords, and a clean style, these features make C language suitable for system programming like an operating system or compiler development.

Why Learn C Programming?

- C helps you to understand the internal architecture of a computer, how computer stores and retrieves information.
- After learning C, it will be much easier to learn other programming languages like Java, Python, etc.
- Opportunity to work on open-source projects.
- Some of the largest opensource projects such as Linux kernel, Python interpreter, SQLite database, etc. are written in C programming.

What are the Most Important Features of C Language?



The main features of the C language include –

- General Purpose and Portable
- Low level Memory Access
- Fast Speed
- Clean Syntax

These features make the C language suitable for system programming like an operating system or compiler development.

Why Should We Learn C?

- Many languages have borrowed syntax/features directly or indirectly from the C language.
- Like syntax of Java, PHP, JavaScript, and many other languages are mainly based on the C language.
- C++ is nearly a superset of C language (Only a few programs may compile in C, but not in C++).
- So, if a person learns C programming first, it will help him to learn any modern programming language as well.
- As learning C help to understand a lot of the underlying architecture of the operating system.
- Like pointers, working with memory locations, etc.

Application of C

C is widely used for developing –

- Operating Systems
- Embedded Systems
- System Software
- Networking Applications
- Database systems
- Gaming
- Artificial Intelligence and machine learning applications
- Scientific applications
- Financial applications

Executing a ‘C’ Program

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

- Creating the program.
- Save it.
- Compile the program.
- Linking the program.
- Run the program.

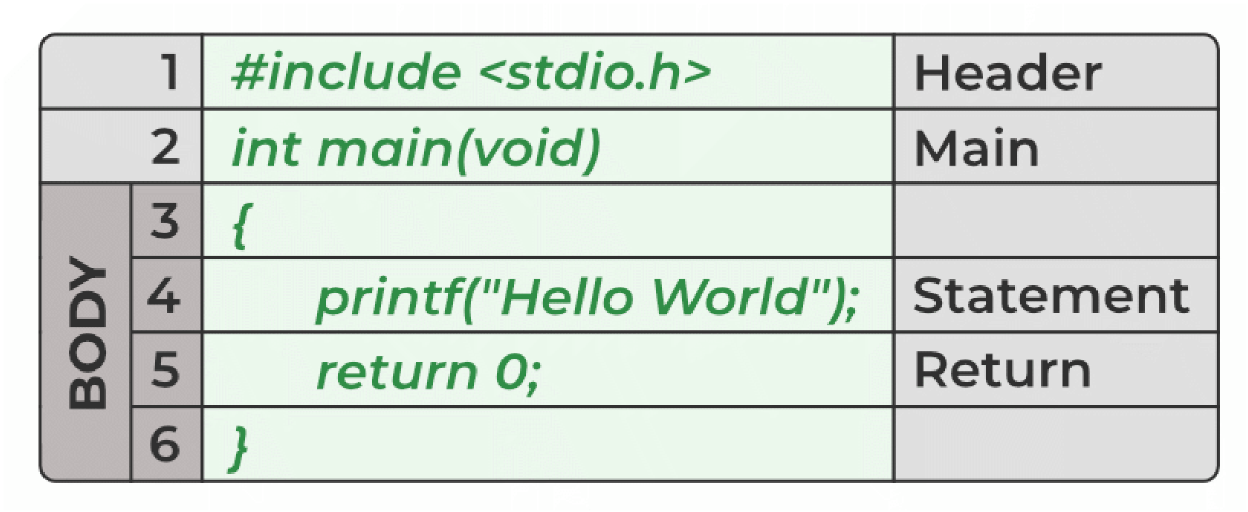
Beginning with C programming

```
#include <stdio.h>
int main() {
int a = 10;
printf("%d", a);
return 0;
}
```

Output = 10

Let’s analyze the structure of our program line by line.

**Structure of the C program**



| | | | |
|---|---|---|---|
| | 1 | #include <stdio.h> | Header |
| | 2 | int main(void) | Main |
| BODY | 3 | { | |
| | 4 | printf("Hello World"); | Statement |
| | 5 | return 0; | Return |
| | 6 | } | |

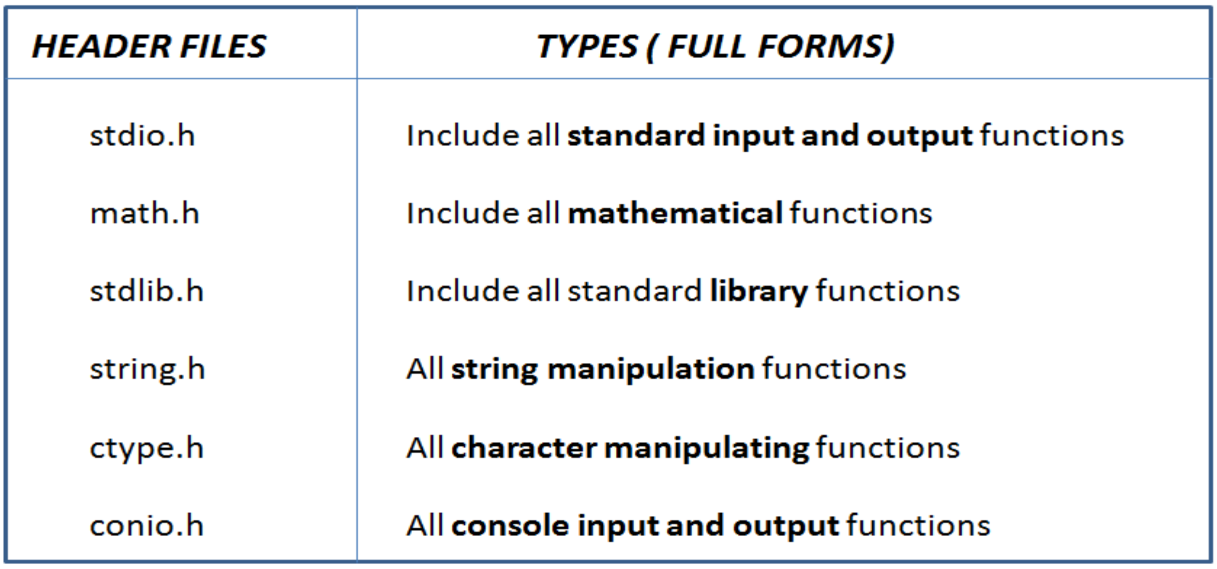
**Components**

```
preprocessor directives
global declarations
main ()
{
   local variable deceleration
   statement sequences
   function invoking
}
```

