Unit-2

Introduction to Programming in C

History Of C

The base or father of programming languages is 'ALGOL.' It was first introduced in 1960. 'ALGOL' was used on a large basis in European countries.

'ALGOL' introduced the concept of structured programming to the developer community.

In 1967, a new computer programming language was announced called as 'BCPL' which stands for Basic Combined Programming Language.

BCPL was designed and developed by Martin Richards, especially for writing system software.

This was the era of programming languages.

Just after three years, in 1970 a new programming language called 'B' was introduced by Ken Thompson that contained multiple features of 'BCPL.'

This programming language was created using UNIX operating system at AT&T and Bell Laboratories. Both the 'BCPL' and 'B' were system programming languages.

In 1972, a great computer scientist Dennis Ritchie created a new programming language called ‘C’ at the Bell Laboratories.

It was created from ‘ALGOL’, ‘BCPL’ and ‘B’ programming languages. ‘C’ programming language contains all the features of these languages and many more additional concepts that make it unique from other languages.

‘C’ is a powerful programming language which is strongly associated with the UNIX operating system. Even most of the UNIX operating system is coded in ‘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.

Let's see the programming languages that were developed 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

Following is the list of popular compilers available online:

- Clang compiler
- MinGW compiler (Minimalist GNU for Windows)
- Portable ‘C’ compiler
- Turbo C

Applications areas of C Language

The development of system software and desktop applications is mostly accomplished via the use of C programming. The following are some examples of C programming applications.

Operating Systems:

A high-level programming language built in the C programming language was used to construct the first operating system, which was UNIX.
Later on, the C programming language was used to write Microsoft Windows and several Android apps.

GUI (Graphical User Interface):

Since the beginning of time, Adobe Photoshop has been one of the most widely used picture editors.
It was created entirely with the aid of the C programming language.
Furthermore, C was used to develop Adobe Illustrator and Adobe Premiere.

Embedded Systems

Because it is directly related to the machine hardware, C programming is often regarded as the best choice for scripting programs and drivers for embedded systems, among other things.

Google

You can also use the C/C++ programming language to create the Google Chrome web browser and the Google File System.
Furthermore, the Google Open Source community includes many projects that are maintained with the aid of the C/C++ programming language.

Design of a Compiler

You can widely use the C programming language to develop compilers, one of its most popular applications.
Many other languages' compilers were created with the connection between C and low-level languages in mind, making it easier for the machine to grasp what was being written.
Many prominent compilers, such as Clang C, Bloodshed Dev-C, Apple C, and MINGW, were developed with the C programming language.

Mozilla Firefox and Thunderbird

Because Mozilla Firefox and Thunderbird were free and open-source email client projects, they were included here.
As a result, they were developed in the C/C++ programming language.

Gaming and animation

Because the C programming language is based on a compiler and is thus far quicker than Python or Java, it has gained popularity in the game industry.
 Some of the most basic games, such as the Dino game, Tic-Tac-Toe, and the Snake game, are written in C programming languages.

MySQL

MySQL is another open-source project that is used in relational database management systems (RDBMS).
It was developed in the C/C++ programming language.

Platforms for new programming languages

It is not only C that gave rise to C++. This programming language incorporates all the features of C while also incorporating the concept of object-oriented programming.
Still, it has also given rise to many other programming languages widely used in today's world, such as MATLAB and Mathematica.
It makes it possible for applications to run more quickly on a computer.

Translators of high-level languages into machine language

Interpreters are also computer programs that are used to translate high-level languages into machine language.
You may write language interpreters in the C programming language.
C language is used to write several computer language interpreters, such as the Python Interpreter, the MATLAB Interpreter, etc.

What is Algorithm

Writing a logical step-by-step method to solve the problem is called the algorithm. In other words, an algorithm is a procedure for solving problems. In order to solve a mathematical or computer problem, this is the first step in the process.

An algorithm includes calculations, reasoning, and data processing. Algorithms can be presented by natural languages, pseudocode, and flowcharts, etc.

What is 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:

Example

Let’s try to learn algorithm-writing by using an example.

Problem – Design an algorithm to add two numbers and display the result.

Step 1 – START

Step 2 – declare three integers **n1**, **n2** & **sum**

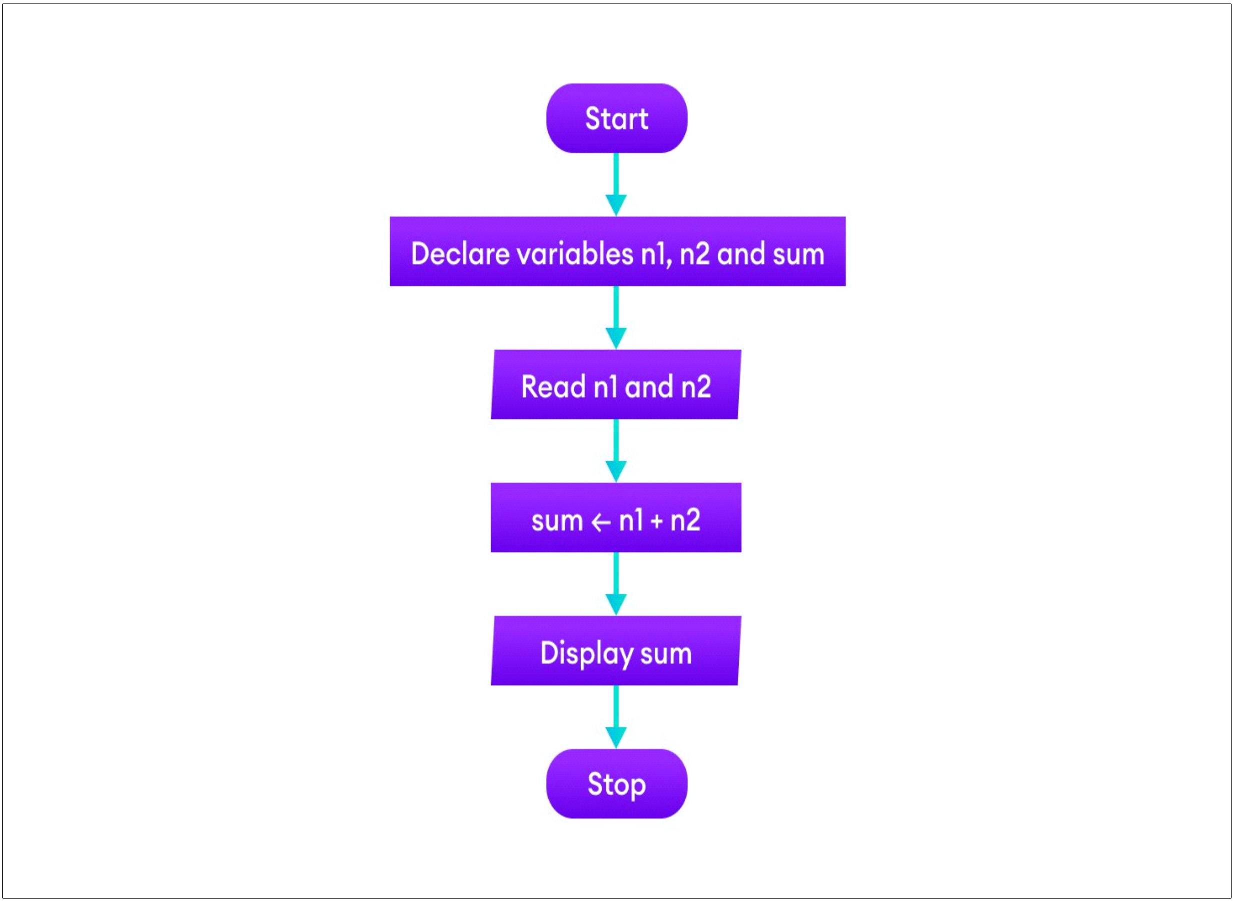
Step 3 – define values of **a** & **b**

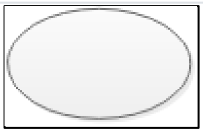
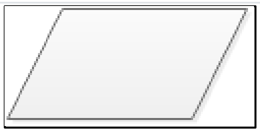
Step 4 – add values of **a** & **b**

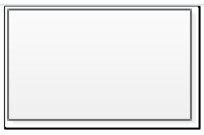
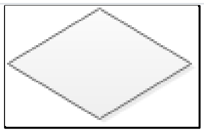
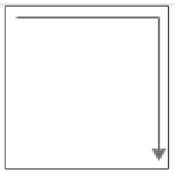
Step 5 – store output of step 4 to **sum**

Step 6 – print **sum**

Step 7 – STOP



Terminal Box - Start / End	
Input / Output	

Process / Instruction	
Decision	
Connector / Arrow	

Algorithm	Flowchart
It is a procedure for solving problems.	It is a graphic representation of a process.
The process is shown in step-by-step instruction.	The process is shown in block-by-block information diagram.
It is complex and difficult to understand.	It is intuitive and easy to understand.
It is convenient to debug errors.	It is hard to debug errors.
The solution is showcased in natural language.	The solution is showcased in pictorial format.
It is somewhat easier to solve complex problem.	It is hard to solve complex problem.
It costs more time to create an algorithm.	It costs less time to create a flowchart.

Structure Of C Program

A C program is divided into different sections. There are six main sections to a basic c program.

The six sections are,

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.

Basic Structure of C Program

Section	Description
Documentation	Consists of the description of the program, programmer's name, and creation date. These are generally written in the form of comments.
Link	All header files are included in this section which contains different functions from the libraries. A copy of these header files is inserted into your code before compilation.
Definition	Includes preprocessor directive, which contains symbolic constants. E.g.: #define allows us to use constants in our code. It replaces all the constants with its value in the code.
Global Declaration	Includes declaration of global variables, function declarations, static global variables, and functions.
Main() Function	For every C program, the execution starts from the main() function. It is mandatory to include a main() function in every C program.
Subprograms	Includes all user-defined functions (functions the user provides). They can contain the inbuilt functions, and the function definitions declared in the Global Declaration section. These are called in the main() function.

