

# *C Programming*

*by Manjiri Deshpande*

## *Day1*



Dr. D. Y. Patil Pratishthan's  
**Institute for Advanced Computing and Software Development**

# Classification of generations of computers

- The evolution of computer technology is often divided into five generations.

| Generations of computers | Generations timeline       | Evolving hardware             |
|--------------------------|----------------------------|-------------------------------|
| First generation         | 1940s-1950s                | Vacuum tube based             |
| Second generation        | 1950s-1960s                | Transistor based              |
| Third generation         | 1960s-1970s                | Integrated circuit based      |
| Fourth generation        | 1970s-present              | Microprocessor based          |
| Fifth generation         | The present and the future | Artificial intelligence based |

# **Classification of Computers by Size**

- Supercomputers**
- Mainframe computers**
- Minicomputers**
- Personal computers (PCs) or microcomputers**

## *Define Computer Program*

- Program is a set of instructions
- Given in a particular sequence
- Having a predefined meaning



- An instruction is a combination of some words. Which have a predefined meaning.
- Instructions are given in a particular sequence.

# What is Programming Language

Are Classified into two type:

1)Low Level Language:-

a)Machine Level Language:-combination of 0 zeros and 1

b)Assembly Language:-some symbols called *mnemonic*

-----

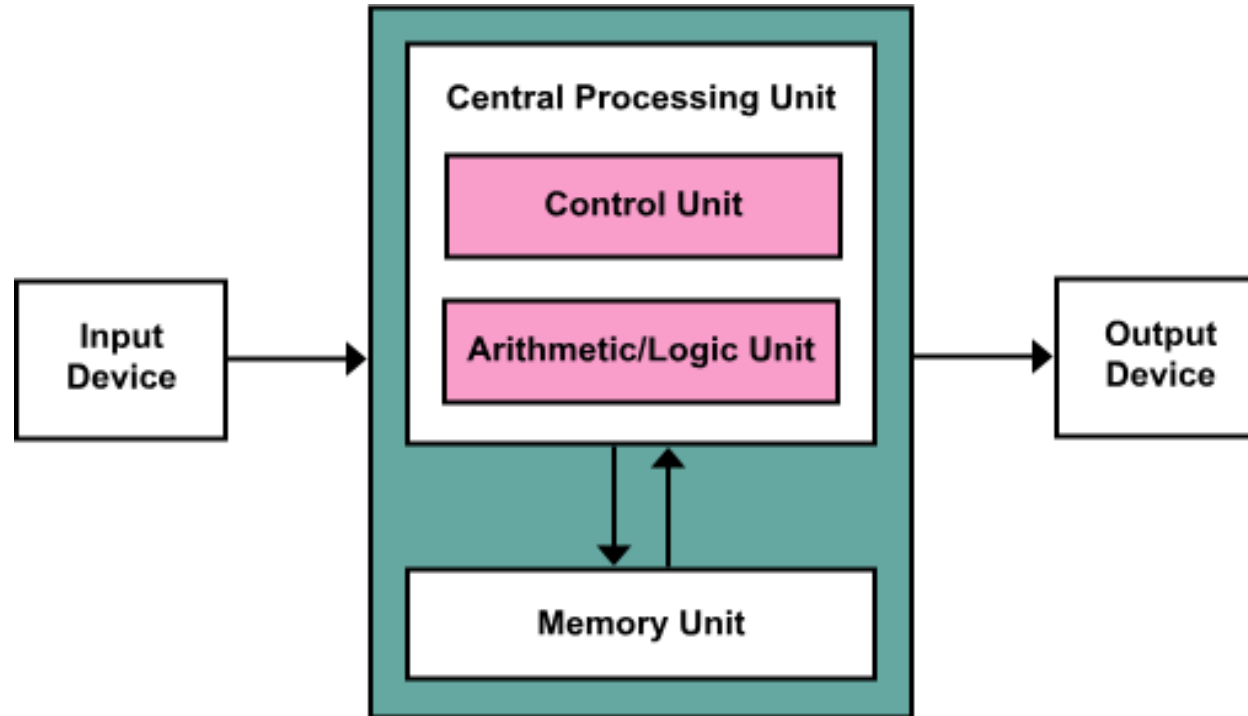
2)High Level Language:-

# *Translator:-*

- **Assembler:-**Used by assembly language.
- **Compiler:-**used by C ,C++(translation, linking ,loading),Covert complete program & run (object code is created)
- **Interpreter:-**line by line conversion only translation(not object code )

# *Programming Instructions*

A Computer has two inseparable parts:-  
Hardware and Software

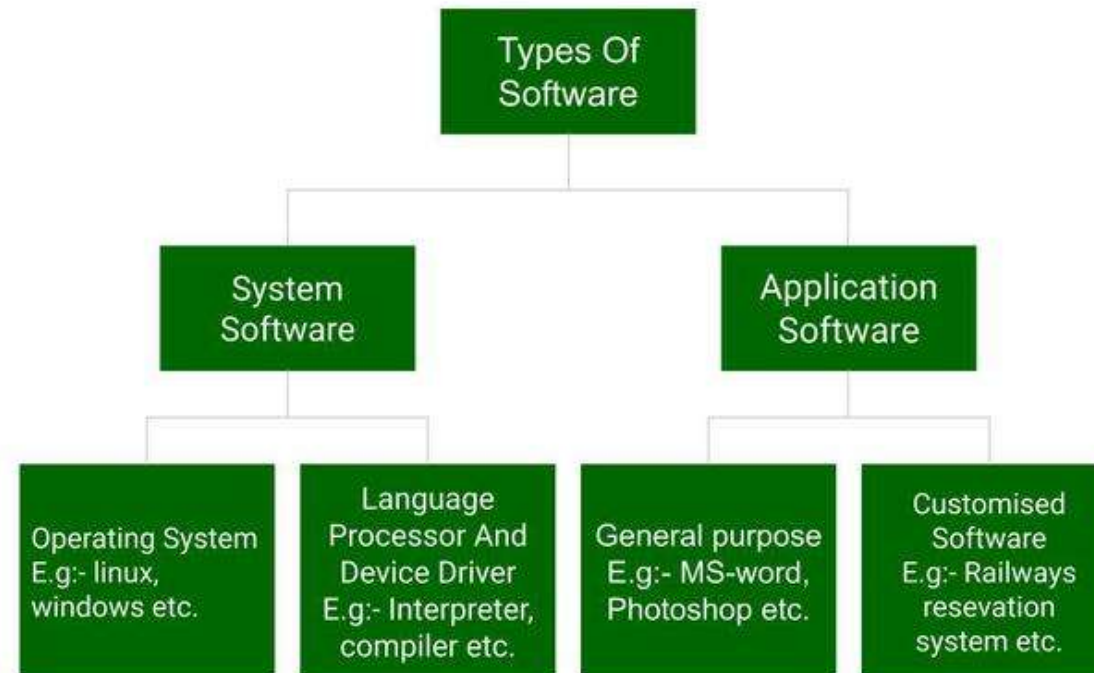


## *What is Software & Hardware?*

- **The instructions are the software**
- **The physical component of a computer that are used in the process are hardware.**
- **Application S/W:-** software that fulfills a specific need or performs tasks
- **System S/W:-** is designed to run a computer's hardware and provides a platform for applications to run on top of.



