

UNIT – 1 (part-1)

PROBLEM SOLVING METHODOLOGIS AND PROGRAMMING IN C

BCA SEM – 1

PROBLEM SOLVING METHODOLOGIS AND
PROGRAMMING IN C

Code : CS-01

Topics :

- ▶ Introduction of Computer Languages
- ▶ Introduction of Programming Concept
- ▶ Introduction of C Language (History & Overview)
- ▶ Difference between traditional and modern c.
- ▶ C character set
- ▶ C tokens
 - ▶ Keywords
 - ▶ Constants
 - ▶ Strings
 - ▶ Identifiers and variables
 - ▶ Operators (all 8 operators)
- ▶ Hierarchy of operators
- ▶ Type casting
- ▶ Data types in c
- ▶ PRE-PROCESSORS IN C

❖ Introduction of Computer Languages :

▶ What is Program ?

▶ Program is a set of data set of operation or set of instructions.

▶ Example :

$$\begin{array}{r} 10 \\ + 20 \\ \hline = 30 \end{array}$$

- Here, 10 20 are the data.
- + and = are symbols its used for perform the various kind of operations.
- We can say that this are the instructions for the computer
- Last 30 is the output (result) of given task.

❖ What is Programming ?

- A computer program consists of code that is executed on a computer to perform particular tasks. This code is written by programmers.

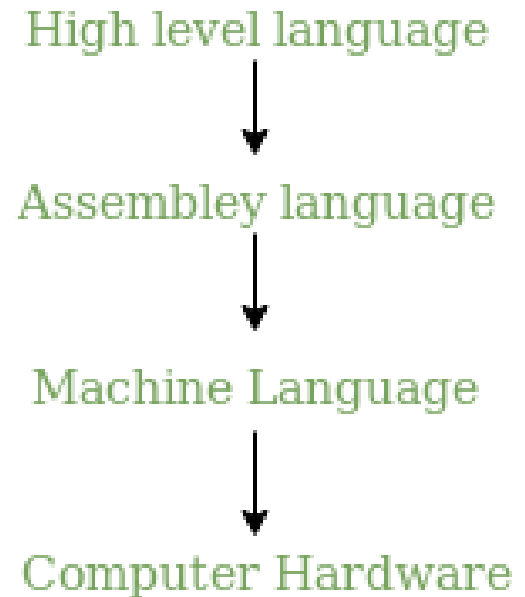
What is Programming Language :

- ▶ A programming language is a **computer language** that is used by **programmers (developers)** to **communicate with computers**.
- ▶ It is a set of instructions written in any specific language (C, C++, Java, Python) to perform a specific task.
- ▶ A programming language is mainly used to **develop desktop applications, websites, and mobile applications**.
- ▶ Mainly there are 2 types of Programming Language.

❖ Types Of Programming Language :

1. Low level Programming Language
 1. Machine Language
 2. Assembly Language
2. High Level Programming Language

❑ Hierarchy of Computer Language :



1. Low-level programming language :

- ▶ Low-level language is **machine-dependent (0s and 1s)** programming language. The processor runs low-level programs directly without the need of a compiler or interpreter, so the programs written in low-level language can be run very fast.
- ▶ Low-level language is further divided into two parts -
- ▶ **i. Machine Language (1GL (1st generation language))**
- ▶ When the human being started programming the computer the instructions were given to it in a language that it could easily understand. And that language was machine language.
- ▶ Computer can understand only one language without any translation which is machine language.
- ▶ Machine language is normally written in **binary** (0s and 1s).
- ▶ All the instructions and input data are fed to the computer in numeric form that is binary form.

Advantages of Machine Language :

Machine level instructions are directly executable
It can be executed very fast by computer
There is no need of translator

Disadvantages of Machine Language :

Machine dependent
Difficult to program
Difficult to modify

- ▶ **ii. Assembly Language (2GL (2nd generation language))**
- ▶ Assembly language is also a type of low-level programming language that is designed for specific processors. It represents the set of instructions in a **symbolic and human-understandable form**. It uses an assembler to convert the assembly language to machine language.
- ▶ **Advantages of Assembly Language :**
 - ▶ Easy to understand and use.
 - ▶ Easy to locate and correct errors.
 - ▶ Easy to modify
- ▶ **Disadvantages of Assembly Language :**
 - ▶ Knowledge of hardware to be required
 - ▶ Machine dependent
- ▶ **Note :** Assembly language need translator that is **assembler**.

2. High Level Programming Language (3GL) :

- ▶ High level language is user friendly language.
- ▶ High level language is combination of words of english and mathematical notation.

❑ EX : 1. COBOL

- ▶ 2. BASIC
- ▶ 3. JAVA
- ▶ 4. C
- ▶ 5. C++ etc....

▶ **Advantages :**

- ▶ Fast program development
- ▶ Machine independent
- ▶ Easy to learn and use
- ▶ Fewer error

Disadvantages :

Lower efficiency
Lack of flexibility.

NOTE FOR LOW LEVEL PROGRAMMING LANGUAGE :