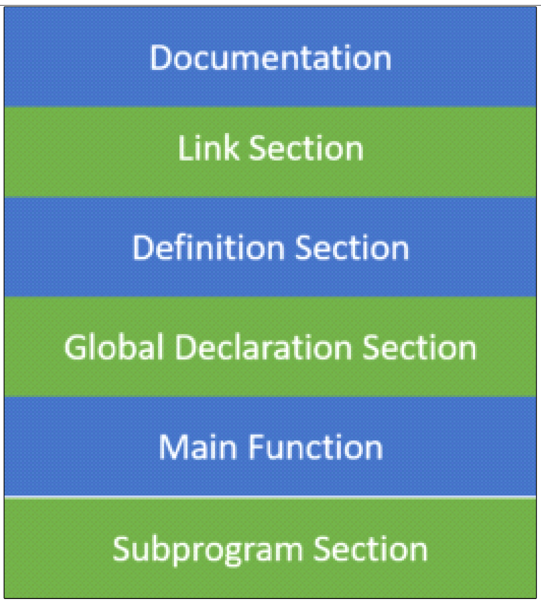


Figure: Basic Structure Of C Program

Moving on to the next bit of this basic structure of a C program article,

Documentation Section

The documentation section is the part of the program where the programmer gives the details associated with the program. He usually gives the name of the program, the details of the author and other details like the time of coding and description. It gives anyone reading the code the overview of the code.

Moving on to the next bit of this basic structure of a C program article,

Link Section

This part of the code is used to declare all the header files that will be used in the program. This leads to the compiler being told to link the header files to the system libraries.

Example

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

Moving on to the next bit of this basic structure of a C program article,

Definition Section

In this section, we define different constants. The keyword define is used in this part.

```
#define PI=3.14
```

Moving on to the next bit of this basic structure of a C program article,

Global Declaration Section

This part of the code is the part where the global variables are declared. All the global variable used are declared in this part. The user-defined functions are also declared in this part of the code.

```
float area(float r);
int a=7;
```

Moving on to the next bit of this basic structure of a C program article,

Main Function Section

Every C-programs needs to have the main function. Each main function contains 2 parts. A declaration part and an Execution part. The declaration part is the part where all the variables are declared. The execution part begins with the curly brackets and ends with the curly close bracket. Both the declaration and execution part are inside the curly braces.

```
int main(void)
{
int a=10;
printf(" %d", a);
return 0;
}
```

Moving on to the next bit of this basic structure of a C program article,

Sub Program Section

All the user-defined functions are defined in this section of the program.

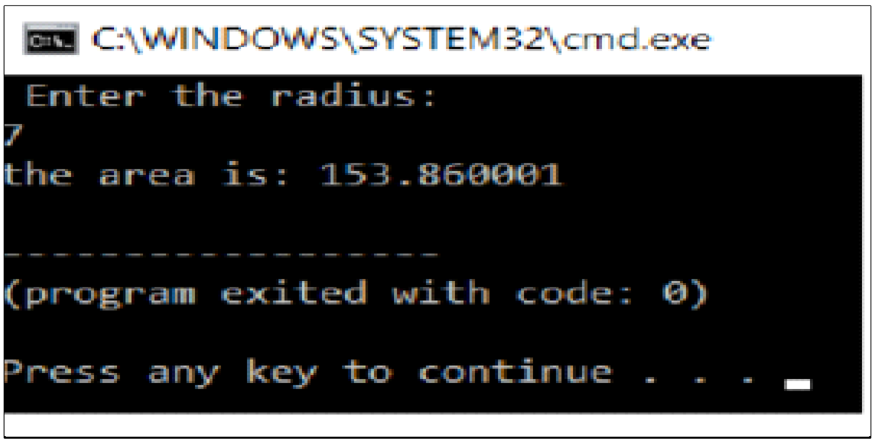
```
int add(int a, int b)
{
return a+b;
}
```

Sample Program

```
File Name: areaofcircle.c
* description: a program to calculate area of circle
*user enters the radius
**/
#include<stdio.h>//link section
#include<conio.h>
#define pi 3.14;//defination section
float area(float r);//global declaration
int main()//main function
{
float r;
printf(" Enter the radius:n");
scanf("%f",&r);
printf("the area is: %f",area(r));
return 0;
}
float area(float r)
{
return pi * r * r;//sub program
}
```

Output:

Output



```
 Enter the radius:
7
the area is: 153.860001

------------------
(program exited with code: 0)

Press any key to continue . . .
```

This is all about the basic structure of a C program. I hope you found this article informative and helpful, stay tuned for more tutorials on similar topics.You may also checkout our training programs, you can enroll here for live online training with 24/7 support and lifetime access.

Line 1: Comments - They are ignored by the compiler

This section is used to provide a small description of the program. The comment lines are simply ignored by the compiler, that means they are not executed. In C, there are two types of comments.

- Single Line Comments: Single line comment begins with // symbol. We can write any number of single line comments.
- Multiple Lines Comments: Multiple lines comment begins with /* symbol and ends with */. We can write any number of multiple lines comments in a program.

In a C program, the comment lines are optional. Based on the requirement, we write comments. All the comment lines in a C program just provide the guidelines to understand the program and its code.

Line 2: Preprocessing Commands

Preprocessing commands are used to include header files and to define constants. We use the **#include** statement to include the header file into our program. We use a **#define** statement to define a constant. The preprocessing statements are used according to the requirements. If we don't need any header file, then no need to write #include statement. If we don't need any constant, then no need to write a #define statement.

Line 3: Global Declaration

The global declaration is used to define the global variables, which are common for all the functions after its declaration. We also use the global declaration to declare functions. This global declaration is used based on the requirement.

Line 4: int main()

Every C program must write this statement. This statement (main) specifies the starting point of the C program execution. Here, main is a user-defined method which tells the compiler that this is the starting point of the program execution. Here, **int** is a data type of a value that is going to return to the Operating System after completing the main method execution. If we don't want to return any value, we can use it as **void**.

Line 5: Open Brace ({)

The open brace indicates the beginning of the block which belongs to the main method. In C program, every block begins with a '{' symbol.

Line 6: Local Declaration

In this section, we declare the variables and functions that are local to the function or block in which they are declared. The variables which are declared in this section are valid only within the function or block in which they are declared.

Line 7: Executable statements

In this section, we write the statements which perform tasks like reading data, displaying the result, calculations, etc., All the statements in this section are written according to the requirements.

Line 9: Closing Brace (})

The close brace indicates the end of the block which belongs to the main method. In C program every block ends with a '}' symbol.

Line 10, 11, 12, ...: User-defined function()

This is the place where we implement the user-defined functions. The user-defined function implementation can also be performed before the main method. In this case, the user-defined function need not be declared. Directly it can be implemented, but it must be before the main method. In a program, we can define as many user-defined functions as we want. Every user-defined function needs a function call to execute its statements.

General rules for any C program

- Every executable statement must end with a semicolon symbol (;).
- Every C program must contain exactly one main method (Starting point of the program execution).
- All the system-defined words (keywords) must be used in lowercase letters.
- Keywords can not be used as user-defined names(identifiers).
- For every open brace ({), there must be respective closing brace (}).
- Every variable must be declared before it is used.

Variables in C language

A variable is a name of memory location. It is used to store data. Changed the variables value keeps changing during the execution of the program and it can be reused many times.

It is a way to represent memory location through symbol so that it can be easily identified.

The C language demands that you declare the name of each variable that you are going to use and its type, or class, before you actually try to do anything with it.

Rules for naming variables :

• First letter must be an alphabet.

• Only alphabets, digits and underscore are allowed in variable names.

• We cannot use any other symbols like comma, hyphen, and period….etc.

• Spaces are strictly not allowed.

• Should not be a keyword like if, else, while…etc.

• The variable name cannot contain a maximum length of 32 characters.

Let's see the syntax to declare a variable :

```
<data type> variable-list;
```

The example of declaring variable is given below :

```
int a;
float f;
```

Here a, f are variables and int, float are data types.

We can also provide values while declaring the variables as given below :

int a=10, b=20; // this is called declaration, not an assignment.

float f;

f=15.5; //this is not an initialization, it is called assignment.

The Programming language C has two main variable types

1. Local Variables

2. Global Variables

Local Variables

Local variables scope is confined within the block or function where it is defined. Local variables must always be defined at the top of a block.

When a local variable is defined, it is not initialized by the system, you must initialize it yourself.

When execution of the block starts the variable is available, and when the block ends the variable 'dies'. It must be declared at the start of the block.

Example :

```
Void main()
{
int x=10; //local variable
}
```

Global Variables

Global variable is defined at the top of the program file and it can be visible and modified by any function that may reference it.

Global variables are initialized automatically by the system when you define them.

If same variable name is being used for global and local variable then local variable takes preference in its scope. But it is not a good practice to use global variables and local variables with the same name.

```
#include <stdio.h>
#include<conio.h>
int x = 20;//global variable
```

```
void function1()
{
  printf("%d\n" , x);
}
void function2()
{
  printf("%d\n" , x);
}
int main() {

  function1();
  function2();
    return 0;
}
```

Output

```
20

20
```

Tokens in C

Tokens in C is the most important element to be used in creating a program in C.

We can define the token as the smallest individual element in C.

For `example, we cannot create a sentence without using words; similarly, we cannot create a program in C without using tokens in C.

Therefore, we can say that tokens in C is the building block or the basic component for creating a program in C language

.Classification of tokens in C

Tokens in C language

can be divided into the following categories:

Tokens in C

Keywords in C

Identifiers in C

Strings in C

Operators in C

Constant in C

Special Characters in C

Keywords in C Language

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 C language. No header file is needed to include the keywords.

As every language has words to construct statements, C programming also has words with a specific meaning which are used to construct c program instructions. In the C programming language, keywords are special words with predefined meaning. Keywords are also known as reserved words in C programming language.

In the C programming language, there are **32 keywords**. All the 32 keywords have their meaning which is already known to the compiler.

Keywords are the reserved words with predefined meaning which already known to the compiler

Whenever C compiler come across a keyword, automatically it understands its meaning.

Properties of Keywords

- All the keywords in C programming language are defined as lowercase letters so they must be used only in lowercase letters
- Every keyword has a specific meaning, users can not change that meaning.
- Keywords can not be used as user-defined names like variable, functions, arrays, pointers, etc...
- Every keyword in C programming language represents something or specifies some kind of action to be performed by the compiler.

The following table specifies all the 32 keywords with their meaning...