# Types of Software



## *Why Number System:-*

- $C=a+b$
- Display c

a,b and c are  
numbers

Input as well as output have to be understood by the computer and the user respectively

A number system plays an important role so that the same instructions can be manipulated by giving different numbers as input.

## *Number System:-*

- Way of counting things
- Computer use different types of number systems.
- 1.Decimal(base 10)
- 2.Binary(base 2) 0 1
- 3.Octal(base 8)
- 4.Hexadecimal(base 16)

## *Binary Number System:-*

- Most modern computer systems operate using binary number system.
- The binary number system works like the decimal number system except that it uses 2 as its base and has only two digits 0 and 1
- The weighted values for each position is determined as follows:

|       |       |       |       |       |       |       |       |          |          |
|-------|-------|-------|-------|-------|-------|-------|-------|----------|----------|
| $2^7$ | $2^6$ | $2^5$ | $2^4$ | $2^3$ | $2^2$ | $2^1$ | $2^0$ | $2^{-1}$ | $2^{-2}$ |
| 128   | 64    | 32    | 16    | 8     | 4     | 2     | 1     | .5       | .25      |






## *Algorithm:*

- ----The word Algorithm means “a process or set of rules to be
- followed in calculations or other problem-solving operations”.
- -----Therefore Algorithm refers to a set of rules/instructions
- that step-by-step define how a work is to be executed upon
- In order to get the expected results.

## *Flowchart:-*

- A flowchart is the graphical or pictorial representation of an algorithm with the help of different symbols, shapes, and arrows to demonstrate a process or a program. With algorithms, we can easily understand a program. The main purpose of using a flowchart is to analyze different methods. Several standard symbols are applied in a flowchart:
- The symbols above represent different parts of a flowchart. The process in a flowchart can be expressed through boxes and arrows with different sizes and colors. In a flowchart, we can easily highlight certain elements and the relationships between each part.

# ***Flowchart & Symbols:-***

|                            |   |
|----------------------------|---|
| Terminal Box – Start / End |    |
| Input / Output             |    |
| Process / Instruction      |    |
| Decision                   |   |
| Connector / Arrow          |  |

# ***Difference between Algorithm and Flowchart***

| Algorithm  | Flowchart   |
|--|---|
| Algorithm is the step-by-step instruction to solve a specific problem. | Flowchart is a pictorial representation to show the algorithm using geometrical diagrams and symbols. |
| Difficult to understand compared to flowcharts                         | Easier to understand  |
| Complex representation of branching and looping                        | Easy representation of branching and looping  |
| Easy debugging of errors   | Difficult debugging of errors   |
| Does not follow any rules to write                                     | Has certain predefined rules of construction  |



# *Why not English why C?*

- Natural Language are ambiguous by nature.
  - Bank-a financial institution or an edge of a river.
  - Good-can mean useful or pleasing
- Programming language are distinct and clear
- Each word in a programming language has one & only one meaning.  
No two words mean the same.

## *History Of C*

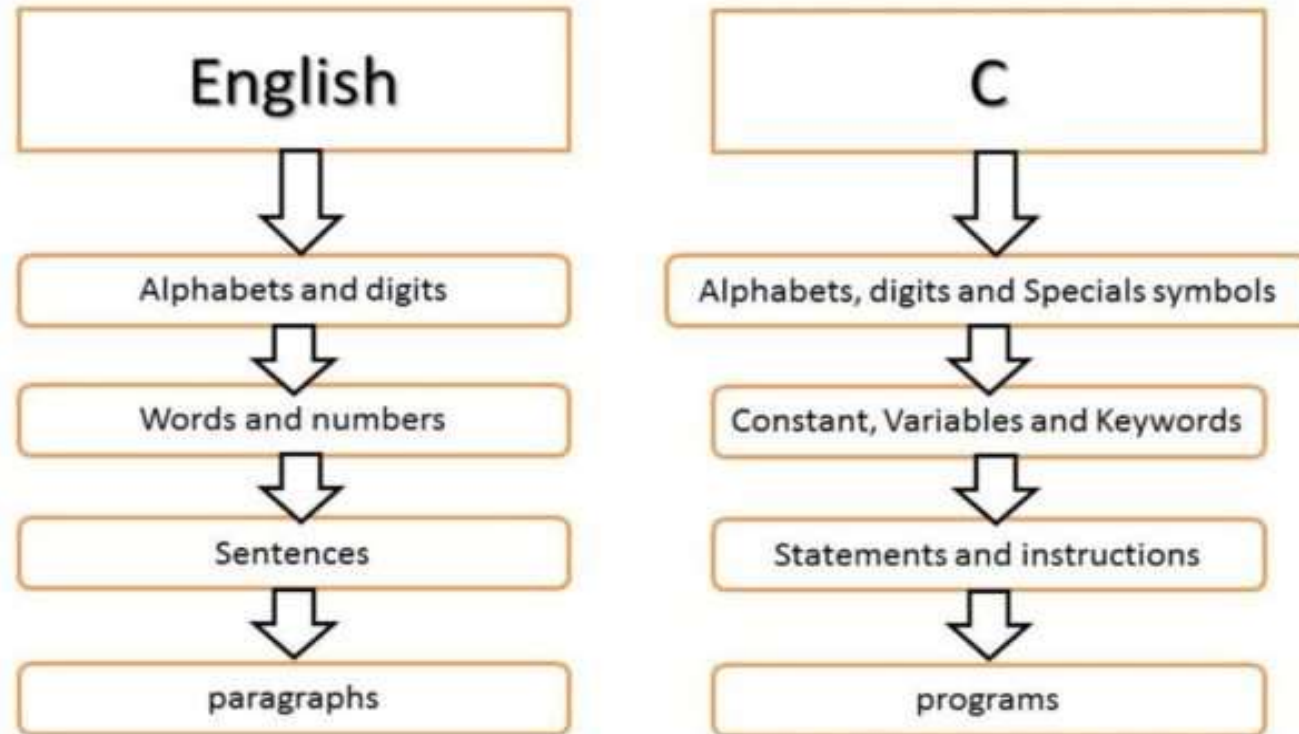


- **C programming language** was developed in 1972 by Dennis Ritchie at bell laboratories of AT&T (American Telephone & Telegraph), located in the U.S.A.
- **Dennis Ritchie** is known as the **founder of the c language**.
- It was developed to overcome the problems of previous languages such as B, BCPL, etc.
- Initially, C language was developed to be used in **UNIX operating system**. It inherits many features of previous languages such as B and BCPL