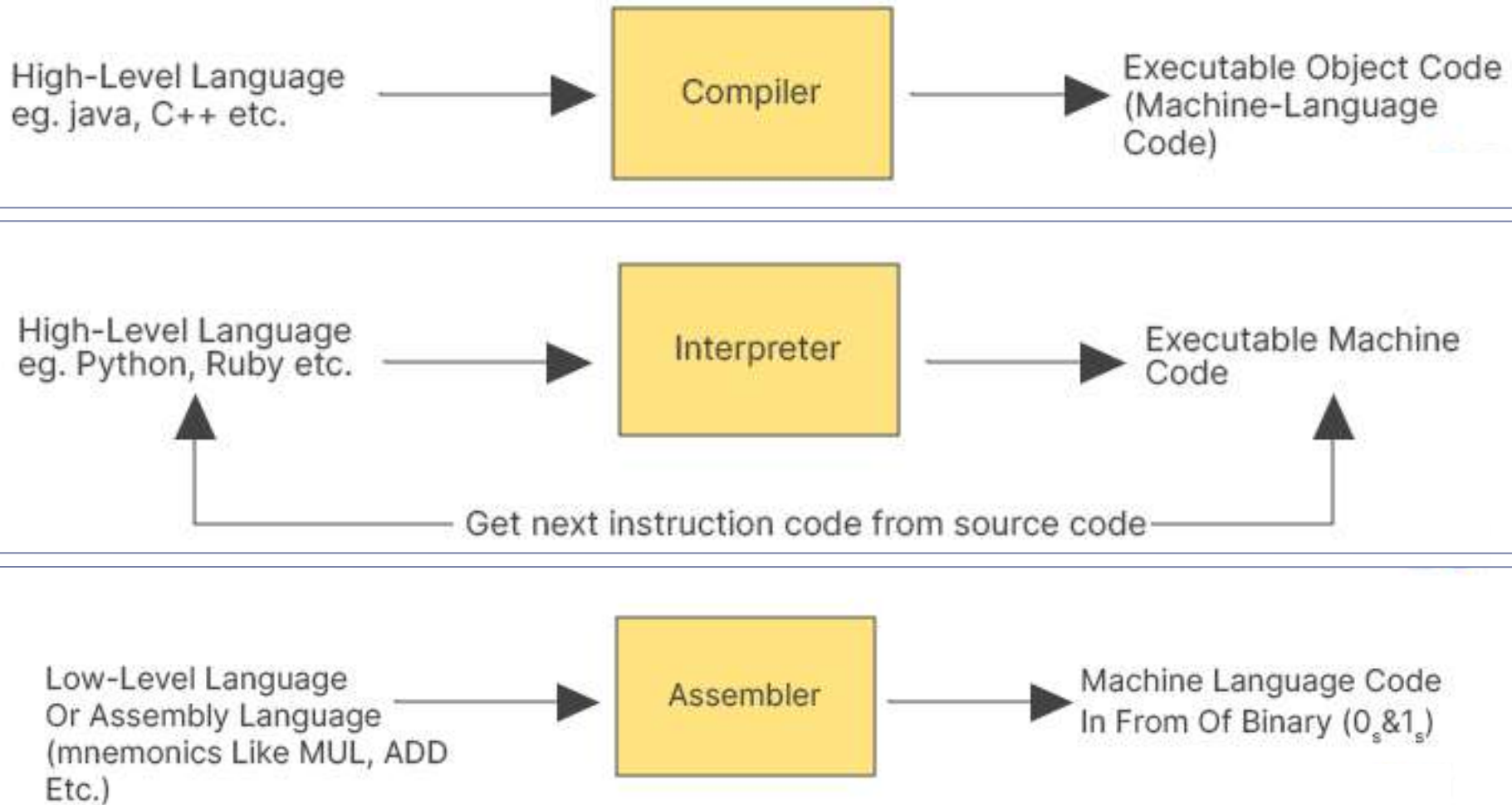
- ▶ Low level programming language is MACHINE ORIENTED language.
- ▶ It means Fast Program Execution.

NOTE FOR HIGH LEVEL PROGRAMMING LANGUAGE :

- ▶ High level programming language is PROBLEM ORIENTED language.
- ▶ It means Fast Program Development.

PARAMETERS	COMPILER	INTERPRETER	ASSEMBLER
Conversion	It converts the high-defined programming language into Machine language or binary code.	It also converts the program-developed code into machine language or binary code.	It converts programs written in the assembly language to the machine language or binary code.
Scanning	It scans the entire program before converting it into binary code.	It translates the program line by line to the equivalent machine code.	It converts the source code into the object code then converts it into the machine code.
Error Detection	Gives the full error report after the whole scan.	Detects error line by line. And stops scanning until the error in the previous line is solved.	It detects errors in the first phase, after fixation the second phase starts.
Code generation	Intermediate code generation is done in the case of Compiler.	There is no intermediate code generation.	There is an intermediate object code generation.
Execution time	It takes less execution time comparing to an interpreter.	An interpreter takes more execution time than the compiler.	It takes more time than the compiler.
Examples	C, C#, Java, C++	Python, Perl, VB, PostScript, LISP, etc...	GAS, GNU

Figure of Compiler , Interpreter & Assembler



❖ Introduction of C Language :

- ▶ **C programming language** was developed in 1972 by Dennis Ritchie at “AT&T’s Bell (American Telephone & Telegraph), laboratories” in the U.S.A.
- ▶ C was originally developed for **UNIX operating system** to beat the issues of previous languages such as B, BCPL, etc.
- ▶ The UNIX operating system development started in the year 1969, and its code was rewritten in C in the year 1972.
- ▶ It was named “C” because many of its features were derived from an earlier language called “B”, which according to ken thompson was a stripped-down version of the BCPL programming language.
- ▶ The American National Standard Institute (ANSI) formed a committee approved a version of C in 1989 which is known as ANSI C .
- ▶ **C language is also called middle level programming language**

Version of C

Before C Language	Year	Developed By
ALGOL	1960	International Group
BCPL	1967	Martin Richard
B	1970	Ken Thompson

Language	Year	Developed By
Traditional C	1972	Dennis Ritchie
K & R C	1978	Kernighan & Dennis Ritchie
ANSI / ISO C	1989-90	ANSI / ISO Committee
C99	1999	Standardization Committee
C11	2011	Standardization Committee
C18	2018	Standardization Committee

Features of 'C' Language :

- ▶ 'C' is powerful language.
- ▶ It flexible language provide fast program execution.
- ▶ C is a structured programming language .
- ▶ C is procedural language .
- ▶ C is case sensitive.
- ▶ All the 'c' statement are end with semicolon (;).
- ▶ There are 32 keywords in ANSI C .
- ▶ Mid-level programming language.
- ▶ It allows Low level access to information and commands why still retaining the probability and syntax of high level programming language.

Why C Language is Middle Level Language ?

- ▶ It binds the gap between machine level language and high-level language.
- ▶ It can be used for both, system programming (like as operating system)
- ▶ Middle-level language are more related to a machine as well as human language. So that's why it is called "Middle-level language".
- ▶ C language merges the best element of high-level language with the rule and flexibility of assembly language.
- ▶ C allows the manipulation of bits and addresses and bytes.
- ▶ That's why C language is a Middle Level Language .

❖ Difference between Traditional C & Modern C

Points	Traditional C	Modern C
Development	Early 1972	Year 1999 So it can also known as C99
Developer	Dennis Ritchie	Standardization Committee
Speed	Very Slow	Very High Speed
Features	Very Few features are exists	New features are added
Keyword	Only 32 keywords	C99 new five keywords added. So, total 37 keywords are in modern C.

❖ Structure of 'C' program :

- ▶ The structure of a C program can be mainly divided into six parts, each having its purpose.
 - ▶ It makes the program easy to read, easy to modify, easy to document, and makes it consistent in format.
1. Documentation Section
 2. Link Section (Pre-processor) (*)
 3. Definition Section
 4. Global Declaration Section
 5. Main() Function Section (*)
 6. Sub Programs Section

1. Documentation Section :

- ▶ Consists of the description of the program, programmer's name, and creation date.
- ▶ These are generally written in the form of **comments**.
- ▶ **What is COMMENT ?**
- ▶ The text include computer program for solve the purpose of providing information about the program.
- ▶ The text included in the comment is ignore by the compiler.
- ▶ There are two types of comment.
 1. Single Line Comment
 2. Multi Line Comment
- ▶ **1. Single Line Comment :**
- ▶ It Indicate by symbol of double slash (//).
- ▶ Single line comment is only one line comment.
- ▶ **2. Multi Line Comment :**
- ▶ It Indicate by the symbol (/* _____ */).
- ▶ It use one or more line comment.