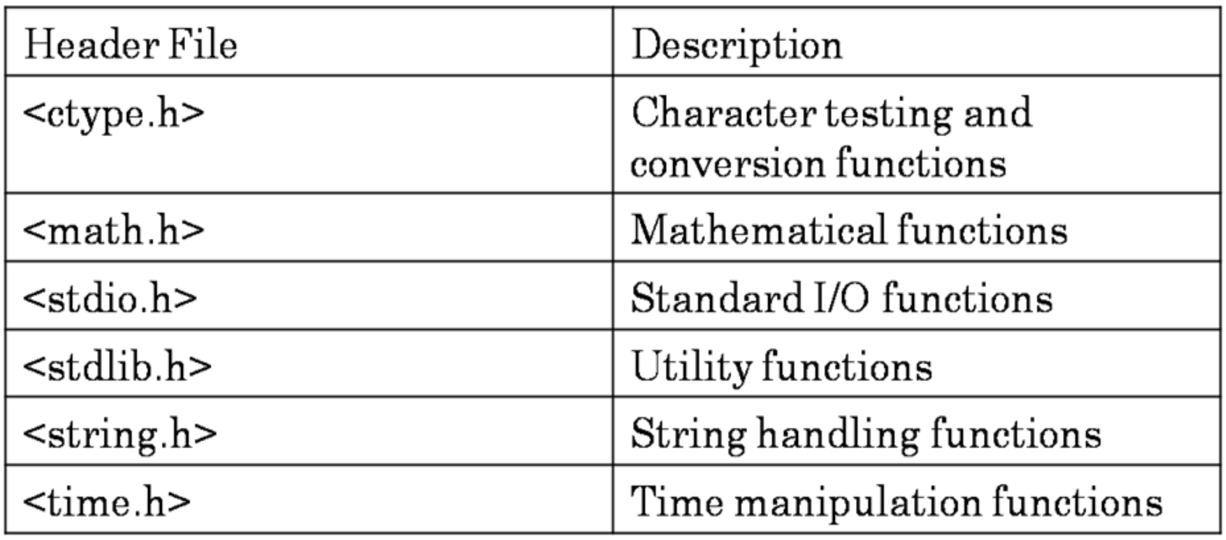
## Components of a C Program

1. Header Files Inclusion – Line 1 [#include <stdio.h>]

- The first and foremost component is the inclusion of the Header files in a C program.
- A header file is a file with extension .h which contains C function declarations and macro definitions to be shared between several source files.
- All lines that start with # are processed by a preprocessor which is a program invoked by the compiler.
- E.g., the preprocessor copies the preprocessed code of stdio.h to our file.
    - The .h files are called header files in C.



| *HEADER FILES* | *TYPES ( FULL FORMS)* |
|---|---|
| stdio.h | Include all **standard input and output** functions |
| math.h | Include all **mathematical** functions |
| stdlib.h | Include all standard **library** functions |
| string.h | All **string manipulation** functions |
| ctype.h | All **character manipulating** functions |
| conio.h | All **console input and output** functions |



| Header File | Description |
|---|---|
| <ctype.h> | Character testing and conversion functions |
| <math.h> | Mathematical functions |
| <stdio.h> | Standard I/O functions |
| <stdlib.h> | Utility functions |
| <string.h> | String handling functions |
| <time.h> | Time manipulation functions |

2. Main Method Declaration – Line 2 [int main ()]

- The next part of a C program is to declare the main () function.
- It is the entry point of a C program and the execution typically begins with the first line of the main ().
- The empty brackets indicate that the main doesn't take any parameter.
- The int that was written before the main indicates the return type of main().
    - The value returned by the main indicates the status of program termination.

3. Body of Main Method – Line 3 to Line 6 [enclosed in {}]

- The body of a function in the C program refers to statements that are a part of that function.
- It can be anything like manipulations, searching, sorting, printing, etc.
- A pair of curly brackets define the body of a function.
    - All functions must start and end with curly brackets.

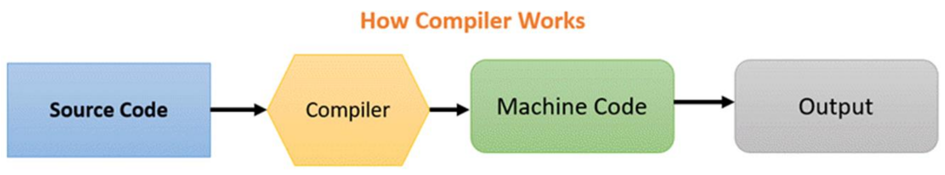
4. Statement – Line 4 [printf("Hello World");]

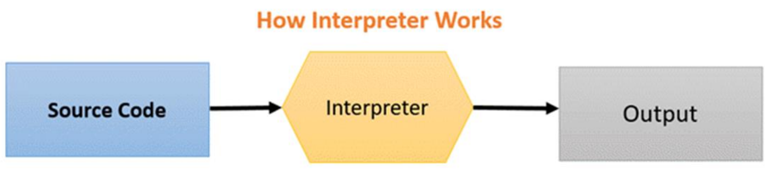
- Statements are the instructions given to the compiler.
- In C, a statement is always terminated by a semicolon (;).
- In this particular case, we use printf() function to instruct the compiler to display "Hello World" text on the screen.

5. Return Statement – Line 5 [return 0;]

- The last part of any C function is the return statement.
- The return statement refers to the return values from a function.
- This return statement and return value depend upon the return type of the function.
- The returned value may be used by an operating system to know the termination status of your program.
- The value 0 typically means successful termination.

**Compiler & Interpreter**

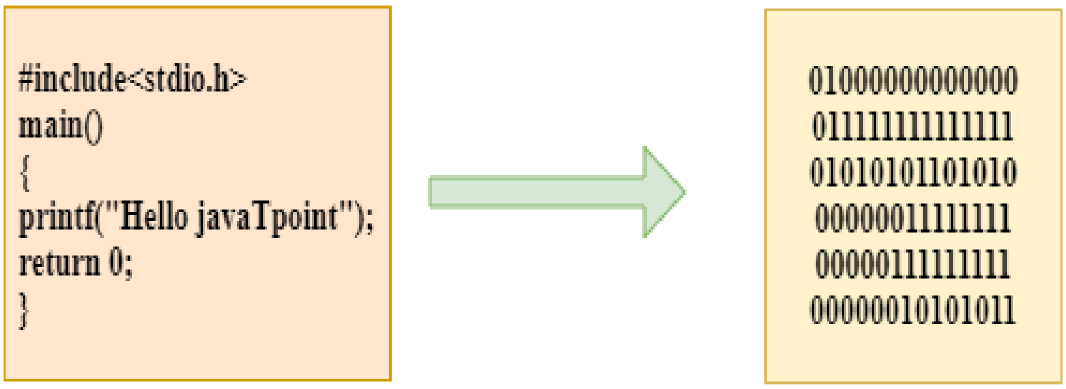




- Compiler and Interpreter are two different ways to translate a program from programming language to machine language/code.
- Machine code / language –
    - It is the elemental language of computers.
    - It is read by the computer's central processing unit (CPU), is composed of digital binary numbers and looks like a very long sequence of zeros and ones.

### A compiler

- It takes entire program and converts it into object code which is typically stored in a file.
- The object code is also referred as binary code and can be directly executed by the machine after linking.



```
#include<stdio.h>
main()
{
printf("Hello javaTpoint");
return 0;
}
```

```
01000000000000
01111111111111
01010101101010
00000011111111
00000111111111
00000010101011
```

### An interpreter

- It directly executes instructions written in a programming or scripting language without previously converting them to an object code or machine code.