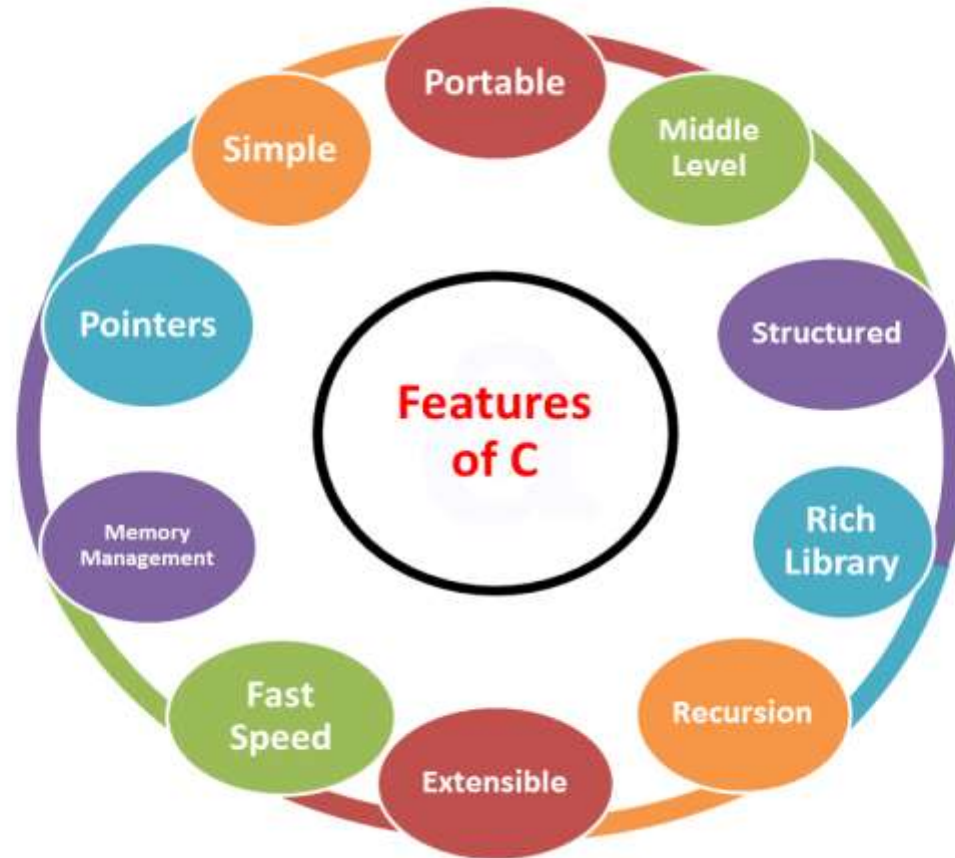
## *Before C Language:-*

| Language      | Year | Developed By               |
|---------------|------|----------------------------|
| Algol         | 1960 | International Group        |
| BCPL          | 1967 | Martin Richard             |
| B             | 1970 | Ken Thompson               |
| Traditional C | 1972 | Dennis Ritchie             |
| K & R C       | 1978 | Kernighan & Dennis Ritchie |
| ANSI C        | 1989 | ANSI Committee             |
| ANSI/ISO C    | 1990 | ISO Committee              |
| C99           | 1999 | Standardization Committee  |

## *English Vs C*



## *Features Of C:-*



## *Simple C Program:-*

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

## *Simple C Program:-*

- **Line 1: [ `#include <stdio.h>` ]** In a C program, all lines that start with `#` are processed by a [preprocessor](#) which is a program invoked by the compiler. In a very basic term, the [preprocessor](#) takes a C program and produces another C program. The produced program has no lines starting with `#`, all such lines are processed by the preprocessor. In the above example, the preprocessor copies the preprocessed code of `stdio.h` to our file. The `.h` files are called header files in C. These header files generally contain declarations of functions. We need `stdio.h` for the function `printf()` used in the program.

# *Simple C Program:-*

- ***Line 2:*** `[int main() ]`
- `main()` is called more or less at the beginning of the program's execution, and when `main()` ends, the runtime system shuts down the program. `main()` always returns an `int`, as shown below:
- `main()` has a special feature: There's an implicit return 0; at the end.
- Thus if the flow of control simply falls off the end of `main()`, the value 0 is implicitly returned to the operating system. Most operating systems interpret a return value of 0 to mean "program completed successfully."



# *Simple C Program:-*

- **Line 3 and 6: [ { and } ]** In C language, a pair of curly brackets define scope and are mainly used in functions and control statements like if, else, loops. All functions must start and end with curly brackets.
- **Line 4 :-** [printf\(\)](#) is a standard library function to print something on standard output. The semicolon at the end of printf indicates line termination. In C, a semicolon is always used to indicate end of a statement
- **Line 5 [ return 0; ]** The return statement returns the value from main(). 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. .

# C Tokens

- Keyword
- Identifiers
- Variables
- Data Types
- Character set
- Constant
- Operators

# Character Set In C:-

## Character Set

- The character set of C represents alphabet, digit or any symbol used to represent information.

| Types               | Character Set  |
|---------------------|--|
| Uppercase Alphabets | A, B, C, ... Y, Z  |
| Lowercase Alphabets | a, b, c, ... y, z  |
| Digits              | 0, 1, 2, 3, ... 9  |
| Special Symbols     | ~ ' ! @ # % ^ & * ( ) _ - + =   \ { } [ ]<br>: ; " ' < > , . ? / |
| White spaces        | Single space, tab, new line.                                     |

## *Keywords in C*

- A keyword is a **reserved word**. You cannot use it as a variable name, constant name, etc. There are only 32 reserved words (keywords) in the C language.
- A list of 32 keywords in the c language is given below:

|        |        |          |        |          |          |          |        |
|--------|--------|----------|--------|----------|----------|----------|--------|
| 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  |

## *C Identifiers*

- C identifiers represent the name in the C program, for example, variables, functions, arrays, structures, unions, labels, etc. An identifier can be composed of letters such as uppercase, lowercase letters, underscore, digits, but the starting letter should be either an alphabet or an underscore

## *Rules for constructing C identifiers:-*

- The first character of an identifier should be either an alphabet or an underscore, and then it can be followed by any of the character, digit, or underscore. E.g.num1,\_num1,NUM1,Num1
- It should not begin with any numerical digit.
- In identifiers, both uppercase and lowercase letters are distinct. Therefore, we can say that identifiers are case sensitive.
- Commas or blank spaces cannot be specified within an identifier.
- Keywords cannot be represented as an identifier.
- The length of the identifiers should not be more than 31 characters.
- Identifiers should be written in such a way that it is meaningful, short, and easy to read.