2. Link Section (pre-processor section) :

- ▶ All **header files** are included in this section.
- ▶ Header files help us to access other's improved code into our code.
- ▶ A copy of these multiple files is inserted into our program before the process of compilation.
- ▶ It starts with the # sign.
- ▶ The include is the pre-processor directive.
- ▶ The angle bracket < > are required to indicate header file name.
- ▶ **Example:**
`#include<stdio.h>`
`#include<math.h>`

3. Definition Section :

- ▶ It define **symbolic constants**.
- ▶ This section is optional.
- ▶ A preprocessor directive in C is any statement that begins with the "#" symbol.
- ▶ The #define is a preprocessor compiler directive used to create constants.
- ▶ In simple terms, #define basically allows the macro definition, which allows the use of constants in our code.
- ▶ **Example :**
`#define BORN 2000`

4. Global Declaration Section :

- ▶ This section includes all **global variables, function declarations, and static variables**.
- ▶ The variables declared in this section can be used anywhere in the program.
- ▶ They're accessible to all the functions of the program.
- ▶ Hence, they are called global variables.

- ▶ **Example :**

```
int num=20;
```

```
void sum(int a,int b);
```

5. Main() Function Section :

- ▶ Every C program must have a main function.
- ▶ The main() function of the program is written in this section.
- ▶ Operations like declaration and execution are performed inside the curly braces of the main program.
- ▶ The main function have return type like, void, int, float, char, etc..
- ▶ **Example :**

```
void main()  
{  
    .....  
    .....  
}
```

```
int main()  
{  
    .....  
    return 0;  
}
```

6. Sub Program Section :

- ▶ This is an optional section.
- ▶ User-defined functions are created in this section of the program.
- ▶ When the user-defined function is called from the main() function, the control of the program shifts to the called function, and when it encounters a return statement, it returns to the main() function.

- ▶ **Example:**

```
int sum(int x, int y)
{
    return x+y;
}
```

Example of C program structure :

// Documentation section

```
/*  
    file: sum.c  
    author: you  
    description: program to find sum.  
*/
```

// Link section

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

// Definition section

```
#define X 20
```

// Global Declaration section

```
int sum(int y);
```

// Main() Function section

```
int main(void)  
{  
    int y = 55;  
    printf("Sum: %d", sum(y));  
    return 0;  
}
```

// Subprogram section

```
int sum(int y)  
{  
    return y + X;  
}
```

❖ C Character Set :

- ▶ In the C programming language, the character set refers to a set of all the valid characters that we can use in the source program for forming words, expressions, and numbers.
- ▶ The source character set contains all the characters that we want to use for the source program text.
- ▶ C language character set contains the following set of characters...
 1. Alphabets
 1. Lower case alphabets → a to z
 2. Upper case alphabets → A to Z
 2. Digits → 0-9
 3. Special Symbols → ~ @ # \$ % ^ & * () _ - + = { } [] ; : ' " / ? . > , < \ |
 4. White space characters → blank space, tab, new line

❖ C Tokens :

- ▶ Tokens are some of the most important elements used in the C language for creating a program.
- ▶ One can define tokens in C as the **smallest individual elements** in a program that is meaningful to the functioning of a compiler.
- ▶ A token is the smallest unit used in a C program.
- ▶ Each and every punctuation and word that you come across in a C program is token.
- ▶ A compiler breaks a C program into tokens and then proceeds ahead to the next stages used in the compilation process.
- ▶ **Types of Tokens in C language :**
 1. Identifiers and variables
 2. Keywords
 3. Constants
 4. Strings
 5. Operators (all 8 operators)

1. Identifier and Variables :

i. Identifier :

- ▶ Identifiers in C language represent the names of various entities such as arrays, functions, variables, user-defined data types, labels, etc.
- ▶ Identifier is created by the programmer in the program.
- ▶ They are name of user define program element which are the basic requirement.
- ▶ There is main future about length of identifier in C.
- ▶ EX :ANSI C recognizes only first 31 characters for the length of identifier.
- ▶ **Rules for Naming Identifiers :**
- ▶ Certain rules should be followed while naming c identifiers which are as follows:
 1. They must begin with a letter or underscore(_).
 2. They must consist of only letters, digits, or underscore. No other special character is allowed.
 3. It should not be a keyword.
 4. It must not contain white space.
 5. It should be up to 31 characters long as only the first 31 characters are significant.

ii. Variable :

- ▶ 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.
- ▶ In other words, variable is a container for storing data and values.
- ▶ It is a way to represent memory location through symbol so that it can be easily identified.
- ▶ **Rules for defining variables :**
 1. A variable can have alphabets, digits, and underscore.
 2. A variable name can start with the alphabet, and underscore only. It can't start with a digit.
 3. No whitespace is allowed within the variable name.
 4. A variable name must not be any reserved word or keyword, e.g. int, float, etc.
 5. Only first 32 characters of variable name are allowed.
 6. Keywords are not allowed in variable name.
 7. Variable name is case sensitive so total, TOTAL, Total are different.
 8. Variable name should be unique.

How to Declare Variable :

- ▶ There are 2 important characteristics for variable declaration.
- ▶ 1. to mention data type before variable name according to which type of data is to be store.

Example : int, float, char....

- ▶ 2. Assign appropriate value depending upon data type.

- ▶ **Syntax of variable declaration :**

- ▶ <Data Type> <Variable Name> [=Value of Variable];

- ▶ Example of declaring variable :

- ▶ int number;
- ▶ float price;
- ▶ char gender;

- ▶ Example of Variable Initialization :

```
int number=200;  
float price=55.5;  
char gender='M';
```

- ▶ **What is Garbage Value ?**

- ▶ Whenever any variable declare in the beginning of the function block and doesn't assign any value, at that time C compiler gives an unknown value to that variable during program execution process so this unknown value is called garbage value.

2. Keywords :

- ▶ Keywords in C language are the collection of **pre-defined** or **reserved words**.
- ▶ Keywords are case-sensitive and written in lower cases. Their meaning and functionality are already known to the compiler.
- ▶ We can't use these keywords as variable names or function names because by doing so, we are trying to assign a new meaning to the keyword, which is not allowed in C language.