A list of 32 keywords in 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

32 Keywords in C Programming Language with their Meaning

S.No	Keyword	Meaning
1	auto	Used to represent automatic storage class
2	break	Unconditional control statement used to terminate swicth & looping statements
3	case	Used to represent a case (option) in switch statement
4	char	Used to represent character data type
5	const	Used to define a constant
6	continue	Unconditional control statement used to pass the control to the begining of looping statements
7	default	Used to represent a default case (option) in switch statement
8	do	Used to define do block in do-while statement
9	double	Used to present double datatype
10	else	Used to define FALSE block of if statement
11	enum	Used to define enumarated datatypes
12	extern	Used to represent external storage class
13	float	Used to represent floating point datatype
14	for	Used to define a looping statement
15	goto	Used to represent unconditional control statement
16	if	Used to define a conditional control statement
17	int	Used to represent integer datatype
18	long	It is a type modifier that alters the basic datatype
19	register	Used to represent register storage class
20	return	Used to terminate a function execution
21	short	It is a type modifier that alters the basic datatype
22	signed	It is a type modifier that alters the basic datatype
23	sizeof	It is an operator that gives size of the memory of a variable
24	static	Used to create static variables - constants
25	struct	Used to create structures - Userdefined datatypes
26	switch	Used to define switch - case statement
27	typedef	Used to specify temporary name for the datatypes
28	union	Used to create union for grouping different types under a name
29	unsigned	It is a type modifier that alters the basic datatype
30	void	Used to indicate nothing - return value, parameter of a function
31	volatile	Used to creating volatile objects
32	while	Used to define a looping statement

- All the keywords are in lowercase letters
- Keywords can't be used as userdefined name like variable name, function name, lable, etc...
- Keywords are also called as Reserved Words

C Constants

In C programming language, a constant is similar to the variable but the constant hold only one value during the program execution. That means, once a value is assigned to the constant, that value can't be changed during the program execution. Once the value is assigned to the constant, it is fixed throughout the program. A constant can be defined as follows...

A constant is a named memory location which holds only one value throughout the program execution.

In C programming language, a constant can be of any data type like integer, floating-point, character, string and double, etc.,

Creating constants in C

In a c programming language, constants can be created using two concepts...

- Using the 'const' keyword
- Using '#define' preprocessor

Using the 'const' keyword

We create a constant of any datatype using 'const' keyword. To create a constant, we prefix the variable declaration with 'const' keyword.

The general syntax for creating constant using 'const' keyword is as follows...

```
const datatype constantName ;
```

OR

```
const datatype constantName = value ;
```

Example

```
const int x = 10 ;
```

Here, 'x' is a integer constant with fixed value 10.

Types of constants in C

Constant	Example
Integer constant :	10, 11, 34, etc.
Floating-point constant :	45.6, 67.8, 11.2, etc.
Octal constant :	011, 088, 022, etc.
Hexadecimal constant:	0x1a, 0x4b, 0x6b, etc.
Character constant :	'a', 'b', 'c', etc.
String constant :	"java", "c++", ".net", etc.

Integer constants

An integer constant can be a decimal integer or octal integer or hexadecimal integer. A decimal integer value is specified as direct integer value whereas octal integer value is prefixed with 'o' and hexadecimal value is prefixed with 'OX'.

An integer constant can also be unsigned type of integer constant or long type of integer constant. Unsigned integer constant value is suffixed with 'u' and long integer constant value is suffixed with 'l' whereas unsigned long integer constant value is suffixed with 'ul'.

Example

125 -----> Decimal Integer Constant

O76 -----> Octal Integer Constant

OX3A -----> Hexa Decimal Integer Constant

50u -----> Unsigned Integer Constant

30l -----> Long Integer Constant

100ul -----> Unsigned Long Integer Constant

Example: Program for integer constatnt

```
#include<stdio.h>
#include<conio.h>
void main(){

  int i = 9 ;
  const int x = 10 ;

  i = 15 ;
  x = 100 ; // creates an error
  printf("i = %d\n x = %d", i, x ) ;

}
```

The above program gives an **error** because we are trying to change the constant variable value (x = 100).

Floating Point constants

A floating-point constant must contain both integer and decimal parts. Some times it may also contain the exponent part. When a floating-point constant is represented in exponent form, the value must be suffixed with 'e' or 'E'.

Example

The floating-point value **3.14** is represented as **3E-14** in exponent form.

Character Constants

A character constant is a symbol enclosed in single quotation. A character constant has a maximum length of one character.

Example

'A'

'2'

'+'

In the C programming language, there are some predefined character constants called escape sequences. Every escape sequence has its own special functionality and every escape sequence is prefixed with '\' symbol. These escape sequences are used in output function called 'printf()'.

String Constants

are always represented as an array of characters having null character '\0' at the end of the string.
This null character denotes the end of the string. Strings in C are enclosed within double quotes, while characters are enclosed within single characters
. The size of a string is a number of characters that the string contains.

Now, we describe the strings in different ways:

char a[10] = "javatpoint"; // The compiler allocates the 10 bytes to the 'a' array.

char a[] = "javatpoint"; // The compiler allocates the memory at the run time.

char a[10] = {'j','a','v','a','t','p','o','i','n','t','\0'}; // String is represented in the form of characters.

All the above three defines the same string constant.

Using '#define' preprocessor

We can also create constants using '#define' preprocessor directive. When we create constant using this preprocessor directive it must be defined at the beginning of the program (because all the preprocessor directives must be written before the global declaration).

We use the following syntax to create constant using '#define' preprocessor directive...

```
#define CONSTANTNAME value
```

Example

#define PI 3.14

Here, **PI** is a constant with value **3.14**

Example Program

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

#defien  PI  3.14

void main(){
   int r, area ;

   printf("Please enter the radius of circle : ") ;
   scanf("%d", &r) ;

   area = PI * (r * r) ;

   printf("Area of the circle = %d", area) ;
}
```

Operators in C:

is a special symbol used to perform the functions. The data items on which the operators are applied are known as operands.

Operators are applied between the operands.

Depending on the number of operands, operators are classified as follows:

Unary Operator

A unary operator is an operator applied to the single operand. For example: increment operator (++), decrement operator (--), sizeof, (type)*.

Binary Operator

The binary operator is an operator applied between two operands. The following is the list of the binary operators:

Arithmetic Operators
Relational Operators
Shift Operators
Logical Operators
Bitwise Operators
Conditional Operators
Assignment Operator
Misc Operator

Ternary operator:

The syntax of ternary operator is :

testCondition ? expression1 : expression 2;

The testCondition is a boolean expression that results in either true or false. If the condition is

true - expression1 (before the colon) is executed
false - expression2 (after the colon) is executed

The ternary operator takes 3 operands (condition, expression1 and expression2).

Hence, the name ternary operator.

Special characters in C

Some special characters are used in C, and they have a special meaning which cannot be used for another purpose.

Square brackets [ ]: The opening and closing brackets represent the single and multidimensional subscripts.

Simple brackets ( ): It is used in function declaration and function calling. For example, printf() is a pre-defined function.

Curly braces { }: It is used in the opening and closing of the code. It is used in the opening and closing of the loops.

Comma (,): It is used for separating for more than one statement and for example, separating function parameters in a function call, separating the variable when printing the value of more than one variable using a single printf statement.

Hash/pre-processor (#): It is used for pre-processor directive. It basically denotes that we are using the header file.

Asterisk (*): This symbol is used to represent pointers and also used as an operator for multiplication.

Tilde (~): It is used as a destructor to free memory.

Period (.): It is used to access a member of a structure or a union.

C Identifiers

In C programming language, programmers can specify their name to a variable, array, pointer, function, etc... An identifier is a collection of characters which acts as the name of variable, function, array, pointer, structure, etc... In other words, an identifier can be defined as the user-defined name to identify an entity uniquely in the c programming language that name may be of the variable name, function name, array name, pointer name, structure name or a label.

The identifier is a user-defined name of an entity to identify it uniquely during the program execution

Example