# Variables in C

- A **variable** is a name of the memory location. It is used to store data. Its value can be changed, and it can be reused many times.
- It is a way to represent memory location through symbol so that it can be easily identified.
- Let's see the syntax to declare a variable:

type variable\_list;

10+20

Num1=10

Num2=20

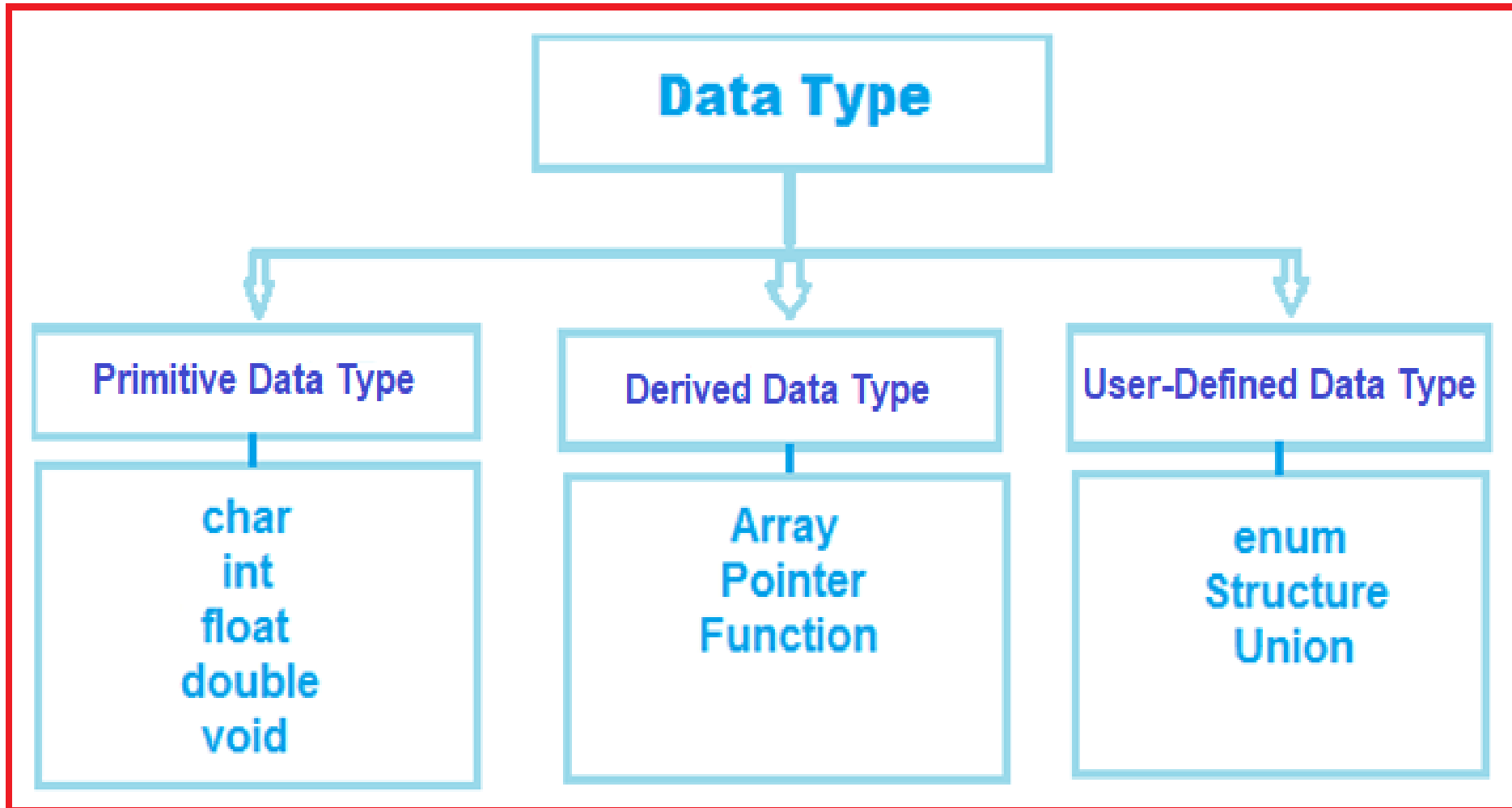
```
1.int a=10,b=20;//declaring 2 variable of integer type  
2.float f=20.8;  
3.char c='A'
```

## *Rules for defining variables*

- Must be declared first use.
- A variable can have alphabets, digits, and underscore.
- A variable name can start with the alphabet, and underscore only. It can't start with a digit.
- No whitespace is allowed within the variable name.
- A variable name must not be any reserved word or keyword, e.g. int, float, etc.
- For local variables ,declaration statements must be placed at the beginning of the block
- For global variables, declaration statements must be placed before first function definition.



# *Data Types in C:-*



# *Data Types in C*

- Each variable in C has an associated data type. Each data type requires different amounts of memory and has some specific operations which can be performed over it. Let us briefly describe them one by one:  
Following are the examples of some very common data types used in C:
- Basic data types(Primitive )
- **char:** The most basic data type in C. It stores a single character and requires a single byte of memory in almost all compilers.
  - `char ch1='a'; %c`
- **int:** As the name suggests, an int variable is used to store an integer.
  - `int num=10; %d`
- **float:** It is used to store decimal numbers (numbers with floating point value) with single precision.
  - `float fnum=10.5; %f`
- **double:** It is used to store decimal numbers (numbers with floating point value) with double precision. `Double dnum=10.657; %lf`

# Format Specifier

- %c—Convert data into character
- %d—Convert data into signed int.
- %f—Convert data into float(6 precisions)
- %s—Converts data into string
- %ld—Converts data into long int
- %u—converts data into unsigned
- %x—converts data into hexadecimal format
- %o—Converts data into octal format
- %e—Converts data into exponential format

# Operators in C

## Operators in C

Unary operator



++, --

Unary operator

Binary operator



+, -, \*, /, %

Arithmetic operator

<, <=, >, >=, ==, !=

Relational operator

&&, ||, !

Logical operator

&, |, <<, >>, ~, ^

Bitwise operator

=, +=, -=, \*=, /=, %=

Assignment operator

Ternary operator



?:

Ternary or  
conditional operator



# *Operators in C*

- **Arithmetic Operators** (+, -, \*, /, %, post-increment, pre-increment, post-decrement, pre-decrement) +
- **Relational Operators** (==, !=, >, <, >= & <=) **Logical Operators** (&&, || and !)
- **Bitwise Operators** (&, |, ^, ~, >> and <<)
- **Assignment Operators** (=, +=, -=, \*=, etc)
- **Other Operators** (conditional, comma, sizeof, address, redirection)

# *Header Files:-*

| Header File | What it does   |
|-------------|--|
| stdio.h     | Mainly used to perform input and output operations like print(), scanf().      |
| string.h    | Mainly used to perform string handling operations like strlen(), strcmp() etc. |
| conio.h     | With this header file, you can execute console input and output operations.    |
| stdlib.h    | Mainly used to perform standard utility functions like malloc(), calloc() etc. |
| math.h      | Mainly used to perform mathematical operations like sqrt(), pow() etc.         |
| ctype.h     | Used to perform character type functions like isdigit(), isalpha() etc.        |

# Comments in C

- Comments in C language are used to provide information about lines of code. It is widely used for documenting code. There are 2 types of comments in the C language.

1. Single Line Comments \

2. Multi-Line Comments \\*-----

## Single Line Comments

- Single line comments are represented by double slash \. Let's see an example of a single line comment in C.

## *C Identifiers*

- C identifiers represent the name in the C program, for example, variables, functions, arrays, structures, unions, labels, etc. An identifier can be composed of letters such as uppercase, lowercase letters, underscore, digits, but the starting letter should be either an alphabet or an underscore



## *Rules for constructing C identifiers:-*

- The first character of an identifier should be either an alphabet or an underscore, and then it can be followed by any of the character, digit, or underscore. E.g. num1, \_num1, NUM1, Num1
- It should not begin with any numerical digit.
- In identifiers, both uppercase and lowercase letters are distinct. Therefore, we can say that identifiers are case sensitive.
- Commas or blank spaces cannot be specified within an identifier.
- Keywords cannot be represented as an identifier.
- The length of the identifiers should not be more than 31 characters.
- Identifiers should be written in such a way that it is meaningful, short, and easy to read.