- ▶ **There are a total 32 keywords supported by the C language:**

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

3. Constants :

- ▶ A constant is a value assigned to the variable which will remain the same throughout the program.
- ▶ The constant value cannot be changed during the program execution.
- ▶ Constants are also known as literals.
- ▶ A number, a character, a string of characters, an array, a structure, a union, a pointer, and an enum can be set as a constant.
- ▶ In the C programming language, A variable can be used as a constant by the following methods:
- ▶ 1. Using **const** keyword.
- ▶ 2. Using the **#define** preprocessor.
- ▶ Example :

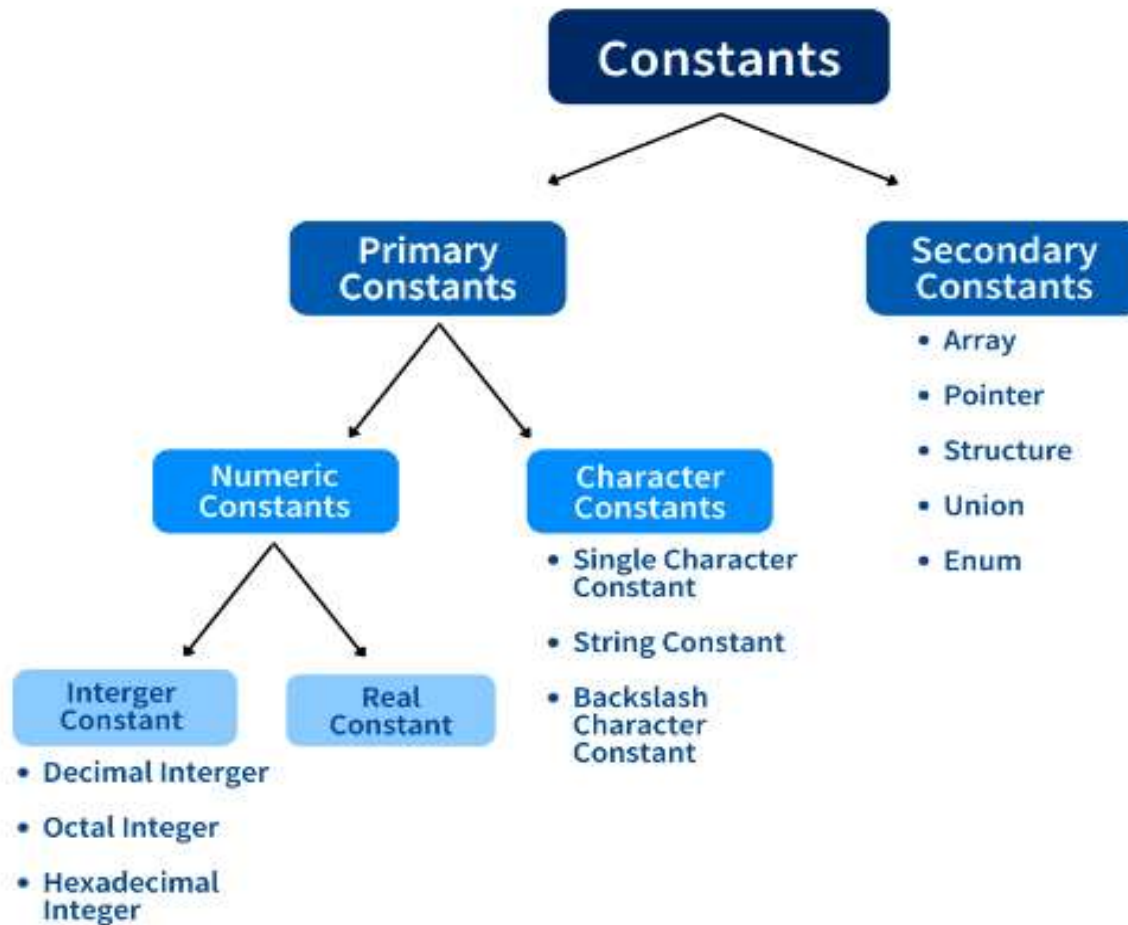
1st way to define

```
int const a=10;  
const int b=12;
```

2nd way to define

```
#define int a 10
```

Types of constant :



Primary Constants :

- ▶ There are two types of Primary Constants.
- ▶ 1. Numeric Constant
- ▶ 2. Character Constant

- ▶ **1. Numeric Constants :**
 - ▶ Numeric constants contain signed or unsigned numerals, or a zero or a decimal. In a nutshell, all types of numbers come under Numeric constants.

 - ▶ In numeric constant there are two main types :
 - ▶ i. Integer Constants
 - ▶ ii. Real Constants (Floating point)

i. Integer Constants :

- ▶ Integer constants are the numbers with decimals (base 10), hexadecimal (base 16), binary (base 2), or octal (base 8). We will understand this better once we look into each of the Integer constants.
- ▶ Let us know more about each integer constant :

INTEGER CONSTANT	VALUE
1. Decimal Integer	0-9
2. Octal Integer	0-7
3. Hexadecimal Integer	A for 10 B for 11 C for 12 D for 13 E for 14 F for 15

ii. Real or Floating Constants :

- ▶ A real or floating values include decimal value
- ▶ It could be either positive or negative.
- ▶ Followed by a decimal point and the fractional part is known as a real constant.
- ▶ Default sign is positive.
- ▶ No commas or blanks are allowed within a real constant.
- ▶ **Example :** 10.10 , 20.55 , 500.001
- ▶ **What is Mantissa Exponent ?**
 - (a) The mantissa part and the exponential part should be separated by a letter e.
 - (b) The mantissa part may have a positive or negative sign.
 - (c) Default sign of mantissa part is positive.
 - (d) The exponent must have at least one digit, which must be a positive or negative integer.
Default sign is positive.
 - (e) Range of real constants expressed in exponential form is $-3.4e38$ to $3.4e38$.
- ▶ Ex.: +3.2e-5 4.1e8 -0.2e+3 -3.2e-5

2. Character Constants :

- ▶ One or more characters are enclosed within a single quote (' ') or (" ") depending on the type of characters.
- ▶ There are two types of strings Constants
 - i. Single Character Constant
 - ii. String Constant

i. **Single Character Constant :**

Single Character constants having a single character enclosed within the ' ' (Single quote) are known as character constants.