```
int marks;
char studentName[30];
```

Here, **marks** and **studentName** are identifiers.

Rules for Creating Identifiers

- An identifier can contain **letters** (UPPERCASE and lowercase), **numerics** & **underscore** symbol only.
- An identifier should not start with a numerical value. It can start with a letter or an underscore.
- We should not use any special symbols in between the identifier even whitespace. However, the only underscore symbol is allowed.
- Keywords should not be used as identifiers.

- There is no limit for the length of an identifier. However, the compiler considers the first 31 characters only.
- An identifier must be unique in its scope.

Rules for Creating Identifiers for better programming

The following are the commonly used rules for creating identifiers for better programming...

- The identifier must be meaningful to describe the entity.
- Since starting with an underscore may create conflict with system names, so we avoid starting an identifier with an underscore.
- We start every identifier with a lowercase letter. If an identifier contains more than one word then the first word starts with a lowercase letter and second word onwards first letter is used as an UPPERCASE letter. We can also use an underscore to separate multiple words in an identifier.

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.

It specifies the type of data that the variable can store like integer, character, floating, double, etc.

The data type is a collection of data with values having fixed values, meaning as well as its characteristics.

Types of Data Types in C Language

There are mainly three types of data types in C language -:

- Pre-defined data types
- Derived data types

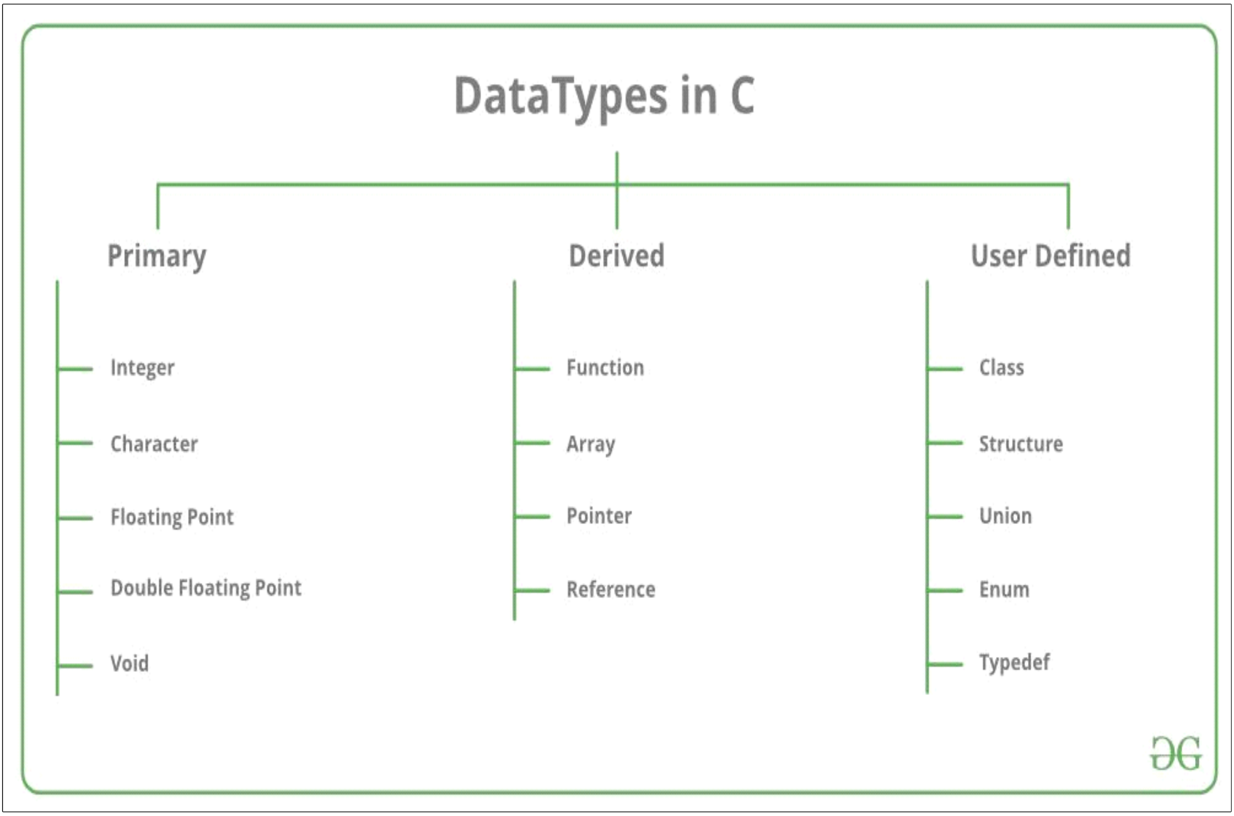
- User-defined data type

1. Pre-defined Data Types

Such data types are called Pre-defined Data Types which are already defined and don't need to be defined separately, such as int, char, float, double, void, etc. These data types come in Basic Data Types.

These data types, don't mean to tell the compiler separately because they are already defined. And the data types that are also keywords is called primitive data types.

Types	Description
Primitive Data Types	Arithmetic types can be further classified into integer and floating data types.
Void Types	The data type has no value or operator and it does not provide a result to its caller. But void comes under Primitive data types.
User Defined DataTypes	It is mainly used to assign names to integral constants, which make a program easy to read and maintain
Derived types	The data types that are derived from the primitive or built-in datatypes are referred to as Derived Data Types.



Different data types also have different ranges up to which they can store numbers. These ranges may vary from compiler to compiler. Below is a list of ranges along with the memory requirement and format specifiers on the *32-bit GCC compiler*.

Data Type	Memory (bytes)	Range	Format Specifier
short int	2	-32,768 to 32,767	%hd
unsigned short int	2	0 to 65,535	%hu
unsigned int	2	0 to 4,294,967,295	%u
int	2	-2,147,483,648 to 2,147,483,647	%d
long int	4	-2,147,483,648 to 2,147,483,647	%ld
unsigned long	4	0 to 4,294,967,295	%lu

int			
signed char	1	-128 to 127	%c
unsigned char	1	0 to 255	%c
float	4	1.2E-38 to 3.4E+38	%f
double	8	1.7E-308 to 1.7E+308	%lf
long double	16	3.4E-4932 to 1.1E+4932	%Lf

Integer Types

Integer Type

- The “int” keyword is used to create a variable of integer type.
- We store numeric values in the variable of Integer Type.
- Variables that are created of “int” data type can store 2 byte, 4 byte, and 8-byte data. But this thing depends on the compiler.
- Some compilers store 2-byte data in a variable made of int data type, while some store 4-byte data.
- int (2 byte) can store values from -32,768 to +32,767.
- int (4 byte) can store values from -2,147,483,648 to + 2,147,483,647.
- If you want to store more data value in variables then you can create a variable using the keyword “long int”. With the help of this keyword, the variable can store more amount of data than the normal variable.
- The ” %d ” Format Specifier is used to print the value of a variable made of int data type.

Note

- We don’t store decimal values in variables created with the help of int data type.
- If we store the decimal value in a variable created with int data type, then the decimal points of that value will be cut and only the whole number will be stored.

- If we want to store the decimal value then we have to use float and double data type.

Syntax -:

```
int x = 5;
long int y;
```

Note: The size of an integer data type is compiler-dependent, when processors are 16-bit systems, then it shows the output of int as 2 bytes.
And when processors are 32-bit then it shows 2 bytes as well as 4 bytes.

```
// C program to print Integer data types.
#include <stdio.h>
#include <conio.h>

void main()
{
   // Integer value with positive data.
   int a = 9;

   // integer value with negative data.
   int b = -9;

   // U or u is Used for Unsigned int in C.
   int c = 89U;

   // L or l is used for long int in C.
   long int d = 99998L;

   printf("Integer value with positive data: %d\n", a);
   printf("Integer value with negative data: %d\n", b);
   printf("Integer value with an unsigned int data: %u\n", c);
   printf("Integer value with an long int data: %ld", d);

   getch();
}
```

Output

```
Integer value with positive data: 9
Integer value with negative data: -9
Integer value with an unsigned int data: 89
Integer value with an long int data: 99998
```

```c
//Program :2
#include <stdio.h>
#include <conio.h>
void main ()
{
 int x = 5;
 long int y;
 printf ("Enter a Number \n");
 scanf ("%d", &y);
 printf ("value of x = %d, y = %d", x, y);

 getch();
}
```

Output -:

```
Enter a Number
10
value of x = 5, y = 10
```

Character Types

- We store character type data in a variable made with char data type.
- The “char” keyword is used to create a variable of character type.
- A variable created by the keyword “char” stores a single character.
- Variables made with Character Type have a size of 1 byte.
- The %c Format Specifier is used to print the value of a variable made of char data type.
  - Range: (-128 to 127) or (0 to 255)
  - Size: 1 byte
  - Format Specifier: %c

Syntax

```
char ch = 'A';
char ch2 = 'a';
```

```
// C program to print char data types.

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

void main()
{

   char a = 'b';

   printf("Value of a: %c\n", a);
          getch();

   }
```

Output

```
Value of a: b
```

Example:2

```
#include <stdio.h>
#include <conio.h>
void main ()
 {

 char ch = 'A';
 char ch2;
 printf ("Enter a Character \n");
 scanf ("%c", &ch2);
 printf ("value of ch = %c, ch2 = %c", ch, ch2);

getch();
}
```

Output -:

```
Enter a Character
B
value of ch = A, ch2 = B
```

Floating-Point Types

In C programming float data type is used to store floating-point values.

Float in C is used to store decimal and exponential values.

It is used to store decimal numbers (numbers with floating point values) with single precision.

- **Range:** 1.2E-38 to 3.4E+38
- **Size:** 4 bytes
- **Format Specifier:** %f

```
// C Program to demonstrate use
// of Floating types

#include <stdio.h>
#include <conio.h>
void main()
{

   float a = 9.0f;
   float b = 2.5f;

     // 2x10^-4
   float c = 2E-4f;
     printf("%f\n",a);
     printf("%f\n",b);
     printf("%f",c);

     getch();
}
```

Output

```
9.000000

2.500000

0.000200
```

// **Example of Float Data Type**

```
#include <stdio.h>
#include <conio.h>
void main ()
{
 float x = 5.50;
 float y;
 printf ("Enter a decimal Number \n");
 scanf ("%f", & y);
 printf ("sum of x and y is %f", x + y);
   getch();
}
```

Output -:

```
Enter a decimal number
10.06
Sum of x and y is 15.560000
```

Double Types

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. Double data type is basically a precision sort of data type that is capable of holding 64 bits of decimal numbers or floating points.

Since double has more precision as compared to that float then it is much more obvious that it occupies twice the memory as occupied by the floating-point type.

It can easily accommodate about 16 to 17 digits after or before a decimal point.

- **Range:** 1.7E-308 to 1.7E+308
- **Size:** 8 bytes
- **Format Specifier:** %lf

- The %lf Format Specifier is used to print the value of a variable made of double data type.

Example of Double Data Type

```
#include <stdio.h>
void main ()
{
 float x = 679999999.454;
 double y = 679999999.454;

 printf ("float x = %f and double y = %lf", x, y);
 getch();
}
```

Output

```
float x = 680000000.000000 and double y = 679999999.454000
```

```
// C Program to demonstrate
// use of double data type
#include <stdio.h>
 #include<conio.h>
void main()
{

    double a = 123123123.00;
    double b = 12.293123;
    double c = 2312312312.123123;

    printf("%lf\n", a);

    printf("%lf\n", b);

    printf("%lf", c);

    getch();
}
```

Output

```
123123123.000000

12.293123

2312312312.123123
```

Void Data types

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, and pointers to void.

Syntax:
// function return type void

```
// C program to demonstrate
// use of void pointers
#include <stdio.h>
 #include<conio.h>
void main()
{
   int val = 30;
   void *ptr = &val;
   printf("%d", *(int *)ptr);
   getch();
}
```

Output

```
30
```

We can use the sizeof() operator to check the size of a variable. See the following C program for the usage of the various data types:

```
// C Program to print size of
// different data type in C
#include <stdio.h>
#include<conio.h>

void main()
{
     int size_of_int=sizeof(int);
    int size_of_char= sizeof(char);
     int size_of_float=sizeof(float);
   int size_of_double=sizeof(double);

   printf("The size of int data type : %d\n",sizeof(int));
   printf("The size of char data type : %d\n",sizeof(char));
   printf("The size of float data type : %d\n",sizeof(float));
   printf("The size of double data type : %d",sizeof(double));

  getch();
}
```

Output

```
The size of int data type : 4

The size of char data type : 1

The size of float data type : 4

The size of double data type : 8
```

2. Enumeration data type in C :

Enumeration data type consists of named integer constants as a list.

It start with 0 (zero) by default and value is incremented by 1 for the sequential identifiers in the list.

Enum syntax in C :

```
enum identifier [optional {enumerator-list}];
```

Example :

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

3. Derived data types in C :

Those data types which are derived from fundamental data types are called derived data types.

There are basically three derived data types.

1. **Array** : A finite collection of data of same types or homogenous data type.

2. **String** : An array of character type.

3. **Structure** : A collection of related variables same or different data types.

4. Void data type in C:

- Void is an empty data type that has no value.
- This can be used in functions and pointers.

Operators:

Operators are symbols that help in performing operations of mathematical and logical nature. The classification of C operators are as follows:

- Arithmetic
- Relational
- Logical
- Bitwise
- Assignment
- Conditional
- Special

C Arithmetic Operators

An arithmetic operator performs mathematical operations such as addition, subtraction, multiplication, division etc on numerical values (constants and variables).

Operator	Meaning of Operator
+	addition or unary plus
-	subtraction or unary minus
*	Multiplication
/	Division
%	remainder after division (modulo division)

Example 1: Arithmetic Operators