# Variables in C

- A **variable** is a name of the memory location. It is used to store data. Its value can be changed, and it can be reused many times.
- It is a way to represent memory location through symbol so that it can be easily identified.
- Let's see the syntax to declare a variable:

type variable\_list;

10+20

Num1=10

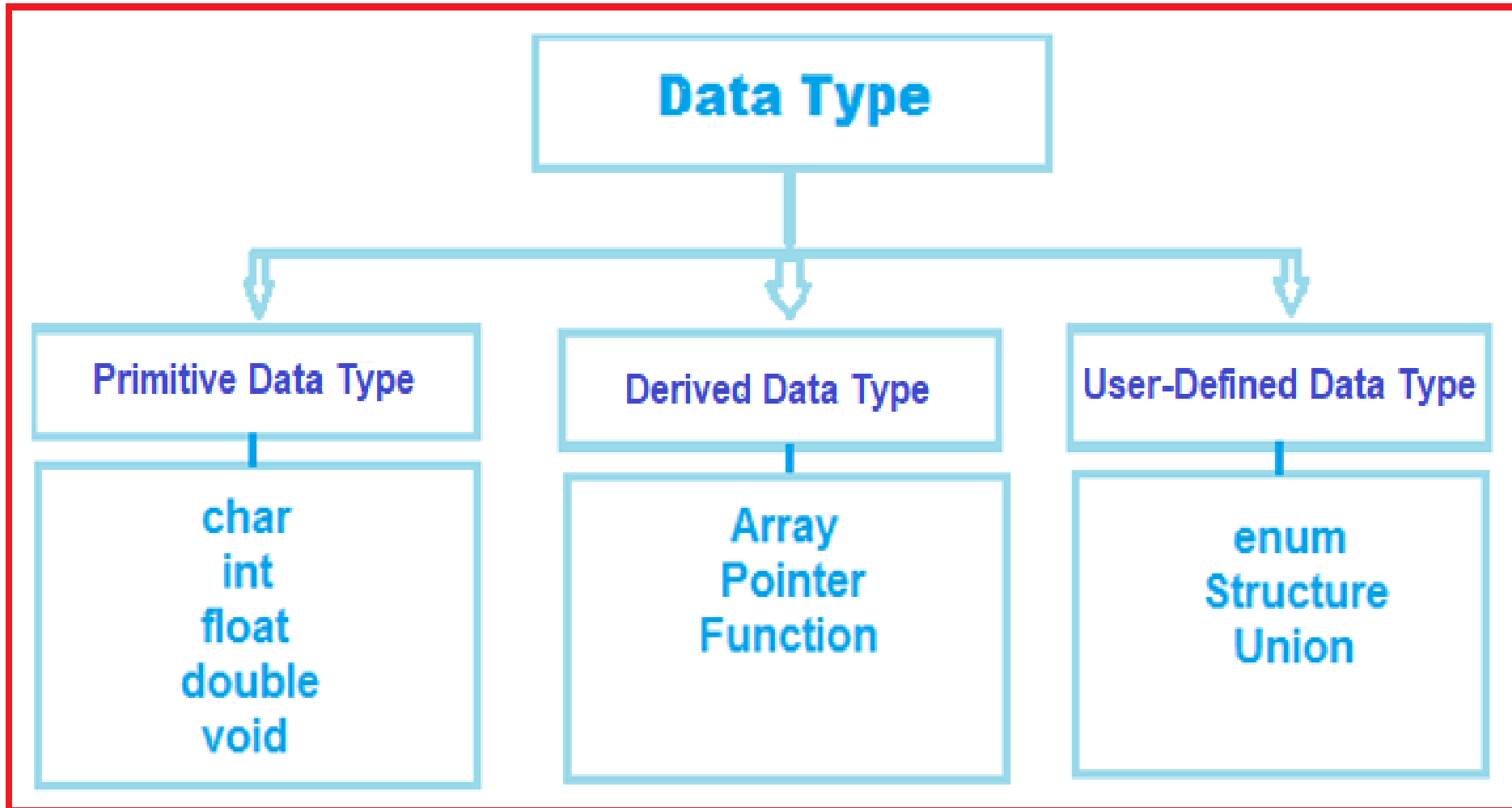
Num2=20

```
1.int a=10,b=20;//declaring 2 variable of integer type  
2.float f=20.8;  
3.char c='A'
```

## *Rules for defining variables*

- Must be declared first use.
- A variable can have alphabets, digits, and underscore.
- A variable name can start with the alphabet, and underscore only. It can't start with a digit.
- No whitespace is allowed within the variable name.
- A variable name must not be any reserved word or keyword, e.g. int, float, etc.
- For local variables ,declaration statements must be placed at the beginning of the block
- For global variables, declaration statements must be placed before first function definition.

# *Data Types in C:-*



# *Data Types in C*

- Each variable in C has an associated data type. Each data type requires different amounts of memory and has some specific operations which can be performed over it. Let us briefly describe them one by one:  
Following are the examples of some very common data types used in C:
- Basic data types(Primitive )
- **char:** The most basic data type in C. It stores a single character and requires a single byte of memory in almost all compilers.
  - `char ch1='a'; %c`
- **int:** As the name suggests, an int variable is used to store an integer.
  - `int num=10; %d`
- **float:** It is used to store decimal numbers (numbers with floating point value) with single precision.
  - `float fnum=10.5; %f`
- **double:** It is used to store decimal numbers (numbers with floating point value) with double precision. `Double dnum=10.657; %lf`

*Try Outs 1*

# ASCII in C

- ASCII stands for **American Standard Code for Information Interchange**.
- ASCII code is a character encoding scheme used to define the value of basic character elements for exchanging information in computers and other electronic devices.
- the ASCII code is a collection of 255 symbols in the character set, divided into two parts, the standard ASCII code, and the extended ASCII code. The standard ASCII code ranges from 0 to 127, 7 bits long, and the extended ASCII code from 128 to 255 is 8 bits long. These characters are a combination of symbol letters (uppercase and lowercase (a-z, AZ), digits (0-9), special characters (!, @, #, \$, etc.), punctuation marks, and control characters. Hence, we can say, each character has its own ASCII value.

## *Example:-*

- For example, when we enter a string as "HELLO", the computer machine does not directly store the string we entered. Instead, the system stores the strings in their equivalent ASCII value, such as '7269767679'. The ASCII value of H is 72, E is 69, L is 76, and O is 79.