Example : 'b' , 'c' , 'a'.

```
const char g='M';
```

```
char const gl='F';
```

▶ Function of single character constant :

FUNCTION NAME

1. scanf()
2. getch()
3. getche()
4. getchar()

▶ ii. **String Constant :**

- ▶ Character constants with various special symbols, digits, characters, and escape sequences enclosed within the ""(double quotes) are known as string constants.
- ▶ **Example :** “bca” , “hello”
 - ▶ Char name[5]=“hello”;
 - ▶ Every string value content null (\0) character at the end of string.
 - ▶ There are 2 type of functions

1. String Input Function :

1. scanf() :

It is use to read string data up to white space.

```
char abc[10];  
scanf("%s",&abc);
```

2. gets() :

it is built-in function to read string data.

string input function, it can take string including white space.

```
gets(10)  
gets(abc)
```

2. String Output Function :

1.printf() :

it is built-in formatted output function which display all type of data.

```
printf("hello bca");  
int a=10;  
printf("%d",a);  
printf("value of A :",a);
```

2. puts() :

it is string output function which display only string data.

```
char name[3]="BCA";  
puts(name);
```

Special Character Constants / Escape Sequence Character :

- ▶ The escape sequence in C is the characters or the sequence of characters that can be used inside the string literal.
- ▶ The purpose of the escape sequence is to represent the characters that cannot be used normally using the keyboard.
- ▶ Some escape sequence characters are the part of ASCII charset but some are not.
- ▶ Escape sequence characters are combination of back slash (\) followed by batters or by combinations.

List of Escape Sequence :

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
\0	Null

4. Operators in C:

- ▶ C Operators are symbols.
- ▶ That represent operations to be performed on one or more operands.
- ▶ C provides a wide range of operators, which can be classified into different categories based on their functionality.
- ▶ Operators are used for performing operations on variables and values.
- ▶ An expression is combination of operators and operands.
- ▶ **OPERAND :**
- ▶ Operand is a any kind of values or variables.
- ▶ Example : $10+20$

`int a=10; int b=20; int c=a+b;`

Here, A and B are variable that contain 10 and 20 .

C is also variable that contains a answer of given sum is 30.

Types of Operators in C :

- ▶ There are main 3 types of operators.
- ▶ That is divided into 7 types as require :

1. Arithmetic Operators

1. Unary
2. Binary

2. Assignment Operator

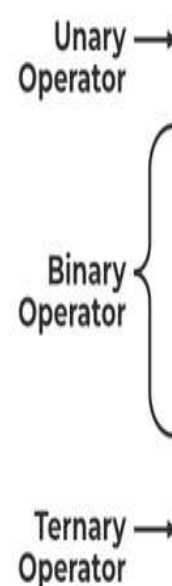
3. Relational Operator

4. Logical Operator

5. Conditional Operator

6. Bitwise Operator

7. Misc Operator



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

1. Arithmetic Operator :

- ▶ These operators are used to perform arithmetic/mathematical operations on operands.
- ▶ Examples: (+, -, *, /, %, ++, --).
- ▶ Arithmetic operators are of two types:
 - A. **Unary Operators**
 - B. **Binary Operators**

A. **Unary Operators (++ / --) :**

Operators that operate or work with a single operand are unary operators.

It is also called Prefix and Postfix operators.

Unary Operators are used to increment(++) or decrement(--) one from given value.

This operators are generally use for looping structure like while, do while, for loop.

❑ Prefix operators :

- ▶ An expression when use with Variable is increment or decrement first.
- ▶ Later the expression is evaluated first using **new value** of variable.
- ▶ Prefix operators first add 1 or -1 from operand.
- ▶ And then result is assign to the variable on the left.

▶ **Postfix Operators :**

- ▶ The expression is evaluated first using **original value** of variable.
- ▶ The variable is increment or decrement later.
- ▶ Postfix operator first assign the value to variable at the left side of assignment operator and then evaluate the expression or increment/decrement value by 1(one) and update itself with new value.

▶ **Example : let a=10; b=0;**

Postfix	b= a++;	now value of b=10 and a=11
Prefix	b= ++a;	now value of b=11 and a=11
Postfix	b= a--;	now value of b=10 and a=09
Prefix	b= --a;	now value of b=09 and a=09

B. Binary Operators (Arithmetic Operators) :

- ▶ Operators that operate or work with two operands are binary operators.
- ▶ The purpose of this operator is to perform mathematical operations like addition, subtraction, division, multiplication etc.

Operator	Operator Name	Example Here A=10 B=20	Result
+	Adds two operands.	$A + B$	30
-	Subtracts second operand from the first.	$A - B$	-10
*	Multiplies both operands.	$A * B$	200
/	Divides numerator by de-numerator.	B / A	2
%	Modulo Operator and remainder of after an integer division.	$B \% A$	0

Different Categories of arithmetic operators :

- ▶ 1. Integer Arithmetic :
 - ▶ When an arithmetic operations are perform on any integer number, at this time result must be an integer.
 - $10+2=12$
 - $10-2=8$
 - $10*2=20$
 - $10/2=5$
 - $10\%2=0$
- ▶ 2. Floating Point Arithmetic :
 - ▶ When an arithmetic operations are performed on any real or float number, at this time result must be real or float number.
- ▶ 3. Mix Mode Arithmetic :
 - ▶ When the any arithmetic operations carried out on any two different type data is called mix mode arithmetic.
 - ▶ When an arithmetic operations are performed on any one real number and any one integer number, at this time result must be real or float number.
 - ▶ Example : $10+5.5=15.5$

2. Assignment Operators :

- ▶ Assignment operators are used to assign value to a variable.
- ▶ The most common assignment operator is =
- ▶ The left side operand of the assignment operator is a variable and the right side operand of the assignment operator is a value.
- ▶ The value on the right side must be of the same data type as the variable on the left side otherwise the compiler will raise an error.

Operator	Example	Same as
=	a = b	a = b
+=	a += b	a = a+b
-=	a -= b	a = a-b
*=	a *= b	a = a*b
/=	a /= b	a = a/b
%=	a %= b	a = a%b

3. Relational Operator :

- ▶ Mainly used for checking relationships between operands With the help of this operator.
- ▶ You can check whether one operand is equal to or greater than the other operand or not.
- ▶ when the relation is true,
It returns 1(one).
- ▶ And when it is false,
it returns 0(zero).
- ▶ Relational Operators are used in decision-making.

Operator	What it does	Example	Result
==	Equal to	5==5	True(1)
>	Greater than	5>6	False(0)
<	Less than	6<7	True(1)
>=	Greater than equal to	2 >= 1	True(1)
<=	Less than equal to	1 <= 2	True(1)
!=	Not equal to	5 != 6	True(1)

4. Logical Operators :

- ▶ Logical Operators are used to combining two or more conditions/constraints or to complement the evaluation of the original condition in consideration.
- ▶ The result of the operation of a logical operator is a Boolean value either **true** or **false**.

Operators	Name Of Operator
&&	AND
	OR
!	NOT

AND Operator (&&) :

- ▶ Logical AND operator is represented by && symbol.
- ▶ If both expressions evaluate to True, then the result is True.
- ▶ If either any one expression evaluates to False, then the result is False.
- ▶ If both expressions evaluate to False, then the result is False.