```
// Working of arithmetic operators
#include <stdio.h>
#include<conio.h>
void main()
{
   int a = 9,b = 4, c;

   c = a+b;
   printf("a+b = %d \n",c);
   c = a-b;
   printf("a-b = %d \n",c);
   c = a*b;
   printf("a*b = %d \n",c);
   c = a/b;
   printf("a/b = %d \n",c);
   c = a%b;
   printf("Remainder when a divided by b = %d \n",c);
      getch();
}
```

Output

```
a+b = 13

a-b = 5

a*b = 36

a/b = 2

Remainder when a divided by b=1
```

The operators `+`, `-` and `*` computes addition, subtraction, and multiplication respectively as you might have expected.

In normal calculation, `9/4 = 2.25`. However, the output is `2` in the program.

It is because both the variables `a` and b are integers. Hence, the output is also an integer. The compiler neglects the term after the decimal point and shows answer 2 instead of 2.25.

The modulo operator % computes the remainder. When a=9 is divided by b=4, the remainder is 1. The % operator can only be used with integers.

C Increment and Decrement Operators

C programming has two operators increment ++ and decrement -- to change the value of an operand (constant or variable) by 1.

Increment ++ increases the value by 1 whereas decrement -- decreases the value by 1.

These two operators are unary operators, meaning they only operate on a single operand.

Example 2: Increment and Decrement Operators

```
// Working of increment and decrement operators
#include <stdio.h>
#include<conio.h>
void main()
{
   int a = 10, b = 100;
   float c = 10.5, d = 100.5;
   printf("++a = %d \n", ++a);
   printf("--b = %d \n", --b);
   printf("++c = %f \n", ++c);
   printf("--d = %f \n", --d);
   getch();
}
```

Output
```
++a = 11
--b = 99
++c = 11.500000
```

```
--d = 99.500000
```

Here, the operators `++` and `--` are used as prefixes. These two operators can also be used as postfixes like `a++` and `a--`.

3. Logical Operators

Logical Operators are used for True or False results.

The table below lists out the logical operators used in C

Operator	Function	Example (if a=1 and b=0)
&&	Logical AND	(a && b) is false
||	Logical OR	(a || b) is true
!	Logical NOT	(!a) is false

Example: C Program using logical operators.

```
#include <stdio.h>
#include<conio.h>
void main()
{
int a = 8, b = 8, c = 12, result;
result = (a == b) && (c > b);
printf("(a == b) && (c > b) equals to %d \n", result);
result = (a == b) && (c < b);
printf("(a == b) && (c < b) equals to %d \n", result);
```

```
result = (a == b) || (c < b);
printf("(a == b) || (c < b) equals to %d \n", result);
result = (a != b) || (c < b);
printf("(a != b) || (c < b) equals to %d \n", result);
result = !(a != b);
printf("!(a == b) equals to %d \n", result);
result = !(a == b);
printf("!(a == b) equals to %d \n", result);
getch();
```

Output :

```
(a == b) && (c > b) equals to 1

(a == b) && (c < b) equals to 0

(a == b) || (c < b) equals to 1


(a != b) || (c < b) equals to 0

!(a != b) equals to 1

!(a == b) equals to 0
```

2. Relational Operators

When we want to compare the values of two operands, then relational operators are used. If we want to check that is one operand is equal to or greater than other operands, then we use >= operator.

The below table lists of the relational operators in C with their functions.

Operator	Function	Example
==	This will check if two operands are equal	6 == 2 returns 0
!=	This will check if two operands are not equal.	6 != 2 returns 1
>	This will check if the operand on the left is greater than operand on the right	6 > 2 returns 1
<	This will check if the operand on the left is smaller than the right operand	6 < 2 returns 0
>=	This will check if the left operand is greater than or equal to the right operand	6 >= 2 returns 1
<=	This will check if operand on left is smaller than or equal to the right operand	6 <= 2 return 0

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

```
void main()
{
int a = 7, b = 7, c = 10;
printf("%d == %d = %d \n", a, b, a == b); // true
printf("%d == %d = %d \n", a, c, a == c); // false
printf("%d > %d = %d \n", a, b, a > b); //false
printf("%d > %d = %d \n", a, c, a > c); //false
printf("%d < %d = %d \n", a, b, a < b); //false
printf("%d < %d = %d \n", a, c, a < c); //true
printf("%d != %d = %d \n", a, b, a != b); //false
printf("%d != %d = %d \n", a, c, a != c); //true
printf("%d >= %d = %d \n", a, b, a >= b); //true
printf("%d >= %d = %d \n", a, c, a >= c); //false
printf("%d <= %d = %d \n", a, b, a <= b); //true
printf("%d <= %d = %d \n", a, c, a <= c); //true
getch();
}
```

Output:

```
7 == 7 = 1

7 == 10 = 0

7 > 7 = 0

7 > 10 = 0
```

7 < 7 = 0

7 < 10 = 1

7 != 7 = 0

7 != 10 = 1

7 >= 7 = 1

7 >= 10 = 0

7 <= 7 = 1

7 <= 10 = 1

Bitwise Operator in C

The bitwise operators are the operators used to perform the operations on the data at the bit-level. When we perform the bitwise operations, then it is also known as bit-level programming. It consists of two digits, either 0 or 1. It is mainly used in numerical computations to make the calculations faster.

We have different types of bitwise operators in the C programming language. The following is the list of the bitwise operators:

Operator	Meaning of operator
&	Bitwise AND operator
|	Bitwise OR operator
^	Bitwise exclusive OR operator
~	One's complement operator (unary operator)
<<	Left shift operator
>>	Right shift operator

Let's look at the truth table of the bitwise operators.

X	Y	X&Y	X|Y	X^Y
0	0	0	0	0
0	1	0	1	1
1	0	0	1	1
1	1	1	1	1

Bitwise AND operator

Bitwise AND operator is denoted by the single ampersand sign (&). Two integer operands are written on both sides of the (&) operator. If the corresponding bits of both the operands are 1, then the output of the bitwise AND operation is 1; otherwise, the output would be 0.

For example,

- We have two variables a and b.
- a =6;
- b=4;
- The binary representation of the above two variables are given below:
- a = 0110
- b = 0100
- When we apply the bitwise AND operation in the above two variables, i. e., a&b, the output would be:
- Result = 0100

As we can observe from the above result that bits of both the variables are compared one by one. If the bit of both the variables is 1 then the output would be 1, otherwise 0.

Let's understand the bitwise AND operator through the program.

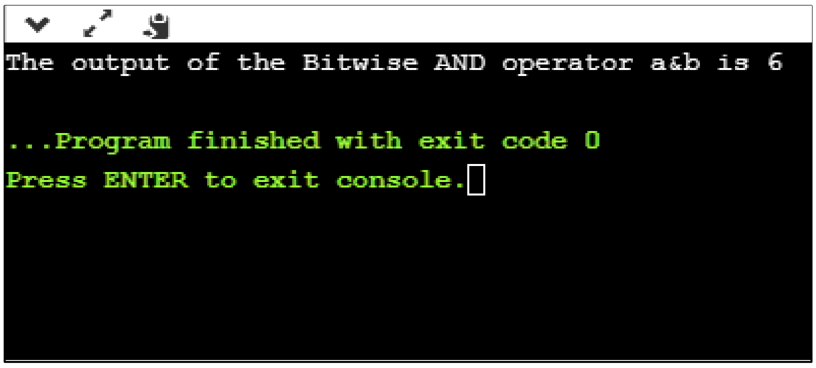
- #include <stdio.h>
- #include<conio.h>
- void main()

- {
- int a=6, b=14; // variable declarations
- printf("The output of the Bitwise AND operator a&b is %d",a&b);
- getch();
- }

In the above code, we have created two variables, i.e., 'a' and 'b'. The values of 'a' and 'b' are 6 and 14 respectively. The binary value of 'a' and 'b' are 0110 and 1110, respectively. When we apply the AND operator between these two variables,

a AND b = 0110 && 1110 = 0110

Output



```
The output of the Bitwise AND operator a&b is 6

...Program finished with exit code 0
Press ENTER to exit console.
```

Bitwise OR operator

The bitwise OR operator is represented by a single vertical sign (|). Two integer operands are written on both sides of the (|) symbol. If the bit value of any of the operand is 1, then the output would be 1, otherwise 0.

For example,

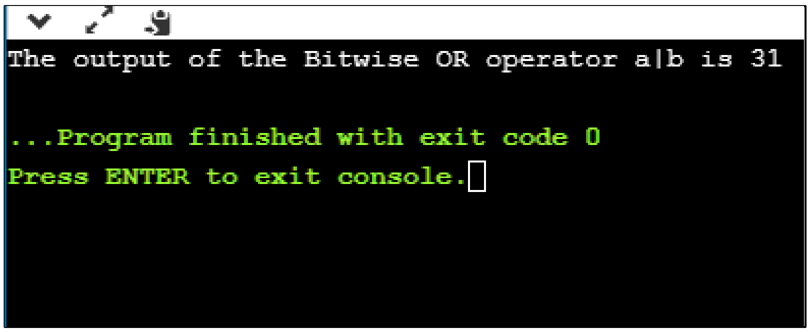
- We consider two variables,
- a = 23;
- b = 10;
- The binary representation of the above two variables would be:
- a = 0001 0111
- b = 0000 1010
- When we apply the bitwise OR operator in the above two variables, i.e., a|b , then the output would be:
- Result = 0001 1111

As we can observe from the above result that the bits of both the operands are compared one by one; if the value of either bit is 1, then the output would be 1 otherwise 0.

Let's understand the bitwise OR operator through a program.

- #include <stdio.h>
- #include<conio.h>
- void main()
- {
-   int a=23,b=10; // variable declarations
-   printf("The output of the Bitwise OR operator a|b is %d",a|b);
-   getch();
- }

Output



```
The output of the Bitwise OR operator a|b is 31

...Program finished with exit code 0
Press ENTER to exit console.
```

Bitwise exclusive OR operator

Bitwise exclusive OR operator is denoted by (^) symbol. Two operands are written on both sides of the exclusive OR operator. If the corresponding bit of any of the operand is 1 then the output would be 1, otherwise 0.

For example,

- We consider two variables a and b,
- a = 12;
- b = 10;
- The binary representation of the above two variables would be:
- a = 0000 1100
- b = 0000 1010
- When we apply the bitwise exclusive OR operator in the above two vari ables (a^b), then the result would be:
- Result = 0000 1110

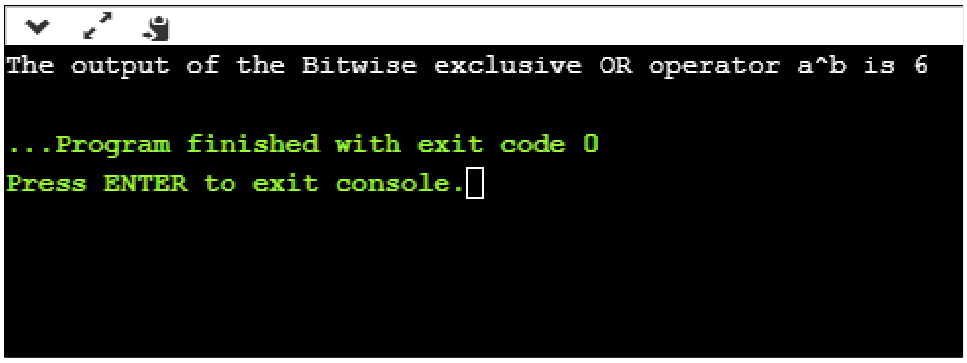
As we can observe from the above result that the bits of both the operands are compared one by one; if the corresponding bit value of any of the operand is 1, then the output would be 1 otherwise 0.

Let's understand the bitwise exclusive OR operator through a program.