### Executing a ‘C’ Program

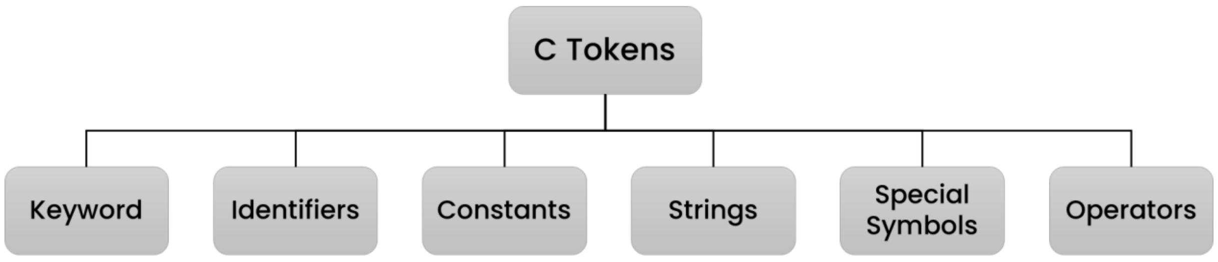
- Before going to write sample C programs we need to have a basic knowledge of the following –
    - C’s Character set
    - C’s Tokens

### C’s Character Set

- The only characters that ‘C’ uses for its programs are –
- Letters: A-Z all alphabets, a-z all alphabets
- Digits: 0-9 digits
- Special Characters: # % & ! _ ,- *+ () & . , : ; ' $ “+ - / * =
- White Spaces: blank space, horizontal tab, new line etc.

## 'C' Tokens

The smallest individual units in a C program are known as tokens



C has 6 different types of tokens –

- Keywords [e.g. float, int, while]
- Identifiers [e.g. sum, amount]
- Constants [e.g. -25.6, 100]
- Strings [e.g. "SMIT", "year"]
- Special Symbols [e.g. {}, [ ] ]
- Operators [e.g. +, -, *]

## Keywords

- Keywords are words that have special meaning to the C compiler.
- Their meaning cannot be changed.
- Serve as basic building blocks for program statements.
- All keywords are written in only lowercase.

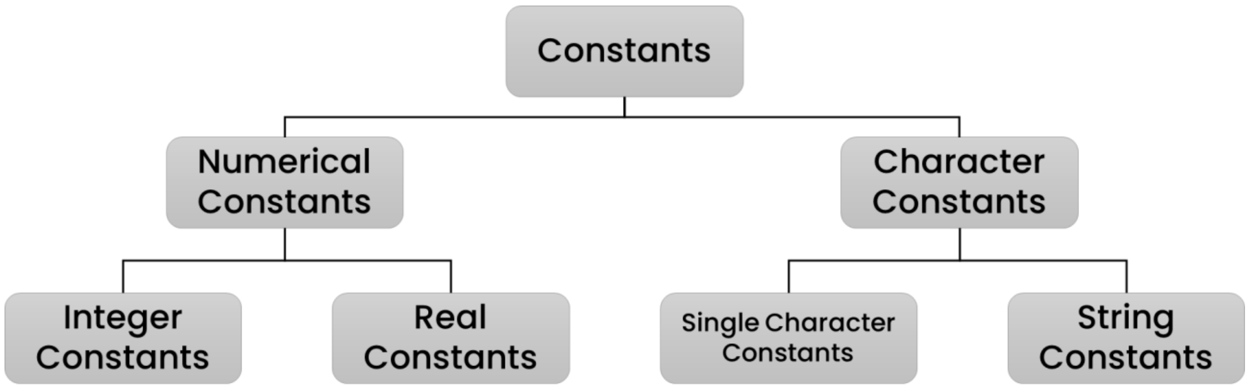
| | | | |
|---|---|---|---|
| auto | break | case | char |
| const | continue | default | do |
| double | else | enum | extern |
| float | for | goto | if |
| int | long | register | return |
| short | signed | sizeof | static |
| struct | switch | typedef | union |
| unsigned | void | volatile | while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

## Identifiers

- Identifiers are user-defined names to represent parts of the program: variables, functions, array etc.
- Cannot use C keywords as identifiers.
- Must begin with alpha characters.
- Must consist of only letters, digits or underscore ( _ ).
- Only first 31 characters are significant.
- Must NOT contain spaces ( ).

## Constants

- Constants in C are the fixed values that do not change during the execution of a program.



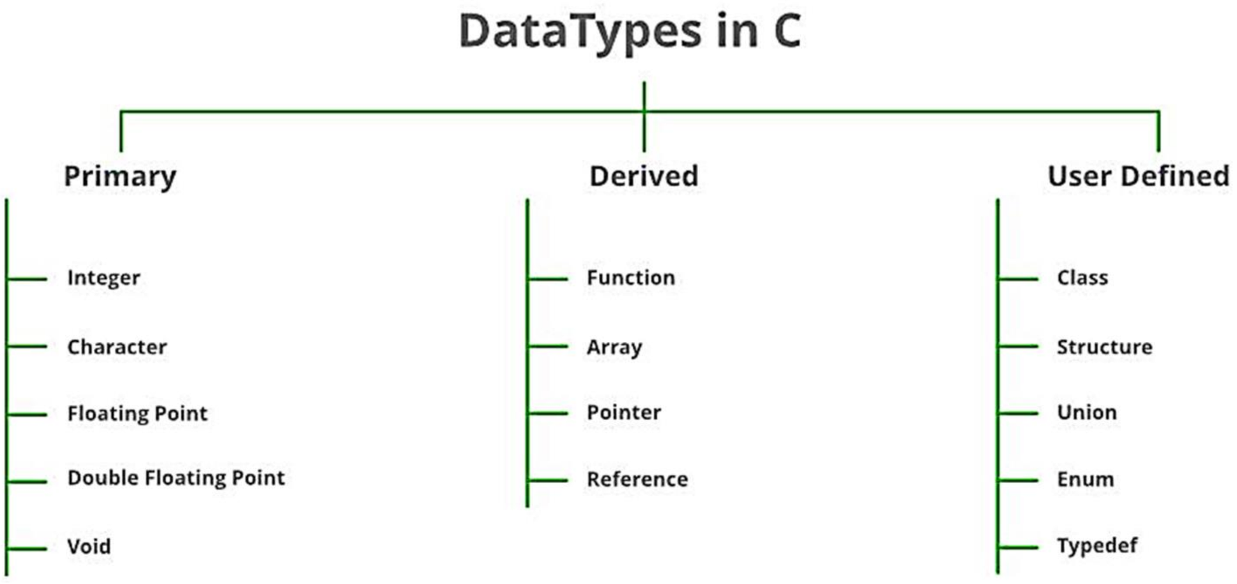
- Integer Constants
    - Refers to sequence of digits.
    - Some of the examples are 112, 0551, 56579u, 0X2 etc.
- Real Constants
    - The floating-point constants such as 0.0083, -0.78, +67.89 etc.
- Single Character Constants
    - A single char const contains a single character enclosed within pair of single quotes [ ‘ ’ ]
    - For example, ‘8’, ‘a’ , ‘i’ etc.
- String Constants
    - A string constant is a sequence of characters enclosed in double quotes [ “ ” ]
    - For example, “0211”, “Stack Overflow” etc.

## Variables

- A Variable is a data name that is used to store any data value.
- Variables are used to store values that can be changed during the program execution.
- Rules for representing Variables –
    - May only consist of letters, digits and underscores.
    - May be as long as you like, but only the first 31 characters are significant.
    - May not begin with a number.
    - May not be a C reserved word (keyword).
    - Should start with a letter or an underscore(_).
    - White spaces are not allowed.
    - C is a case sensitive language.
    - It matters whether an identifier, such as a variable name, is uppercase or lowercase.
    - Example: area, Area, AREA, ArEa are all seen as different variables by the compiler.