▶ TRUTH TABLE OF AND :

Example :

(10>11) && (20>5)

result : FALSE

Condition 1	Condition 2	Result
TRUE	TRUE	TRUE
TRUE	FALSE	FALSE
FALSE	TRUE	FALSE
FALSE	FALSE	FALSE

OR Operator (||) :

- ▶ Logical OR operator is represented by || symbol.
- ▶ If both expressions evaluate to True, then the result is True.
- ▶ If either any one expression evaluates to False, then the result is True.
- ▶ If both expressions evaluate to False, then the result is False.

▶ TRUTH TABLE OF OR :

Example :

(10>11) || (20>5)

result :TRUE

Condition 1	Condition 2	Result
TRUE	TRUE	TRUE
TRUE	FALSE	TRUE
FALSE	TRUE	TRUE
FALSE	FALSE	FALSE

NOT Operator (!) :

- ▶ Logical NOT operator is represented by ! symbol.
- ▶ It is used to reverse the logical state of its operand.
- ▶ If a condition is true, then the Logical NOT operator will make it false and vice versa.

- ▶ TRUTH TABLE OF NOT :

Expression	Result
TRUE	FALSE
FALSE	TRUE

- ▶ Example :

!(10>9)= False

because : expression is true but we are use NOT Operator that's why the result was FALSE

5. Conditional Operator :

- ▶ Also known as Ternary operator.
 - ▶ It takes three operand.
 - ▶ The main purpose of conditional operators is in decision making statements.
 - ▶ It is similar to an if-else statement.
 - ▶ **Syntax of a conditional operator :**
 - ▶ Expression1? Expression2: Expression3 ;
 - ▶ Expression1 is the condition to be evaluated.
 - ▶ If the condition(Expression1) is True then we will execute and return the result of Expression2.
 - ▶ otherwise if the condition(Expression1) is false then we will execute and return the result of Expression3.
 - ▶ Example :
int a=0;
a=(10>32)?10:32;
- Output :
- a=32 because the condition is false.

6. Bitwise Operators :

- ▶ Bitwise operators are the operators which work on bits and perform the bit-by-bit operation.
- ▶ Mathematical operations like addition, subtraction, multiplication, division, etc are converted to bit-level which makes processing faster and easier to implement during computation and compiling of the program.

Operators	Meaning of operators
&	Bitwise AND
	Bitwise OR
^	Bitwise exclusive OR
~	Bitwise complement
<<	Shift left
>>	Shift right

- ▶ Here, we will assume that $A = 50$ and $B = 25$ in binary format as follows.

- ▶ $A = 00110010$

- ▶ $B = 00011001$

- ▶ -----

- ▶ $A \& B = 00010000$

- ▶ $A | B = 00111011$

- ▶ $A \wedge B = 00101011$

p	q	$p \& q$	$p q$	$p \wedge q$
0	0	0	0	0
0	1	0	1	1
1	1	1	1	0
1	0	0	1	1

- ▶ The table provided below demonstrates the bitwise operators supported by C. Assume variable 'A' holds 50 and variable 'B' holds 25.

7. Misc Operators :

- ▶ Besides all the other operators discussed above, the C programming language also offers a few other important operators including sizeof, comma, pointer(*), and conditional operator (? :).

Operator	Description
sizeof()	The sizeof is a unary operator that returns the size of data (constants, variables, array, structure, etc).
&	It returns the address of a memory location of a variable.
*	Pointer to a variable.
? :	conditional operator (? : in combination) to construct conditional expressions.

❖ Hierarchy of operators / precedence and order of evaluation :

- ▶ While executing an arithmetic statement, which has two or more operators, we may confuse to calculate the result.
- ▶ For example, the expression $2 * x - 3 * y$ corresponds to $(2x) - (3y)$ or to $2(x-3y)$?
- ▶ Similarly, $A / B * C$ correspond to $A / (B * C)$ or to $(A / B) * C$?
- ▶ To answer these questions, one has to understand the 'hierarchy' of operations. The priority in which the operations in an arithmetic statement are performed is called the hierarchy of operations.
- ▶ Precedence is also known as priority.

Precedence of Operators

▶ There are 2 different priorities of arithmetic operators :

1. High Priority: $*$ / $\%$
2. Low Priority: $+$ -

▶ The equation is evaluated in two passes :

1. First pass: High priority operators.
2. Second pass: Low priority operators.

▶ Expression: $x = 9 - 12 / 3 + 3 * 2 - 1$

▶ 1st Pass

▶ $x = 9 - 4 + 3 * 2 - 1$

▶ $x = 9 - 4 + 6 - 1$

▶

2nd Pass

$x = 5 + 6 - 1$

$x = 11 - 1$

$x = 10$

Rules for Evaluation of Expression :

- ▶ Parenthesized sub expression from left to right are evaluated.
- ▶ If parenthesis are nested evaluation begins with innermost braces
- ▶ If operators of same precedence are encounter then is used.
- ▶ Arithmetic expression are evaluated from left to right.

Example 1: Determine the hierarchy of operations and evaluate the following expression:

$$i = 2 * 3 / 4 + 4 / 4 + 8 - 2 + 5 / 8$$

Stepwise evaluation of this expression is shown below:

$$i = 2 * 3 / 4 + 4 / 4 + 8 - 2 + 5 / 8$$

$$i = 6 / 4 + 4 / 4 + 8 - 2 + 5 / 8$$

$$i = 1 + 4 / 4 + 8 - 2 + 5 / 8$$

$$i = 1 + 1 + 8 - 2 + 5 / 8$$

$$i = 1 + 1 + 8 - 2 + 0$$

$$i = 2 + 8 - 2 + 0$$

$$i = 10 - 2 + 0$$

$$i = 8 + 0$$

$$i = 8$$

operation: *

operation: /

operation: /

operation: /

operation: +

operation: +

operation: -

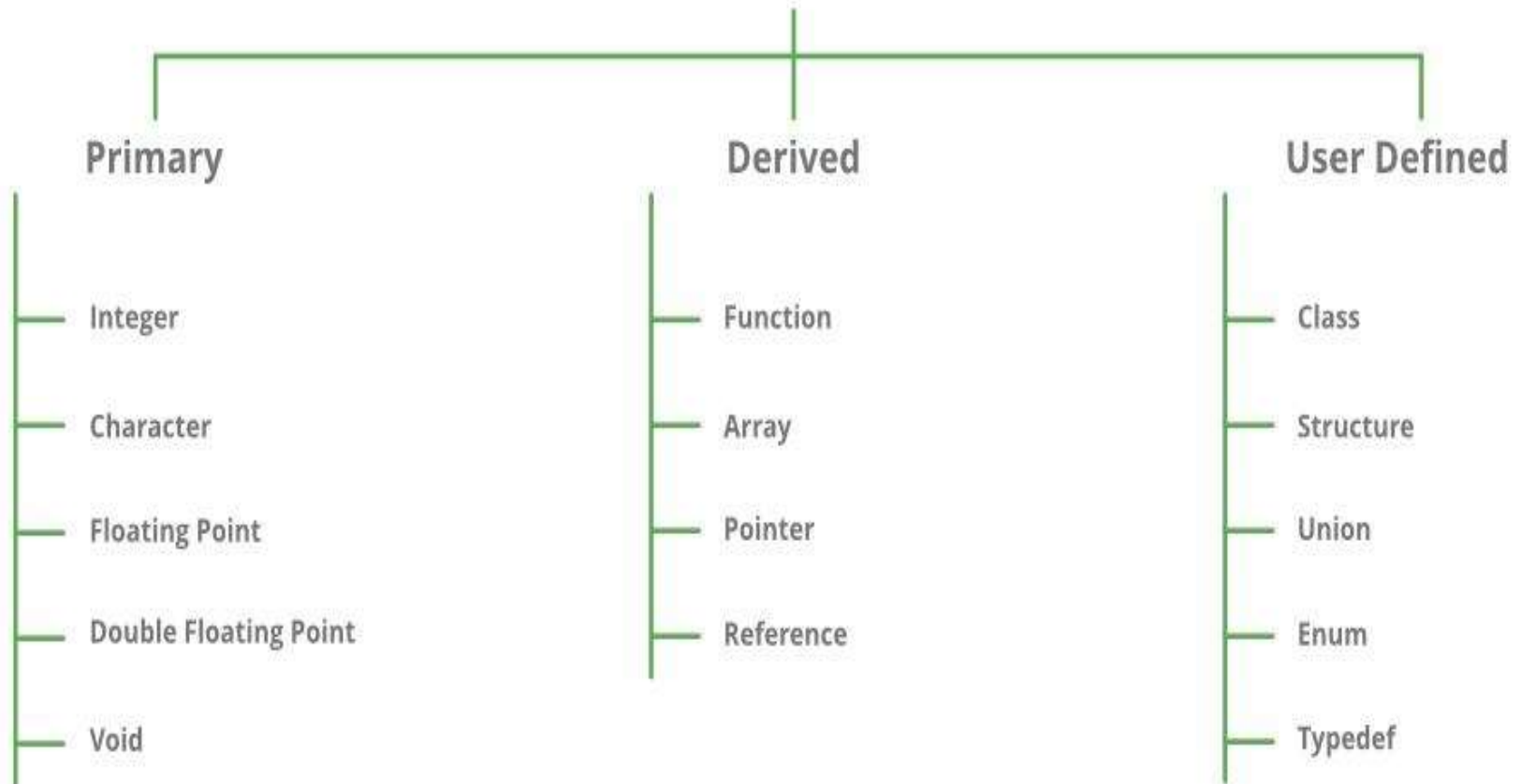
operation: +

❖ Data Types in C :

- ▶ Data type is an attribute of data which tells the C Compiler, which type of data a variable is holding.
- ▶ It can be of type integer, float(decimal), character , boolean(true/false) etc.
- ▶ Each data type requires different amounts of memory and has some specific operations which can be performed over it.

Types	Description
Primitive Data Types	Primitive data types are the most basic data types that are used for representing simple values such as integers, float, characters, etc.
User Defined Data Types	The user-defined data types are defined by the user himself.
Derived Types	The data types that are derived from the primitive or built-in data types are referred to as Derived Data Types.

DataTypes in C



Primary Data Types or Basic Data types :

- ▶ These are the most basic data types and all the other data typed are derived or made from them only. It contains integer, floating point and char.
- ▶ Five main types of primary/basic data types are:
 1. Char
 2. Integer
 3. Float
 4. Double
 5. Void

Data type Range, Format Specifiers in C :

Data Type	Keyword	Size (bytes)	Range	Format Specifier
Character	char	1 byte	-128 to +127	%c
	unsigned char	1 byte	0 to 255	%c
Integer	int	2 bytes	-32768 to +32767	%d
	unsigned int	2 bytes	0 to 65535	%u
	long int (signed)	4 bytes	-2,147,483,648 to 2,147,483,647	%ld
	unsigned long	4 bytes	0 to 4,294,967,295	%lu
Float	float	4 bytes	3.4E-38 to 3.4E+38	%f
	double	8 bytes	1.7E-308 to 1.7E+308	%lf
	long double	10 bytes	3.4E-4932 to 1.1E+4932	%Lf



1. Character Data type :

- ▶ Character data type allows its variable to store only a single character. The size of the character is 1 (one) byte. It is the most basic data type in C. It stores a single character and requires a single byte of memory in almost all compilers.
- ▶ The **char** keyword is used to declare the variable of character type.
- ▶ **char** *var_name*='a';

8	7	6	5	4	3	2	1
Sign							
2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0

2. Integer Data Type :

- ▶ The integer datatype in C is used to store the whole numbers without decimal values.
- ▶ It could be either positive or negative.
- ▶ Declaration has to be done using the keyword **int**.
- ▶ Octal values, hexadecimal values, and decimal values can be stored in int data type in C.
- ▶ **int** number=10;
- ▶ **Note:** The **long, short, signed and unsigned** are datatype modifier that can be used with some primitive data types to change the size or length of the datatype.

16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1
15															
2^{15}	2^{14}	2^{13}	2^{12}	2^{11}	2^{10}	2^9	2^8	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0

3. Float Data Type :

- ▶ **float** data type occupies **4 bytes** of memory to store real numbers that have at least one digit after the decimal point.
- ▶ It could be either positive or negative.
- ▶ Declaration has to be done using the keyword **float**.
- ▶ `float price=50.45;`

32	31	30	4	3	2	1
sign												
2^{31}	2^{30}	2^{29}	2^3	2^2	2^1	2^0

4. Double Data Type :

- ▶ A double data type in C is used to store decimal numbers (numbers with floating point values) with double precision. It is used to define numeric values which hold numbers with decimal values in C.
- ▶ The variable can be declared as double precision floating point using the **double keyword**.
- ▶ **double** a=23|23|23|2.|23|23;

5. Void Data Type :

- ▶ The void data type in C is used to specify that **no value** is present.
 - ▶ It does not provide a result value to its caller.
 - ▶ It has no values and no operations.
 - ▶ It is used to represent nothing. Void is used in multiple ways as function return type, function arguments as void.
-

❖ Check size of the data type :

- ▶ The size of the data types in C is dependent on the size of the architecture, so we cannot define the universal size of the data types. For that, the C language provides the `sizeof()` operator to check the size of the data types.
 - ▶ `int a=10;`
 - ▶ `printf("size of integer data type : %d",sizeof(a));`
 - ▶ **OUTPUT** : size of integer data type : 2
-

❖ Type Casting :

- ▶ When you convert the data type of a variable to another data type then this technique is known as typecasting or type-conversion.
- ▶ Types of Type casting in C :
 1. **Implicit Type Casting**
 2. **Explicit Type Casting**

1. Implicit Type Casting :

- ▶ Implicit type casting means conversion of data types without losing its original meaning.
- ▶ This type of typecasting is essential when you want to change data types **without** changing the significance of the values stored inside the variable.
- ▶ In other words ,When the type conversion is performed automatically by the compiler without programmers intervention is known as implicit type casting.