# Format Specifier

- %c—Convert data into character
- %d—Convert data into signed int.
- %f—Convert data into float(6 precisions)
- %s—Converts data into string
- %ld—Converts data into long int
- %u—converts data into unsigned
- %x—converts data into hexadecimal format
- %o—Converts data into octal format
- %e—Converts data into exponential format

# Operators in C

## Operators in C

Unary operator

Binary operator

Ternary operator

| Operator              | Type                            |
|-----------------------|---------------------------------|
| ++, --                | Unary operator                  |
| +, -, *, /, %         | Arithmetic operator             |
| <, <=, >, >=, ==, !=  | Relational operator             |
| &&,   , !             | Logical operator                |
| &,  , <<, >>, ~, ^    | Bitwise operator                |
| =, +=, -=, *=, /=, %= | Assignment operator             |
| ?:                    | Ternary or conditional operator |

# *Operators in C*

- **Arithmetic Operators** (+, -, \*, /, %, post-increment, pre-increment, post-decrement, pre-decrement) +
- **Relational Operators** (==, !=, >, <, >= & <=) **Logical Operators** (&&, || and !)
- **Bitwise Operators** (&, |, ^, ~, >> and <<)
- **Assignment Operators** (=, +=, -=, \*=, etc)
- **Other Operators** (conditional, comma, sizeof, address, redirection)

# Comments in C

- Comments in C language are used to provide information about lines of code. It is widely used for documenting code. There are 2 types of comments in the C language.

1. Single Line Comments \

2. Multi-Line Comments \\*-----

## Single Line Comments

- Single line comments are represented by double slash \. Let's see an example of a single line comment in C.

## *Escape Sequence in C:-*

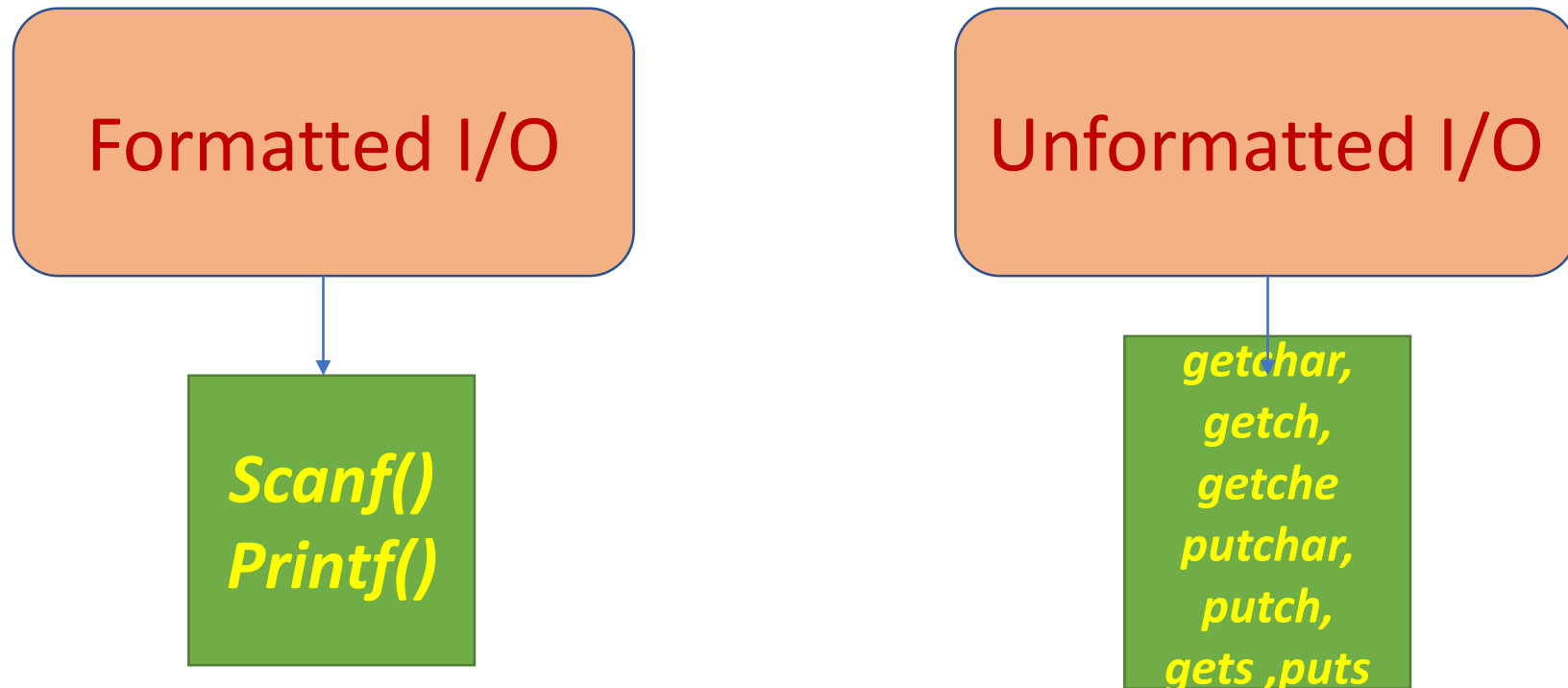
- An escape sequence in C language is a sequence of characters that doesn't represent itself when used inside string literal or character.
- Character Combination consisting of a backslash(\) followed by a letter. Or combination of digits is called escape sequence.
- **The backward slash is called the escape character** because it makes Character **following it escape from its original meaning and gives special meaning to it.**

## *List of Escape Sequences in C*

| Escape Sequence | Meaning            |
|-----------------|--------------------|
| \a              | Alarm or Beep      |
| \b              | Backspace          |
| \f              | Form Feed          |
| \n              | New Line           |
| \r              | Carriage Return    |
| \t              | Tab (Horizontal)   |
| \v              | Vertical Tab       |
| \\              | Backslash          |
| \'              | Single Quote       |
| \"              | Double Quote       |
| \?              | Question Mark      |
| \nnn            | octal number       |
| \xhh            | hexadecimal number |
| \0              | Null               |

# *Console I/O Functions:-*

- Classification of console I/O functions



## *Input Data:-*

- The field specification for reading an input number is ***:-%wd***
- ***W:- [is optional] is an integer number which specifies the field width of the number to be read.***



# ***Operator Precedence and Associativity in C***

- **Operator precedence** determines which operator is performed first in an expression with more than one operators with different precedence.
- **Operators Associativity** is used when two operators of same precedence appear in an expression. Associativity can be either **Left to Right** or **Right to Left**.  
**For example:** '\*' and '/' have same precedence and their associativity is **Left to Right**, so the expression "100 / 10 \* 10" is treated as "(100 / 10) \* 10".

# Operator Precedence and Associativity in C

| OPERATOR                             | TYPE                         | ASSOCIATIVITY |
|--------------------------------------|------------------------------|---------------|
| () [] . ->                           |                              | left-to-right |
| ++ -- + - ! ~ (type) * &<br>sizeof   | Unary Operator               | right-to-left |
| * / %                                | Arithmetic Operator          | left-to-right |
| + -                                  | Arithmetic Operator          | left-to-right |
| << >>                                | Shift Operator               | left-to-right |
| < <= > >=                            | Relational Operator          | left-to-right |
| == !=                                | Relational Operator          | left-to-right |
| &                                    | Bitwise AND Operator         | left-to-right |
| ^                                    | Bitwise EX-OR Operator       | left-to-right |
|                                      | Bitwise OR Operator          | left-to-right |
| &&                                   | Logical AND Operator         | left-to-right |
|                                      | Logical OR Operator          | left-to-right |
| ? :                                  | Ternary Conditional Operator | right-to-left |
| = += -= *= /= %= &= ^=<br> = <<= >>= | Assignment Operator          | right-to-left |
| ,                                    | Comma                        | left-to-right |