- Some examples of valid variable –
    - Sum, value, num1, total_marks.
- Some examples of invalid variables –
    - 123, 1rollno, (area)

**Data Types**



| Data Type | Size | Description |
|---|---|---|
| int | 2 or 4 bytes | Stores whole numbers, without decimals |
| float | 4 bytes | Stores fractional numbers, containing one or more decimals. Sufficient for storing 6-7 decimal digits |
| double | 8 bytes | Stores fractional numbers, containing one or more decimals. Sufficient for storing 15 decimal digits |
| char | 1 byte | Stores a single character/letter/number, or ASCII values |

| Type | Size (bits) | Size (bytes) | Range |
|---|---|---|---|
| char | 8 | 1 | -128 to 127 |
| unsigned char | 8 | 1 | 0 to 255 |
| int | 16 | 2 | $-2^{15}$ to $2^{15}-1$ |
| unsigned int | 16 | 2 | 0 to $2^{16}-1$ |
| short int | 8 | 1 | -128 to 127 |
| unsigned short int | 8 | 1 | 0 to 255 |
| long int | 32 | 4 | $-2^{31}$ to $2^{31}-1$ |
| unsigned long int | 32 | 4 | 0 to $2^{32}-1$ |
| float | 32 | 4 | 3.4E-38 to 3.4E+38 |
| double | 64 | 8 | 1.7E-308 to 1.7E+308 |
| long double | 80 | 10 | 3.4E-4932 to 1.1E+4932 |

**Operators**

- Operators are used to perform operations on variables and values.
- The symbols which are used to perform logical and mathematical operations are called operators.
- E.g.: A + B * 5
- where, +, * are operators,
  - A, B are variables, 5 is constant,
  - A + B * 5 is an expression.

**Types of C Operators**

- C divides the operators into the following groups:
  - Arithmetic operators
  - Assignment operators
  - Comparison operators
  - Logical operators
  - Bitwise operators
- **Arithmetic Operators**
  - Used to perform common mathematical operations.

| Operator | Name | Description | Example |
|---|---|---|---|
| + | Addition | Adds together two values | x + y |
| - | Subtraction | Subtracts one value from another | x - y |

| Operator | Name | Description | Example |
|---|---|---|---|
| * | Multiplication | Multiplies two values | x * y |
| / | Division | Divides one value by another | x / y |
| % | Modulus | Returns the division remainder | x % y |
| ++ | Increment | Increases the value of a variable by 1 | ++x |
| -- | Decrement | Decreases the value of a variable by 1 | --x |

- **Assignment Operators**
  - Assignment operators are used to assign values to variables.

| Operator | Example | Same As |
|---|---|---|
| = | x = 5 | x = 5 |
| += | x += 3 | x = x + 3 |
| -= | x -= 3 | x = x - 3 |
| *= | x *= 3 | x = x * 3 |
| /= | x /= 3 | x = x / 3 |
| %= | x %= 3 | x = x % 3 |
| &= | x &= 3 | x = x & 3 |
| \|= | x \|= 3 | x = x \| 3 |
| ^= | x ^= 3 | x = x ^ 3 |
| >>= | x >>= 3 | x = x >> 3 |
| <<= | x <<= 3 | x = x << 3 |

- **Comparison Operators**
  - Comparison operators are used to compare two values or variables.
  - This is important in programming, because it helps us to find answers and make decisions.
  - The return value of a comparison is either 1 or 0, which means true (1) or false (0).
  - These values are known as Boolean values.

| Operator | Name | Example |
|---|---|---|
| == | Equal to | x == y |
| != | Not equal | x != y |
| > | Greater than | x > y |
| < | Less than | x < y |
| >= | Greater than or equal to | x >= y |
| <= | Less than or equal to | x <= y |

- **Logical Operators**
  - You can also test for true or false values with logical operators.
  - Logical operators are used to determine the logic between variables or values.

| Operator | Name | Description | Example |
|---|---|---|---|
| && | Logical and | Returns true if both statements are true | x < 5 && x < 10 |
| \|\| | Logical or | Returns true if one of the statements is true | x < 5 \|\| x < 4 |
| ! | Logical not | Reverse the result, returns false if the result is true | !(x < 5 && x < 10) |

- **Sizeof Operator**
  - The memory size (in bytes) of a data type or a variable can be found with the sizeof operator

## Managing Input and Output operations

Input and Output

- A program without any input or output has no meaning.

Input → Process → Output

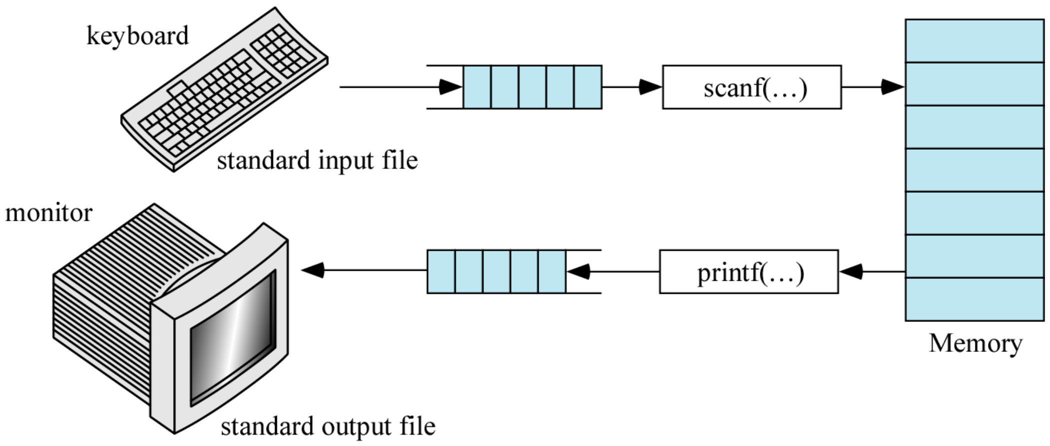
- Eg. marks sheet of students

Input –

- Provide the program with some data to be used in program.

Output –

- Display data on screen as result or information.



Input functions are used to read data from keyboard are called standard input functions.

- scanf(), getchar(), getche(), getch() etc.

Output functions are used to display the result on the screen are called standard output functions.

- printf(), putchar(), putch(), puts() etc.

In C, the standard library stdio.h provides functions for input and output.

The instruction #include<stdio.h> tells the compiler to search for a file named stdio.h and places its contents at this point in the program.