- ▶ **Example :**

```
int f= 9/4;
```

```
printf("f : %d\n", f );
```

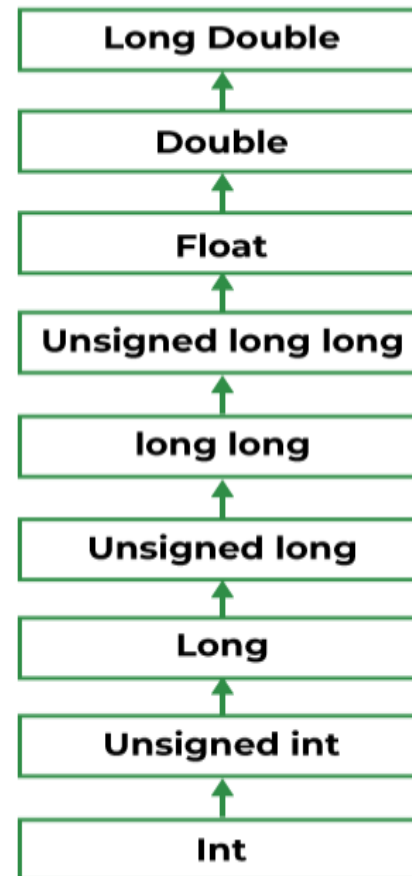
```
//Output: 2
```

POINTS :

- ▶ Implicit type of type conversion is also called as standard type conversion. We do not require any keyword or special statements in implicit type casting.
- ▶ Converting from smaller data type into larger data type is also called as **type promotion**.
- ▶ The implicit type conversion always happens with the compatible data types.

- ▶ The compiler first performs *integer promotion* .
- ▶ if the operands still have different types, then they are converted to the type that appears highest in the following hierarchy of arithmetic

conversion : -



2. Explicit Type Casting :

- ▶ There are some cases where if the datatype remains unchanged, it can give incorrect output.
- ▶ In such cases, typecasting can help to get the correct output and reduce the time of compilation.
- ▶ In explicit type casting, we have to force the conversion between data types. This type of casting is explicitly defined within the program.
- ▶ Explicit type casting is user-define.
- ▶ In this conversion, the user can define the type to which the expression is to be converted.
- ▶ **Cast Operator**
- ▶ The cast operator is a unary type operator that temporarily transforms an expression, variable, or constant to a specific data type.
- ▶ **Syntax:**
(data-type)expression;

Example :

```
float a=5.2l  
int b = (int)a + 1;  
printf("Value of b is %d\n",b);  
//output is Value of b is 8
```

❖ Pre-Processor in C :

- ▶ The **C Preprocessor** is not a part of the compiler, but is a **separate step in the compilation**
- ▶ process. In simple terms, a C Preprocessor is just a text substitution tool and it instructs the
- ▶ compiler to do required pre-processing before the actual compilation. We'll refer to the C
- ▶ Preprocessor as CPP.
- ▶ All preprocessor commands begin with a hash symbol (#). It must be the first nonblank
- ▶ character, and for readability, a preprocessor directive should begin in the first column. The
- ▶ following section lists down all the important preprocessor directives –

Pre-processor	Directive & Description
#define	Substitutes a preprocessor macro.
#include	Inserts a particular header from another file.
#undef	Undefines a preprocessor macro.
#ifdef	Returns true if this macro is defined.
#ifndef	Returns true if this macro is not defined.
#if	Tests if a compile time condition is true.
#else	The alternative for #if.
#elif	#else and #if in one statement.
#endif	Ends preprocessor conditional.
#error	Prints error message on stderr.

▶ **1. #include**

- ▶ The `#include` preprocessor directive is used to paste code of given file into current file. It is used include system-defined and user-defined header files. If included file is not found, compiler renders error. It has three variants:

▶ **#include <file>**

- ▶ This variant is used for system header files. It searches for a file named file in a list of directories specified by you, then in a standard list of system directories.

▶ **2. Macro's (#define)**

- ▶ Let's start with macro, as we discuss, a macro is a segment of code which is replaced by the value of macro. Macro is defined by `#define` directive.

▶ **Syntax**

- ▶ `#define token value`

- ▶ There are two types of macros:

1. Object-like Macros
2. Function-like Macros

Example :

```
#define PI 3.1415
```

▶ **3. #undef / #define :**

- ▶ To undefine a macro means to cancel its definition. This is done with the #undef directive.

▶ **Syntax:**

- ▶ #undef token **define and undefine example**

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

```
#define PI 3.1415
```

```
#undef PI
```

```
main() {  
    printf("%f",PI);  
}
```

Output:

- ▶ Compile Time Error: 'PI' undeclared

▶ **4. #ifdef**

- ▶ The #ifdef preprocessor directive checks if macro is defined by #define. If yes, it executes the code.

▶ **Syntax:**

```
#ifdef MACRO
```

```
    //code
```

```
#endif
```

▶ **5. #ifndef**

- ▶ The `#ifndef` preprocessor directive checks if macro is not defined by `#define`. If yes, it executes the code.

▶ **Syntax:**

```
#ifndef MACRO
```

```
    //code
```

```
#endif
```

▶ **6. #if**

- ▶ The `#if` preprocessor directive evaluates the expression or condition. If condition is true, it executes the code.

▶ **Syntax:**

```
#if expression
```

```
    //code
```

```
#endif
```

▶ **7. #else**

- ▶ The `#else` preprocessor directive evaluates the expression or condition if condition of `#if` is false. It can be used with `#if`, `#elif`, `#ifdef` and `#ifndef` directives.

▶ **Syntax:**

```
#if expression
//if code
#else
//else code
#endif
```

Syntax with #elif

```
#if expression
//if code
#elif expression
//elif code
#else
//else code
#endif
```

▶ **8. #error**

- ▶ The `#error` preprocessor directive indicates error. The compiler gives fatal error if `#error` directive is found and skips further compilation process.

▶ **9. #pragma**

- ▶ The `#pragma` preprocessor directive is used to provide additional information to the compiler. The `#pragma` directive is used by the compiler to offer machine or operating-system feature. Different compilers can provide different usage of `#pragma` directive.

▶ **Syntax:**

- ▶ `#pragma token`