```
• #include <stdio.h>
• #include<conio.h>
• void main()
• {
•   int a=12,b=10;  // variable declarations
•   printf("The output of the Bitwise exclusive OR operator a^b is %d",a^b);

•   getch();
• }
```

Output



```
The output of the Bitwise exclusive OR operator a^b is 6

...Program finished with exit code 0
Press ENTER to exit console.
```

Bitwise complement operator

The bitwise complement operator is also known as one's complement operator. It is represented by the symbol tilde (~). It takes only one operand or variable and performs complement operation on an operand. When we apply the complement operation on any bits, then 0 becomes 1 and 1 becomes 0.

For example,

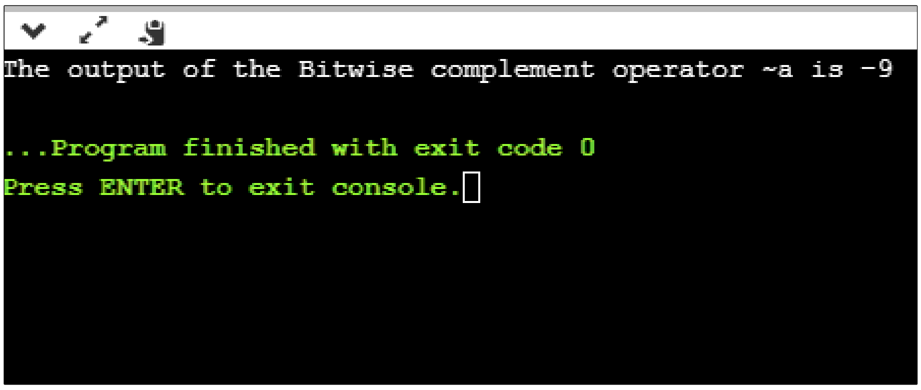
- If we have a variable named 'a',
- a = 8;
- The binary representation of the above variable is given below:
- a = 1000
- When we apply the bitwise complement operator to the operand, then t he output would be:
- Result = 0111

As we can observe from the above result that if the bit is 1, then it gets

changed to 0 else 1.
Let's understand the complement operator through a program.

- #include <stdio.h>
- #include<conio.h>
- void main()
- {
-   int a=8; // variable declarations
-   printf("The output of the Bitwise complement operator ~a is %d",~a);
-   getch();
- }

Output



```
The output of the Bitwise complement operator ~a is -9

...Program finished with exit code 0
Press ENTER to exit console.
```

Bitwise shift operators

Two types of bitwise shift operators exist in C programming. The bitwise shift operators will shift the bits either on the left-side or right-side. Therefore, we can say that the bitwise shift operator is divided into two categories:

- Left-shift operator
- Right-shift operator

Left-shift operator

It is an operator that shifts the number of bits to the left-side.
Syntax of the left-shift operator is given below:

- Operand << n

- Right-shift operator
  It is an operator that shifts the number of bits to the left-side.

- Syntax of the Right-shift operator is given below:
- Operand >> n

Program for Bitwise Operator in C

Let us now take a look at the program using all the bitwise operators.

```
#include <stdio.h>
int main()
 {
unsigned char x = 20, y = 21; // x = 20 (00010100), y = 21
(00010101)
int g = 0;
   g = x & y; /* 20 = 010100 */
   printf(" The result of Bitwise AND is %d \n", g );
   g = x | y; /* 21 = 010101 */
   printf(" The result of Bitwise OR is %d \n", g );
   g = x ^ y; /* 1 = 0001 */
   printf(" The result of Bitwise XOR is %d \n", g );
   g = ~x;
   printf(" The result of Bitwise NOT is %d \n", g );
   g = x << 1;
   printf(" The result of Bitwise Left Shift is %d \n", g );
   g = x >> 1;
   printf(" The result of Bitwise Right Shift is %d \n", g );
    getch();
}
```

OUTPUT:

```
The result of Bitwise AND is 20
 The result of Bitwise OR is 21
 The result of Bitwise XOR is 1
 The result of Bitwise NOT is -21
 The result of Bitwise Left Shift is 40
 The result of Bitwise Right Shift is 10
```

6. Assignment Operators

These types of operators are used to assign a value to a variable.

Operator	Function	Example
=	This will assign values from right side operands to left side operand	a=b
+=	This will add the right operand to the left operand and assign the result to left	a+=b is same as a=a+b
-=	This will subtract the right operand from the left operand and assign the result to the left operand	a-=b is same as a=a-b
*=	This will multiply the left operand with the right operand and assign the result to the left operand	a*=b is same as a=a*b
/=	This will divide the left operand with the right operand and assign the result to the left operand	a/=b is same as a=a/b
%=	This will calculate modulus using two operands and assign the result to the left operand	a%=b is the same as a=a%b

Conditional Operator in C

The conditional operator is also known as a **ternary operator**. The conditional statements are the decision-making statements which depends upon the output of the expression. It is represented by two symbols, i.e., '?' and ':'.

As conditional operator works on three operands, so it is also known as the ternary operator.

The behavior of the conditional operator is similar to the 'if-else' statement as 'if-else' statement is also a decision-making statement.

Syntax of a conditional operator

- Expression1? expression2: expression3;

```
•   #include <stdio.h>
•   #include <conio.h>
•   void main()
•   {
•      int age;  // variable declaration
•      printf("Enter your age");
•      scanf("%d",&age);   // taking user input for age variable
•      (age>=18)?
•      printf("eligible for voting") :
•      printf("not eligible for voting");  // conditional operator
•      getch();
•    }
```

In the above code, we are taking input as the 'age' of the user. After taking input, we have applied the condition by using a conditional operator.

In this condition, we are checking the age of the user. If the age of the user is greater than or equal to 18, then the statement1 will execute, i.e., (printf("eligible for voting")) otherwise, statement2 will execute, i.e., (printf("not eligible for voting")).

8. Special Operators

Here are some special operators used in C

Operator	Function
&	This operator is used to get the address of the variable. **Example:** &a will give an address of a.
*	This operator is used as a pointer to a variable. **Example:** * a where * is a pointer to the variable a.
size of ()	This operator gives the size of the variable.

	Example: The size of (char) will give us 1.

Example: C program using a special operator

```
#include <stdio.h>

#include <conio.h>

void main()

{


int *ptr, q;

q = 40;

/* address of q is assigned to ptr */

ptr = &q;

/* display q's value using ptr variable */

printf("%d", *ptr);

getch();
}
```

Output: 40

The sizeof operator

The `sizeof` is a unary operator that returns the size of data (constants, variables, array, structure, etc).

Example 6: sizeof Operator

```
#include <stdio.h>
#include <conio.h>
void main()
{
   int a;
   float b;
   double c;
   char d;
   printf("Size of int=%lu bytes\n",sizeof(a));
   printf("Size of float=%lu bytes\n",sizeof(b));
   printf("Size of double=%lu bytes\n",sizeof(c));
   printf("Size of char=%lu byte\n",sizeof(d));
getch();
}
```

Output

```
Size of int = 4 bytes
Size of float = 4 bytes
Size of double = 8 bytes
Size of char = 1 byte
```

Precedence & Associativity In C Language

Precedence : Precedence is nothing but priority that indicates which operator has to be evaluated first when there is more than one operator.

Associativity : when there is more than one operator with same precedence [priority] then we consider associativity, which indicated the order in which the expression has to be evaluated. It may be either from Left to Right or Right to Left.

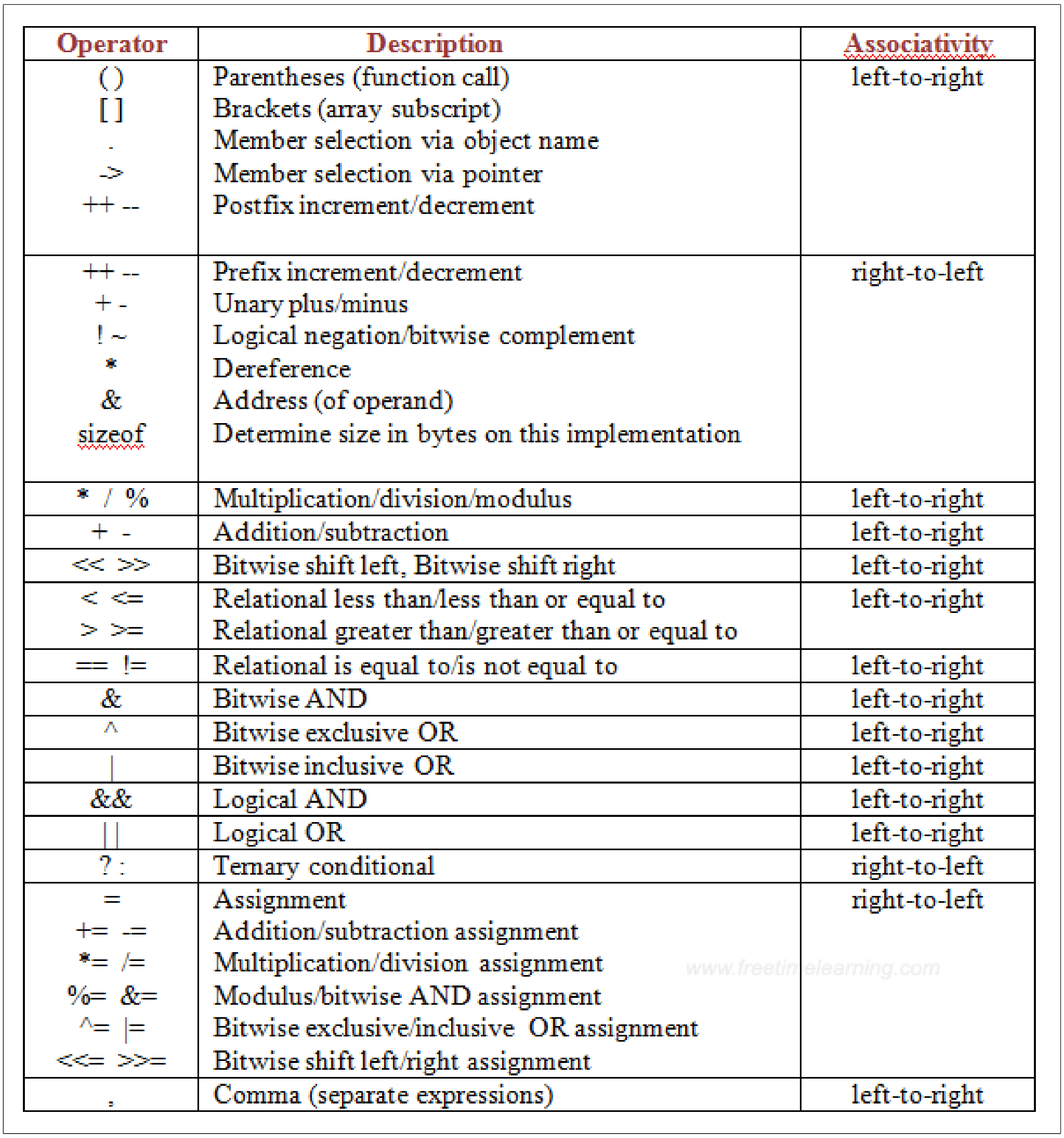
Example :