The input/output functions are classified into two types –

1. Formatted Input/Output functions
2. Unformatted Input/Output functions

## Formatted Input/Output functions

- Formatted console input/output functions are used to take a single or multiple inputs from the user at console and it also allows us to display one or multiple values in the output to the user at the console.
- Format specifiers allow us to read or display any value of a specific data type.
- Input function: scanf()
- Output function: printf()

| Data Type | Format | Meaning |
|---|---|---|
| int | %d | Represents a decimal integer value. |
| | %u | Represents an unsigned integer value. |
| | %o | Represents an unsigned octal value. |
| | %x | Represents an unsigned hexadecimal value. |
| float | %f | Represents a floating point value. |
| | %e | Represents a floating point value in decimal or exponential form. |
| char | %c | Represents a single character value. |
| | %s | Represents a string of value of characters. |

## Formatted Input

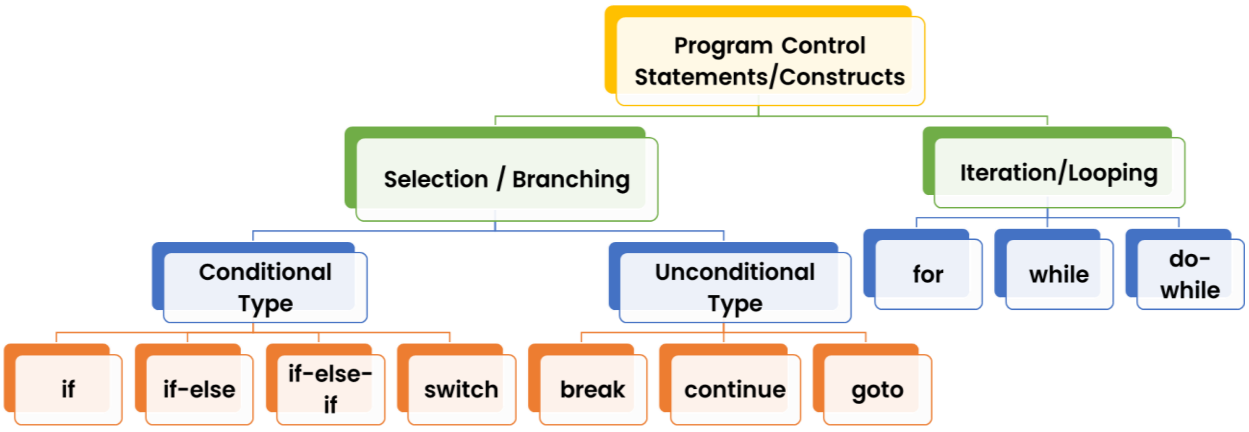
- Consider the following data:
  - 50 → Int,
  - 13.45 → float,
  - Ram → char variables.
- The built-in function scanf() can be used to enter input data into the computer from a standard input device.
  - This function is used to read one or multiple inputs from the user.
- Example: scanf(“ %d", &n1);
- The general form of scanf is,
  - scanf(“control string” , arg1, arg1,….. argn);
- Control string: format in which data is to be entered.
- arg1,arg2… :	location where the data is stored.
  preceded by ampersand (&)

## Decision Making & Branching Statements

C provides various key condition statements to check condition and execute statements according conditional criteria.

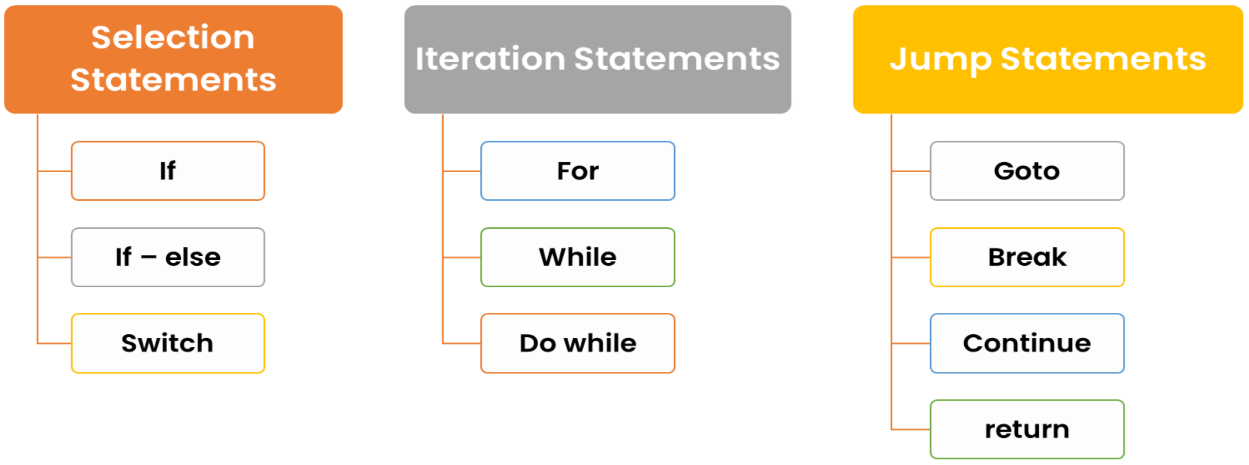
These statements are called as 'Decision Making Statements' or 'Conditional Statements'.

## Program Control Statements



## Control Statements

Control flow statements are blocks of code that control the flow of a program.



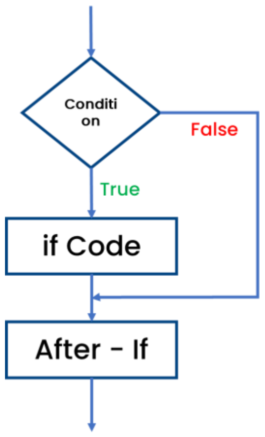
## Decision Making Statements

- Types of Decision-Making Statements –
  - If Statement
  - Simple If
  - If Else Statement
  - Nested If Statement
  - Else If Ladder
- Switch Case Statements
- Conditional Operator

## Simple if statement –

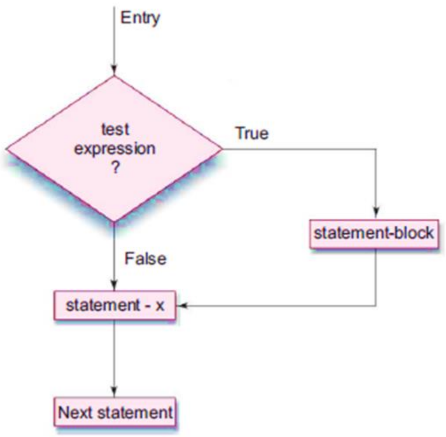
- Takes an expression in parenthesis.
- If the expression is true then true-block statements get executed otherwise these statements are skipped.
- Takes an expression in parenthesis.
- If the expression is true then true-block statements gets executed otherwise these statements are skipped.
- Syntax:

```
if(condition)
{
        statements;
}
statement-x;
```



- Example –

```
void main()
{
int a=5;
        if(a%2==0)
        {
        printf("number %d is even", a);
        }
}
```



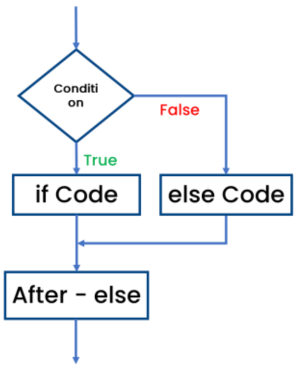
- Advantages of if Statement
  - It is the simplest decision-making statement.
  - It is easy to use and understand.
  - It can evaluate expressions of all types such as int, char, bool, etc.

- Disadvantages of if Statement
  - It contains only a single block. In case when there are multiply related if blocks, all the blocks will be tested even when the matching if block is found at the start
  - When there are a large number of expressions, the code of the if block gets complex and unreadable.
  - It is slower for a large number of conditions.