## ***Control Flow:-***

- 1) A C program executes one command after another from top to bottom.**
- 2) *Is there any thing that altering the flow of the program.***  
***Repeating commands according to the different statement.***
- 3) *I need control the flow of my program***

# *C Control Statements:-*

- **Decision & Selection Control Statements**:-if –else , switch case, conditional operators
- **Looping** :-While, do—while, for
- **Branching** :-break, continue, return, goto

# ***C if else Statement***

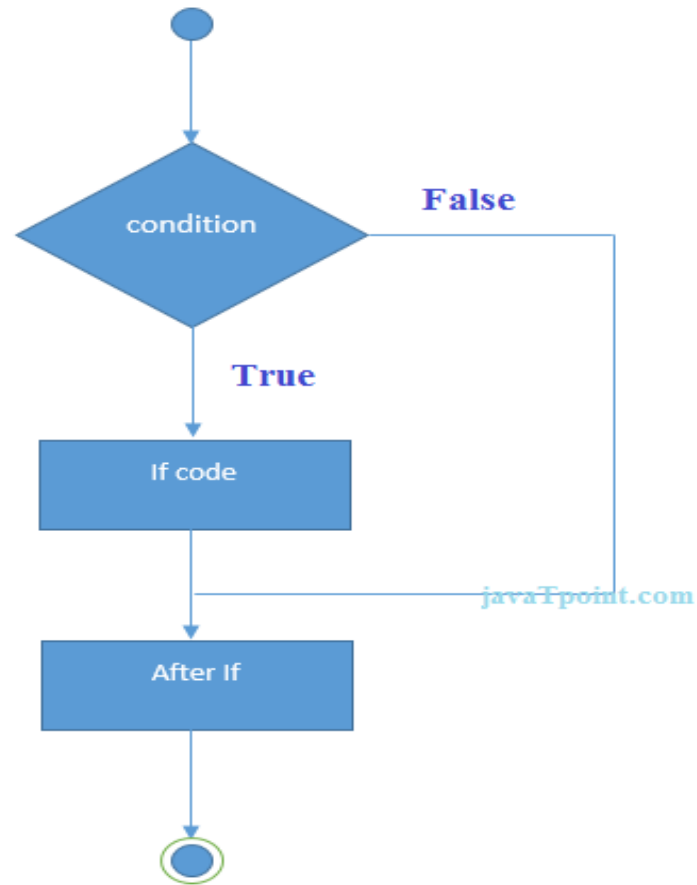
- The if-else statement in C is used to perform the operations based on some specific condition. The operations specified in if block are executed if and only if the given condition is true.
- There are the following variants of if statement in C language.
- If statement
- If-else statement
- If else-if ladder
- Nested if

# If Statement

- The if statement is used to check some given condition and perform some operations depending upon the correctness of that condition. It is mostly used in the scenario where we need to perform the different operations for the different conditions. The syntax of the if statement is given below.

```
if(expression){  
  //code to be executed  
}
```

# Flowchart of if statement in C



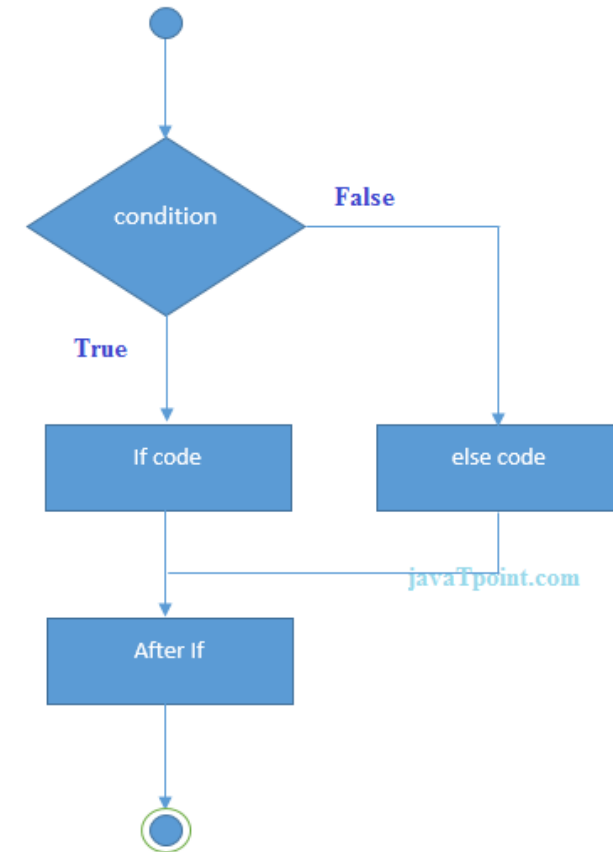
# If-else Statement

- The if-else statement is used to perform two operations for a single condition. **The if-else statement is an extension to the if statement using which, we can perform two different operations, i.e., one is for the correctness of that condition(i.e true part), and the other is for the incorrectness of the condition(i.e false part).** Here, we must notice that if and else block cannot be executed simultaneously Using if-else statement is always preferable since it always invokes an otherwise case with every if condition.



# Syntax & Flowchart of if---else

```
if(expression)
{
    //code to be executed if condition is true
}
else{
    //code to be executed if condition is false
}
```