## if else Statements –

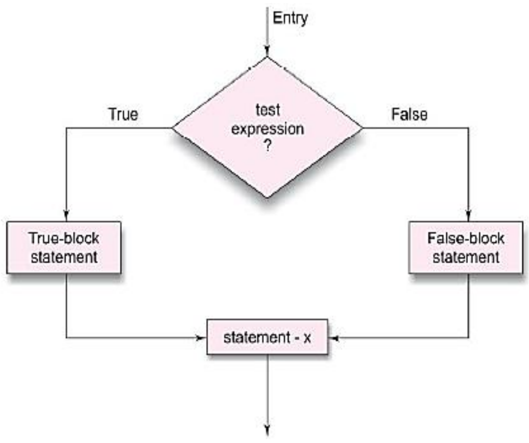
- Takes an expression in parenthesis.
- If test expression is true, then true-block statements get executed; otherwise, the false-statements gets executed.
- Syntax:

```
        if(condition)
            {
            true statements;
            }
        else
            {
            false statements;
            }
```



- Example –

```
void main()
{
int a=5;
    if(a%2==0)
    {
       printf("number %d is even", a);
    }
    else
    {
       printf("number %d is odd",a);
    }
}
```



- Advantages of if-else Statement
    - The if-else statement enables the user to execute different statements based on different conditions.
    - It can evaluate test expressions of type int, char, boolean, and more.
    - It helps in modifying the flow of the program.
    - It is simple, efficient, and easier to read when there is less number of conditions.
- Disadvantages of if-else Statement
    - If there are a lot of if statements present, the code becomes unreadable and complex.
    - It also becomes slower compared to the switch statement.

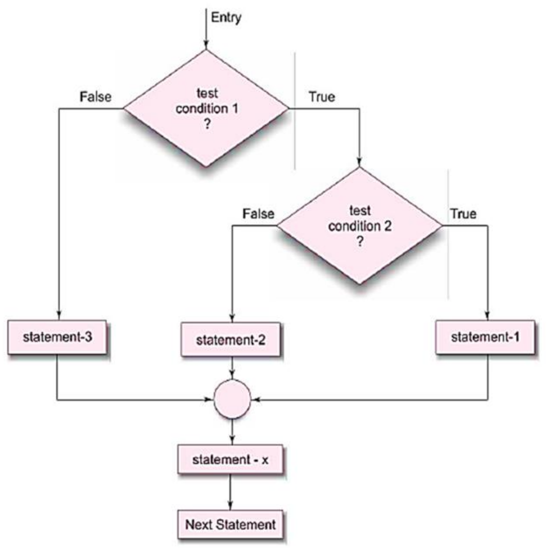
## nested if...else statements –

- If within an If condition is called Nested If condition.
- It is a conditional statement which is used when we want to check more than one conditions at a time in a same program.

- Syntax -

```
if(condition-1)

{
        if(condition-2)
        {
        true statement-1;
        }
        else
        {
        false statement-2;
        }
}
else
{
        statements-3;
}
statement-x;
```



- Example –

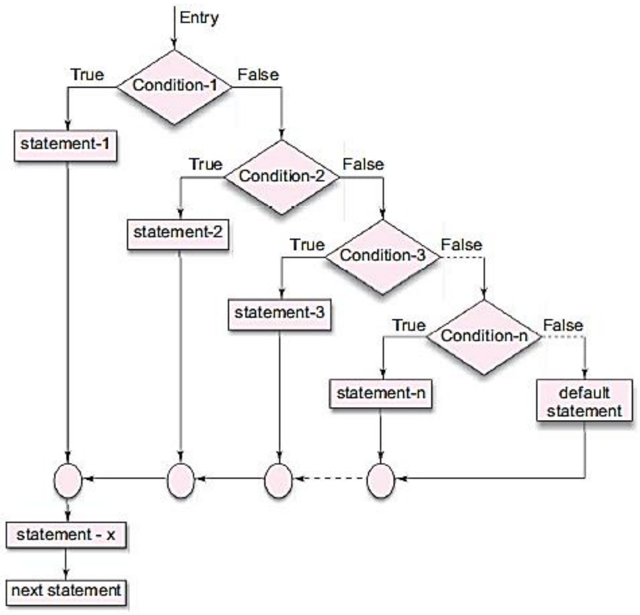
```
void main()
{
        int a, b, c;
        a=5, b=6, c=8;
        if(a>b)
        {
                if(a>c)
                {
                printf("a is big");
                }
                else
                {
                printf("c is big");
                }
        }

        else
        {
                if(b>c)
                {
                printf("b is big");
                }
                else
                {
                printf("c is big");
                }
        }
        getch();
}
```

else…if Ladder

- Else if() ladder is similar to if else control statement.
- Else if() ladder is used to check for another condition if the previously specified condition becomes false.
- It is used when multipath decisions are involved.
- Syntax –

```
if (condition-1)
{
statement-1;
}
else if(condition-2)
{
statement-2
}
else if (condition-n)
{
statement-n;
}
else
{
        default - statements;
}
statement-x;
```



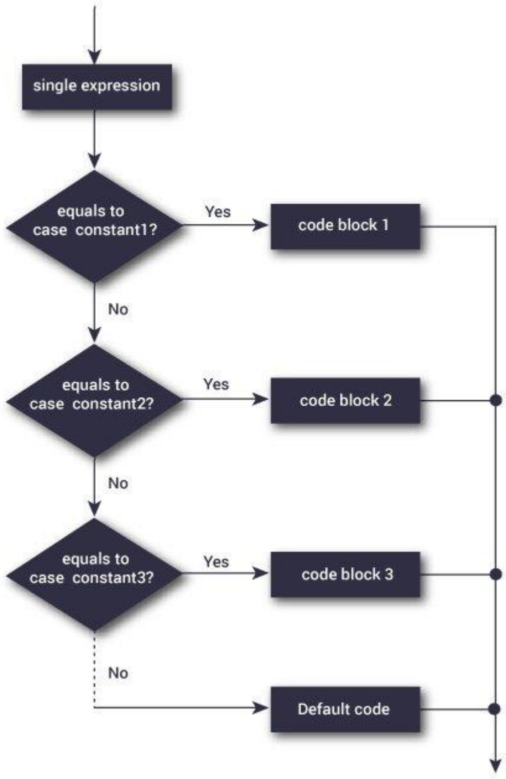
- Example -

```
void main()
{
int a,b,c;
a=5, b=6, c=7;
        if(a>b && a>c)
        {
        printf("a is big");
        }
        else if(b>c)
        {
        printf("b is big");
        }
        else
        {
        printf("c is big");
        }
getch();
}
```

**Switch Case Statements**

- This is a multiple or multiway branching decision making statement.
- Switch case checks the value of a expression against a case values, if condition matches the case values then the control is transferred to that point.
- Syntax:

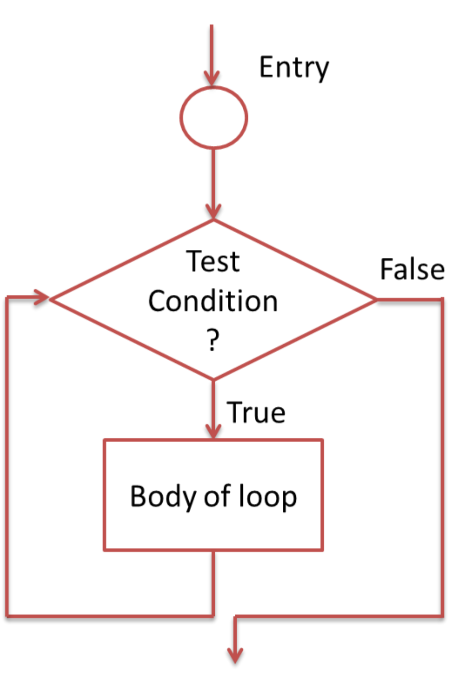
```
switch(expression)
        {
                case value1:
                        block-1 statement;
                        break;
                case value2:
                        block-2 statement;
                        break;
                default:
                        default block statement;
                        break;
        }
        statement-x;
```



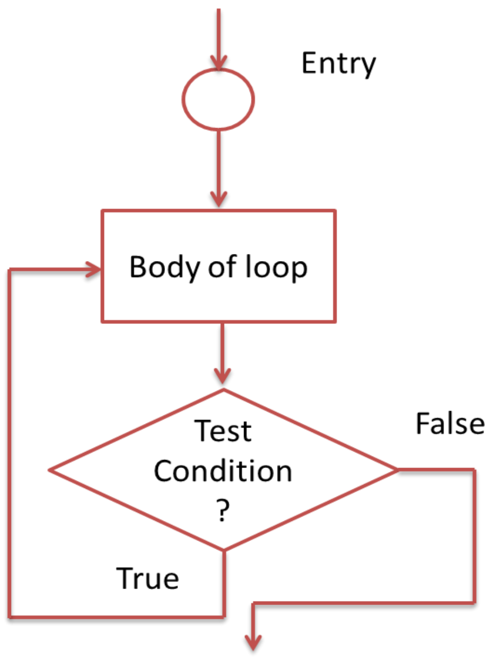
- Rules
    - Switch expression must be integral type.
    - Labels must be constants. Labels must be unique.
    - No two labels can have the same value.
    - Case labels must be end with colon.
    - Break statement transfer the control out of the switch statement.
    - Break statement is optional. Default labels optional.
    - Default may be placed anywhere but usually placed at the end.

## Decision Making Looping Statements

- Depending upon the position of a control statement in a program, looping statement in C is classified into two types:
  - Entry controlled loop
  - Exit controlled loop
- In an entry control loop in C, a condition is checked before executing the body of a loop.
  - It is also called as a pre-checking loop.
- In an exit-controlled loop, a condition is checked after executing the body of a loop.
  - It is also called as a post-checking loop.



Entry Controlled Loop



Exit Controlled Loop

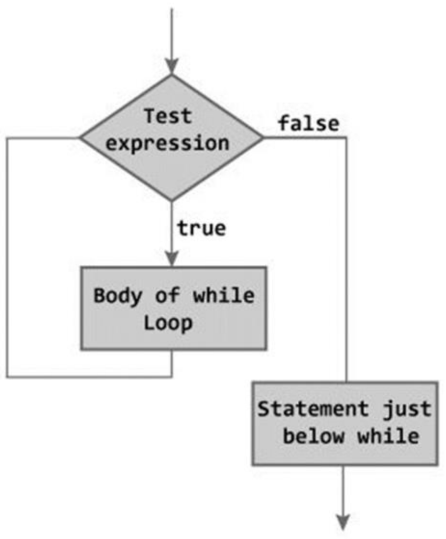
A looping process -

- Initialization of conditional variable
- Loop condition is tested before iteration to decide whether to continue or terminate the loop.
- Execution of statements in loop.
- Incrementing or updating the conditional variable at the end of each loop iteration.
- Types of loops:
  - While loop
  - Do…while loop
  - For loop

**While Loop**

- It has a loop condition only that is tested before each iteration to decide whether to continue or terminate the loop.
- If the test condition is true: the statements get executed until the condition becomes false.
- Syntax:

```
while(test condition)
{
        body of the loop;
}
```



- Example: Display “Hello DYP” 10 times.

```
        int main()
        {
                int i = 0;
                while (i<10)
                {
                        printf(“Hello DYP”);
                        i = i + 1;
                }
        }
```

- Example: Display 10 natural numbers.

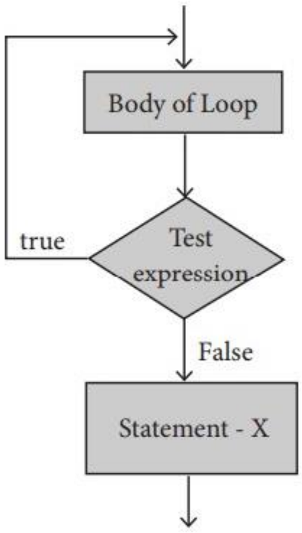
```
        {
                        int i = 1;
                        while (i<=10)
                        {
                                printf(“Natural numbers are: \n”);
                                printf(“%d \t”, i);
                                i = i + 1;
                        }
                }
```

**Do…while Loop**

- Do while has a loop condition only that is tested after each iteration to decide whether to continue with next iteration or terminate the loop.
- The statements get executed once.
- Then the condition is evaluated.
- Syntax:

```
do
{
        body of the loop;

} while(test condition);
```



- When the program control first comes to the do…while loop, the body of the loop is executed first and then the test condition/expression is checked, unlike other loops where the test condition is checked first.
- Due to this property, the do…while loop is also called exit controlled or post-tested loop.
- When the test condition is evaluated as true, the program control goes to the start of the loop and the body is executed once more.
- The above process repeats till the test condition is true.
- When the test condition is evaluated as false, the program controls move on to the next statements after the do…while loop.

- Example: Display “Hello DYPians” 10 times.

```
int main()
        {
        int i = 0;
                do
                {
                        printf("Hello DYPians");
                        i = i + 1;
                } while (i<10);
        }
```

- Example: Display 10 natural numbers.

```
int main()
        {
                int i = 1;
                do
                {
                        printf("Natural numbers are: \n);
                        printf("%d \t", i);
                        i = i + 1;
                } while (i<=10);
        }
```

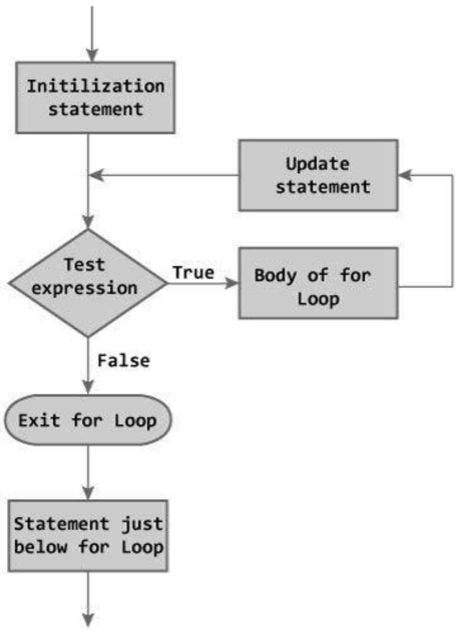
**Difference between while and do...while Loop in C**

| **while Loop** | **do...while Loop** |
|---|---|
| • The test condition is checked **before the loop body is executed.** | • The test condition is checked **after executing the body.** |
| • When the condition is false, the **body is not executed** not even once. | • The body of the **do...while loop is executed at least once** even when the condition is false. |
| • It is a type of **pre-tested or entry-controlled loop.** | • It is a type of **post-tested or exit-controlled loop.** |
| • Semicolon is not required. | • Semicolon is required at the end. |

**for Loop**

- for loop has three parts:
  - Initializer is executed at start of loop.
  - Loop condition is tested before iteration to decide whether to continue or terminate the loop.
  - Increment is executed at the end of each loop iteration.
- Syntax:

```
        for (initialization; test condition; increment)
        {
                body of loop;
        }
```



- Example: Display "Hello DYPians" 10 times.

```
        int main()
                {
                int i;
                for(i=0; i<10; i++)
                        {
                        printf("Hello DYPians");
                        }
                }
```

- Example: Display 10 natural numbers.