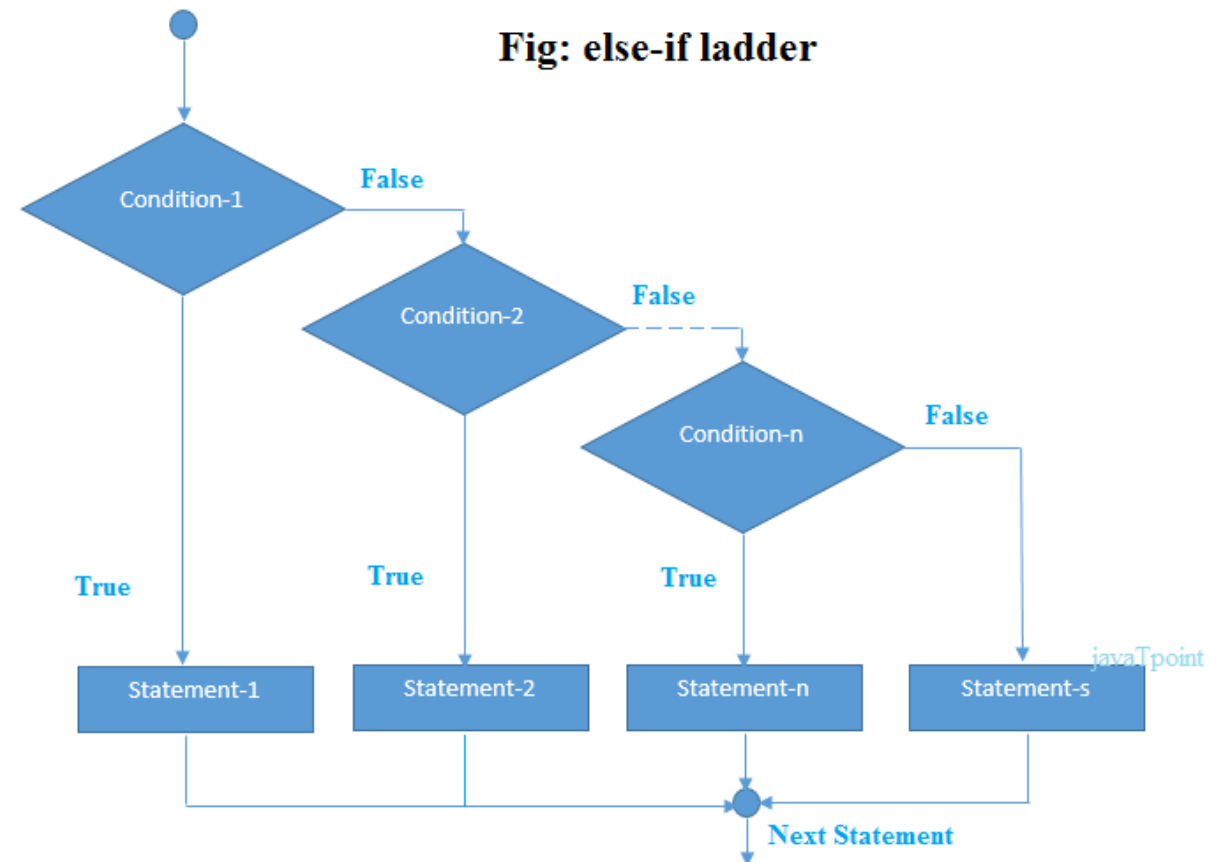


# If else-if ladder Statement

- The if-else-if ladder statement is an extension to the if-else statement. It is used in the scenario where there are multiple cases to be performed for different conditions. In if-else-if ladder statement, if a condition is true then the statements defined in the if block will be executed, otherwise if some other condition is true then the statements defined in the else-if block will be executed, at the last if none of the condition is true then the statements defined in the else block will be executed.

# Syntax & Flowchart:-

```
if(condition1)
{
//code to be executed if condition1 is true
}
else if(condition2){
//code to be executed if condition2 is true
}
else if(condition3){
//code to be executed if condition3 is true
}
...
else{
//code to be executed if all the conditions are false
}
```



# Nested if—else

```
if(Condition1)//outer
{
  if(Condition2)//inner
  {
    Statements;
  }
  else{
    Statements;
  }
}

if(condition3)
{-----}
else{
  -----}
}
```

# Conditional Operator(Ternary Operator)

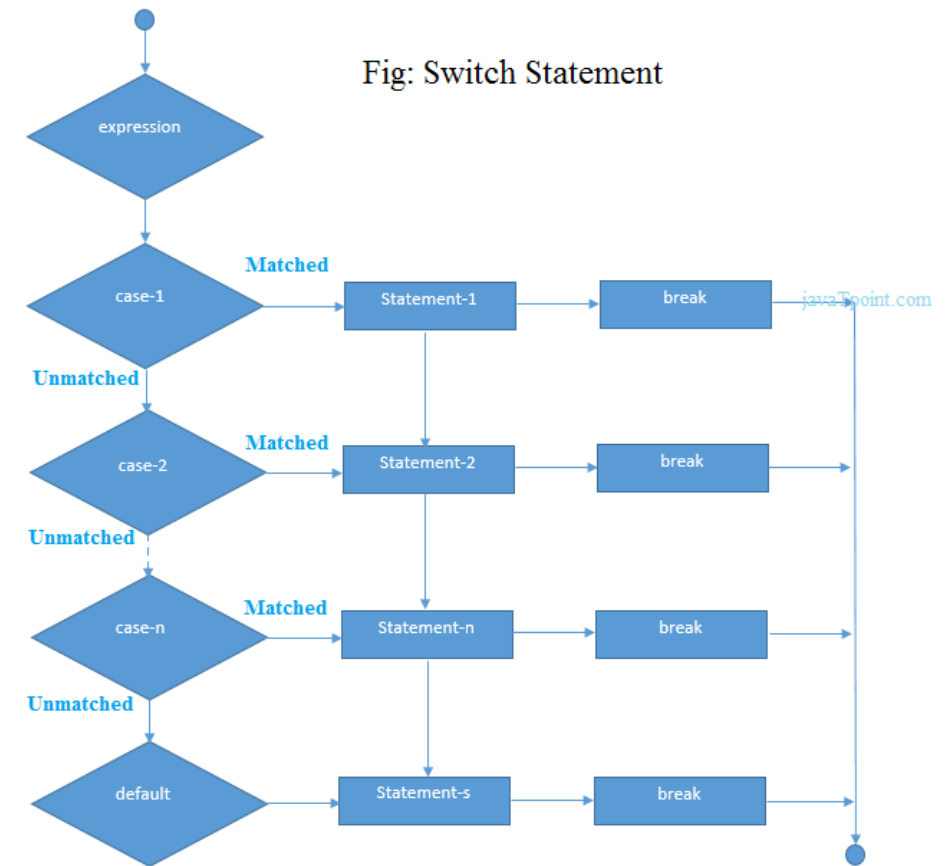
- Syntax:-
- Expression1 ?Expression 2 :Expression3
- Max=(num1>num2)? num1:num 2;

# C Switch Statement

- The switch statement in C is an alternate to if-else-if ladder statement which allows us to execute **multiple operations for the different possible values of a single variable called switch variable**. Here, We can define various statements in the multiple cases for the different values of a single variable.
- The Switch case statements is used for selective decision.
- The keyword **switch** is followed by an expressions which must yield an integer or character value.

# Syntax of Switch & Flowchart:-

```
switch(expression){  
  case value1:  
    //code to be executed;  
    break;  
  case value2:  
    //code to be executed;  
    break;  
  .....  
  default:  
    code to be executed if all cases are not matched;  
}
```



# Rules for switch statement in C language

- 1) The *switch expression* must be of an integer or character type.
- 2) The *case value* must be an integer or character constant.
- 3) The *case value* can be used only inside the switch statement.
- 4) The *break statement* in switch case is not must. It is optional. If there is no break statement found in the case, all the cases will be executed present after the matched case. It is known as *fall through* the state of C switch statement.



*Example:-*

**int** x,y,z;    case 1:2: 3 (invalid) case 1:

case 1:2:3(not allowed)

**char** a,b;

**float** f;

| Valid Switch      | Invalid Switch | Valid Case    | Invalid Case |
|-------------------|----------------|---------------|--------------|
| switch(x)         | switch(f)      | case 3;       | case 2.5;    |
| switch(x>y)       | switch(x+2.5)  | case 'a';     | case x;      |
| switch(a+b-2)     |                | case 1+2;     | case x+2;    |
| switch(func(x,y)) |                | case 'x'>'y'; | case 1,2,3;  |

# Type Casting in C

- Typecasting allows us to convert one data type into other. In C language, we use cast operator for typecasting which is denoted by (type).
- Syntax:  
1.(type)value;

# C Loops

- Loops are used when a task needs to be repeated a number of times
- For example
- Printing numbers from 1 to 100;
- Printing even numbers from 1 to 50
- Printing details of all bank customers
- Sequentially searching a record in a file
- Loops can be categorized as
- 1 Pre tested loop
- 2 Post tested loop

*Point to Remember:-*