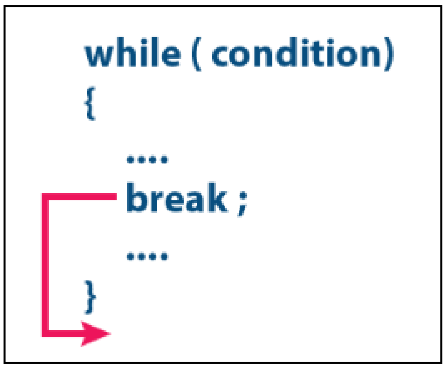
```
                int main()
                        {       int i ;
                                printf("Natural numbers are: \n);
                                for(i=1; i<=10; i++)
                                {
                                        printf("%d \t", i);
                                }
                        }
```

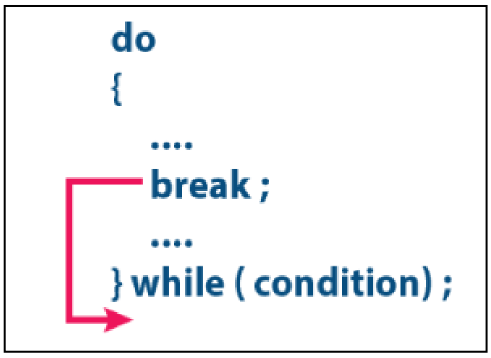
**Unconditional Control Statements**

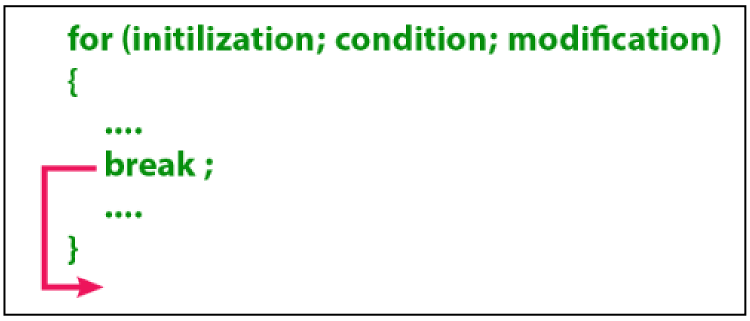
- There are control statements that do not need any condition to control the program execution flow.
- C programming language provides the following unconditional control statements...
    - break
    - continue
    - goto
    - return

**Break Statement**

- A break statement terminates the execution of the loop and the control is transferred to the statement immediately following the loop.
    - i.e., the break statement is used to terminate loops or to exit from a switch.
- It can be used within for, while, do-while, or switch statement.
- The break statement is written simply as break;





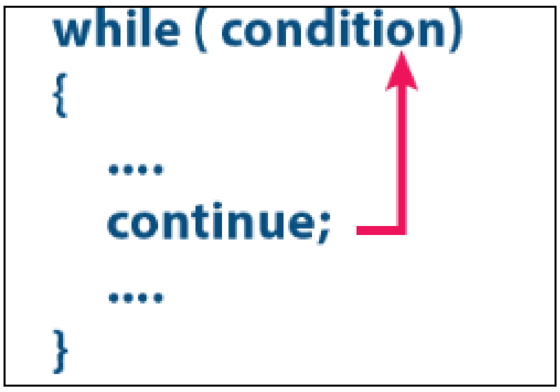


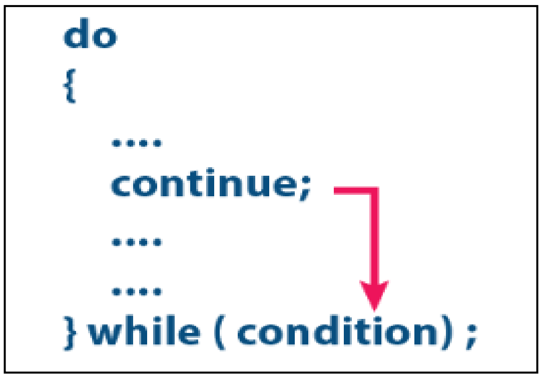
- Example:

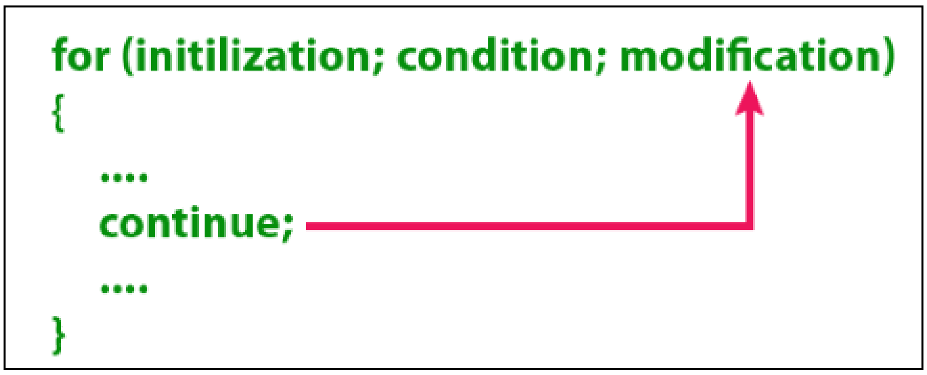
```
#include <stdio.h>
int main()
{
        int i;
        for(i=1; i<=10; i++)
                {
                printf("%d ", i);
                if(i==5) //if value of i is equal to 5, it will break the loop
                        {
                                break;
                        }
                }//end of for loop
        return 0;
}
```

**Continue Statement**

- The continue statement is used to move the program execution control to the beginning of the looping statement.
- When the continue statement is encountered in a looping statement, the execution control skips the rest of the statements in the looping block and directly jumps to the beginning of the loop.







- Example:

```
# include<stdio.h>
int main( )
 {
   int  i=1, num, sum =0;
   for(i=0; i < 5; i ++)
    {
     printf(“ Enter an integer:”);
     scanf( “%d”, &num);
     if(num < 0)
    {
       printf(“\n you have entered a negative number”);
       continue ;     /* skip the remaining part of loop */
    }
   sum += num;
 }
printf(“The sum of the Positive Integers Entered = % d \ n”, sum);
}
```

**goto Statement**

- The goto statement is used to jump from one line to another line in the program.
- Using goto statement we can jump from top to bottom or bottom to top.
- The goto requires a label in order to identify the place where the branch is to be made.
- Example:

```
#include<stdio.h>
int main( )
{
   printf("Hello");
   goto l1;
   printf("How are");
   l1: printf("you");
}
```

**return Statement**

- The return statement terminates the execution of a function and returns control to the calling function.
- A return statement can also return a value to the calling function.
- Syntax:

```
return expression;
```

- Example:

```
#include<stdio.h>

int main()

{

   int a,b,c;

   printf("enter a and b value:");

   scanf("%d%d",&a,&b);

   c=a*b;

   return(c);

}
```