```
x = 7 + 3 * 2;
```

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

$$\boxed{7 + \underbrace{3*2}_{1} \quad = \quad \underbrace{7+6}_{2} \quad = 13}$$

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



Operator	Description	Associativity
( ) [ ] . -> ++ --	Parentheses (function call) Brackets (array subscript) Member selection via object name Member selection via pointer Postfix increment/decrement	left-to-right
++ -- + - ! ~ * & sizeof	Prefix increment/decrement Unary plus/minus Logical negation/bitwise complement Dereference Address (of operand) Determine size in bytes on this implementation	right-to-left
* / %	Multiplication/division/modulus	left-to-right
+ -	Addition/subtraction	left-to-right
<< >>	Bitwise shift left, Bitwise shift right	left-to-right
< <= > >=	Relational less than/less than or equal to Relational greater than/greater than or equal to	left-to-right
== !=	Relational is equal to/is not equal to	left-to-right
&	Bitwise AND	left-to-right
^	Bitwise exclusive OR	left-to-right
|	Bitwise inclusive OR	left-to-right
&&	Logical AND	left-to-right
||	Logical OR	left-to-right
? :	Ternary conditional	right-to-left
= += -= *= /= %= &= ^= |= <<= >>=	Assignment Addition/subtraction assignment Multiplication/division assignment Modulus/bitwise AND assignment Bitwise exclusive/inclusive OR assignment Bitwise shift left/right assignment	right-to-left
,	Comma (separate expressions)	left-to-right

Note : Unary, Conditional & Assignment operators are evaluated from Right to Left, Remaining operators are from Left to Right.

Printf & scanf in C Language

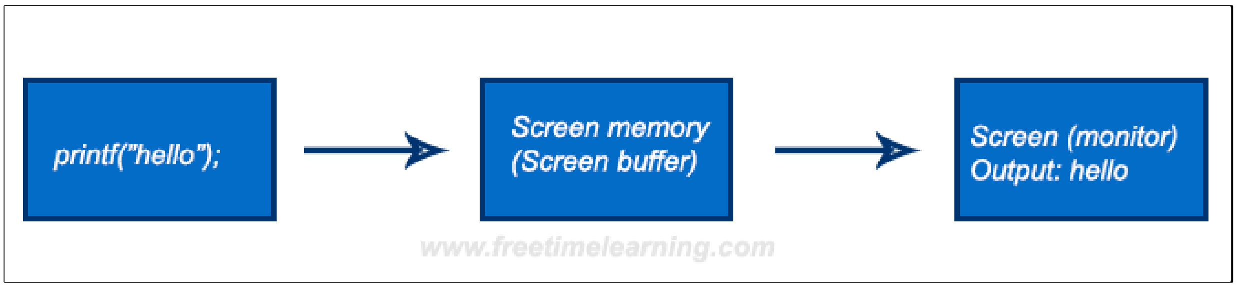
The **printf()** and **scanf()** functions are used for input and output in C language.

Both functions are inbuilt library functions, defined in stdio.h (header file).

printf() function :

The **printf()** function is used for output. It prints the given statement to the console.

The general form of **printf()** function is given below :



Syntax :

```
printf("format string", list of variables);
```

Example :

```
printf("%d %d",a,b);
```

The formatted string represents, what format we want to display on the screen with the list of variables. The format string can be following table :

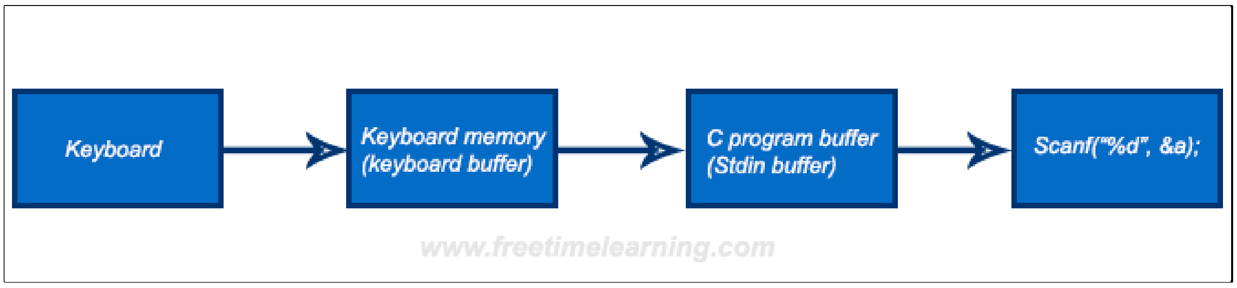
Data Type	Conversion	format string
Integer	short integer	%d or % i
	short unsigned	% u
	long signed	% ld
	long unsigned	% lu
	unsigned hexadecimal	% x
	unsigned octal	%O
Real	float	% f or % g
	double	%lf
	signed character	%c
	unsigned char	%c

	string	%s

scanf() function :

This function is used to read values using keyboard. It is used for runtime assignment of variables.

The general form of scanf( ) is :


Keyboard
Keyboard memory (keyboard buffer)
C program buffer (Stdin buffer)
Scanf("%d", &a);
www.freetimelearning.com

Syntax :

```
scanf("format string", list_of_addresses_of_Variables );
```

The format string contains - Conversion specifications that begin with % sign

Example :

```
scanf("%d %f %c", &a, &b, &c);
```

" & " is called the address operator, in **scanf( )** the " & " operator indicates the memory location of the variable. So that the value read would be placed at that location.

Program : Let's see a simple example of C language that gets input from the user and prints the square of the given number.

```
#include<stdio.h>
#include<conio.h>
void main()
{
int number;

clrscr();
printf("enter a number:");
scanf("%d",&number);
printf("square of number is:%d ",number*number);
```

```
 getch();
}
```

Output :

```
enter a number: 5
square of number is: 25
```

The **scanf("%d", &number)** statement reads integer number from the console and stores the given value in number variable.

The **printf("square of number is:%d ",number*number)** statement prints the square of number on the console.

Program : Let's see a simple example of input and output in C language that prints addition of 2 numbers.

```
#include<stdio.h>
#include<conio.h>
void main()
{
int x=0,y=0,result=0;
clrscr();
printf(“Enter first number : ”);
scanf(“%d”,&x);
printf(“Enter second number : ”);
scanf(“%d”,&y);
result=x+y;
printf(“Sum of 2 numbers : %d “,result);
getch();
}
```

Output :

```
Enter first number : 9
Enter second number : 27
Sum of 2 numbers : 36
```

Formatted I/O Functions

Formatted I/O functions are used to take various inputs from the user and display multiple outputs to the user.

These types of I/O functions can help to display the output to the user in different formats using the format specifiers.

These I/O supports all data types like int, float, char, and many more.
Why they are called formatted I/O?

These functions are called formatted I/O functions because we can use format specifiers in these functions and hence, we can format these functions according to our needs.

List of some format specifiers-

S NO.	Format Specifier	Type	Description
1	%d	int/signed int	used for I/O signed integer value
2	%c	char	Used for I/O character value
3	%f	float	Used for I/O decimal floating-point value
4	%s	string	Used for I/O string/group of characters
5	%ld	long int	Used for I/O long signed integer value
6	%u	unsigned int	Used for I/O unsigned integer value
7	%i	unsigned int	used for the I/O integer value
8	%lf	double	Used for I/O fractional or floating data
9	%n	prints	prints nothing

The following formatted I/O functions will be discussed in this section-

- printf()
- scanf()
- sprintf()
- sscanf()

printf():

printf HYPERLINK "https://www.geeksforgeeks.org/return-values-of-printf-and-scanf-in-c-cpp/" HYPERLINK "https://www.geeksforgeeks.org/return-values-of-printf-and-scanf-in-c-cpp/" HYPERLINK "https://www.geeksforgeeks.org/return-values-of-printf-and-scanf-in-c-cpp/" HYPERLINK "https://www.geeksforgeeks.org/return-values-of-printf-and-scanf-in-c-cpp/" HYPERLINK "https://www.geeksforgeeks.org/return-values-of-printf-and-scanf-in-c-cpp/" HYPERLINK "https://www.geeksforgeeks.org/return-values-of-printf-and-scanf-in-c-cpp/" HYPERLINK "https://www.geeksforgeeks.org/return-values-of-printf-and-scanf-in-c-cpp/"(HYPERLINK "https://www.geeksforgeeks.org/return-values-of-printf-and-scanf-in-c-cpp/" HYPERLINK "https://www.geeksforgeeks.org/return-values-of-printf-and-scanf-in-c-cpp/" HYPERLINK "https://www.geeksforgeeks.org/return-values-of-printf-and-scanf-in-c-cpp/" HYPERLINK "https://www.geeksforgeeks.org/return-values-of-printf-and-scanf-in-c-cpp/" HYPERLINK "https://www.geeksforgeeks.org/return-values-of-printf-and-scanf-in-c-cpp/" HYPERLINK "https://www.geeksforgeeks.org/return-values-of-printf-and-scanf-in-c-cpp/" HYPERLINK "https://www.geeksforgeeks.org/return-values-of-printf-and-scanf-in-c-cpp/") function is used in a C program to display any value like float, integer, character, string, etc on the console screen.

It is a pre-defined function that is already declared in the stdio.h(header file).

Syntax 1:

To display any variable value.

printf("Format Specifier", var1, var2, …., varn);

Example:

-

```
// C program to implement
// printf() function
#include <stdio.h>

// Driver code
void main()
{
   // Declaring an int type variable
```

```
    int a;

    // Assigning a value in a variable
    a = 20;

    // Printing the value of a variable
    printf("%d", a);

    getch();
}
```

Output

```
20
```

Syntax 2:
To display any string or a message

printf("Enter the text which you want to display");
Example:

Example:

- C

```
// C program to implement
// printf() function
#include <stdio.h>
#include<conio.h>

// Driver code
void main()
{
    // Displays the string written
    // inside the double quotes
    printf("This is a string");
    getch();
}
```

Output

```
This is a string
```

scanf():

scanf HYPERLINK "https://www.geeksforgeeks.org/scanf-and-fscanf-in-c/" HYPERLINK "https://www.geeksforgeeks.org/scanf-and-fscanf-in-c/" HYPERLINK "https://www.geeksforgeeks.org/scanf-and-fscanf-in-c/" HYPERLINK "https://www.geeksforgeeks.org/scanf-and-fscanf-in-c/" HYPERLINK "https://www.geeksforgeeks.org/scanf-and-fscanf-in-c/" HYPERLINK "https://www.geeksforgeeks.org/scanf-and-fscanf-in-c/" HYPERLINK "https://www.geeksforgeeks.org/scanf-and-fscanf-in-c/"(HYPERLINK "https://www.geeksforgeeks.org/scanf-and-fscanf-in-c/" HYPERLINK "https://www.geeksforgeeks.org/scanf-and-fscanf-in-c/" HYPERLINK "https://www.geeksforgeeks.org/scanf-and-fscanf-in-c/" HYPERLINK "https://www.geeksforgeeks.org/scanf-and-fscanf-in-c/" HYPERLINK "https://www.geeksforgeeks.org/scanf-and-fscanf-in-c/" HYPERLINK "https://www.geeksforgeeks.org/scanf-and-fscanf-in-c/" HYPERLINK "https://www.geeksforgeeks.org/scanf-and-fscanf-in-c/") function is used in the C program for reading or taking any value from the keyboard by the user, these values can be of any data type like integer, float, character, string, and many more.

This function is declared in stdio.h(header file), that's why it is also a pre-defined function.

In scanf() function we use &(address-of operator) which is used to store the variable value on the memory location of that variable.

Syntax:

scanf("Format Specifier", &var1, &var2, …., &varn);

Example:

- C

```
// C program to implement
// scanf() function
#include <stdio.h>

// Driver code
Void main()
{
   int num1;

   // Printing a message on
   // the output screen
   printf("Enter a integer number: ");

   // Taking an integer value
   // from keyboard
   scanf("%d", &num1);

   // Displaying the entered value
```

```
    printf("You have entered %d", num1);

    getch();
}
```

Output

```
Enter a integer number: You have entered 0
```

Output:

```
Enter a integer number: 56

You have entered 56
```

sprintf():

sprintf stands for **"string print"**.

This function is similar to printf() function but this function prints the string into a character array instead of printing it on the console screen.

Syntax:

sprintf(array_name, "format specifier", variable_name);

Example:

- C

```
// C program to implement
// the sprintf() function
#include <stdio.h>

// Driver code
void main()
{
    char str[50];
    int a = 2, b = 8;

    // The string "2 and 8 are even number"
    // is now stored into str
    sprintf(str, "%d and %d are even number",
            a, b);

    // Displays the string
    printf("%s", str);
    getch();
}
```

Output

```
2 and 8 are even number
```

sscanf():

sscanf stands for “string scanf”.

This function is similar to scanf() function but this function reads data from the string or character array instead of the console screen.

Syntax:

sscanf(array_name, “format specifier”, &variable_name);

Example:

```c
// C program to implement
// sscanf() function
#include <stdio.h>

// Driver code
void main()
{
   char str[50];
   int a = 2, b = 8, c, d;

   // The string "a = 2 and b = 8"
   // is now stored into str
   // character array
   sprintf(str, "a = %d and b = %d",
           a, b);

   // The value of a and b is now in
   // c and d
   sscanf(str, "a = %d and b = %d",
          &c, &d);

   // Displays the value of c and d
   printf("c = %d and d = %d", c, d);
   getch();
}
```

Output

```
c = 2 and d = 8
```

Unformatted Input/Output functions

Unformatted I/O functions are used only for character data type or character array/string and cannot be used for any other datatype.

These functions are used to read single input from the user at the console and it allows to display the value at the console.

Why they are called unformatted I/O?
These functions are called unformatted I/O functions because we cannot use format specifiers in these functions and hence, cannot format these functions according to our needs.

The following unformatted I/O functions will be discussed in this section-

- getch()
- getche()
- getchar()
- putchar()
- gets()
- puts()
- putch()

getch():

getch HYPERLINK "https://www.geeksforgeeks.org/getch-function-in-c-with-examples/" HYPERLINK "https://www.geeksforgeeks.org/getch-function-in-c-with-examples/" HYPERLINK "https://www.geeksforgeeks.org/getch-function-in-c-with-examples/" HYPERLINK "https://www.geeksforgeeks.org/getch-function-in-c-with-examples/" HYPERLINK "https://www.geeksforgeeks.org/getch-function-in-c-with-examples/" HYPERLINK "https://www.geeksforgeeks.org/getch-function-in-c-with-examples/" HYPERLINK "https://www.geeksforgeeks.org/getch-function-in-c-with-examples/"(HYPERLINK "https://www.geeksforgeeks.org/getch-function-in-c-with-examples/" HYPERLINK "https://www.geeksforgeeks.org/getch-function-in-c-with-examples/" HYPERLINK "https://www.geeksforgeeks.org/getch-function-in-c-with-examples/" HYPERLINK "https://www.geeksforgeeks.org/getch-function-in-c-with-examples/" HYPERLINK "https://www.geeksforgeeks.org/getch-function-in-c-with-examples/" HYPERLINK "https://www.geeksforgeeks.org/getch-function-in-c-with-examples/" HYPERLINK "https://www.geeksforgeeks.org/getch-function-in-c-with-examples/") function reads a single character from the keyboard by the user but doesn't display that character on the console screen and immediately returned without pressing enter key. This function is declared in conio.h(header file). getch() is also used for hold the screen.

Syntax:

getch();
or
variable-name = getch();

Example:

- C

```
// C program to implement
// getch() function
#include <conio.h>
#include <stdio.h>

// Driver code
void main()
{
   printf("Enter any character: ");

   // Reads a character but
   // not displays
   getch();
       getch();
}
```

Output:
Enter any character:

getche():

getche HYPERLINK "https://www.geeksforgeeks.org/difference-getchar-getch-getc-getche/" HYPERLINK "https://www.geeksforgeeks.org/difference-getchar-getch-getc-getche/" HYPERLINK "https://www.geeksforgeeks.org/difference-getchar-getch-getc-getche/" HYPERLINK "https://www.geeksforgeeks.org/difference-getchar-getch-getc-getche/" HYPERLINK "https://www.geeksforgeeks.org/difference-getchar-getch-getc-getche/" HYPERLINK "https://www.geeksforgeeks.org/difference-getchar-getch-getc-getche/" HYPERLINK "https://www.geeksforgeeks.org/difference-getchar-getch-getc-getche/"(HYPERLINK

"https://www.geeksforgeeks.org/difference-getchar-getch-getc-getche/" HYPERLINK "https://www.geeksforgeeks.org/difference-getchar-getch-getc-getche/" HYPERLINK "https://www.geeksforgeeks.org/difference-getchar-getch-getc-getche/" HYPERLINK "https://www.geeksforgeeks.org/difference-getchar-getch-getc-getche/" HYPERLINK "https://www.geeksforgeeks.org/difference-getchar-getch-getc-getche/" HYPERLINK "https://www.geeksforgeeks.org/difference-getchar-getch-getc-getche/" HYPERLINK "https://www.geeksforgeeks.org/difference-getchar-getch-getc-getche/") function reads a single character from the keyboard by the user and displays it on the console screen and immediately returns without pressing the enter key.
This function is declared in conio.h(header file).

Syntax:

getche();

or
variable_name = getche();

Example:

- C

```
// C program to implement
// the getche() function
#include <conio.h>
#include <stdio.h>

// Driver code
 void main()
{
   printf("Enter any character: ");

   // Reads a character and
   // displays immediately
   getche();
   getch();
}
```

Output:
Enter any character: g

getchar():

The getchar() function is used to read only a first single character from

the keyboard whether multiple characters is typed by the user and this function reads one character at one time until and unless the enter key is pressed.
This function is declared in stdio.h(header file)

Syntax:

Variable-name = getchar();

Example:

```
// C program to implement
// the getchar() function
#include <conio.h>
#include <stdio.h>

// Driver code
int main()
{
    // Declaring a char type variable
    char ch;

    printf("Enter the character: ");

    // Taking a character from keyboard
    ch = getchar();

    // Displays the value of ch
    printf("%c", ch);
    getch();
}
```

Output:
Enter the character: a

a

putchar():

The putchar() function is used to display a single character at a time by passing that character directly to it or by passing a variable that has already stored a character.
 This function is declared in stdio.h(header file)

Syntax:

putchar(variable_name);

Example:

- C

```
// C program to implement
// the putchar() function
#include <conio.h>
#include <stdio.h>

// Driver code
void main()
{
    char ch;
    printf("Enter any character: ");

    // Reads a character
    ch = getchar();

    // Displays that character
    putchar(ch);
    getch();
}
```

Output:

```
Enter any character: Z

Z
```

gets():

gets() function reads a group of characters or strings from the keyboard by the user and these characters get stored in a character array.

This function allows us to write space-separated texts or strings.

This function is declared in stdio.h(header file).

Syntax:

char str[length of string in number]; //Declare a char type variable of any length
gets(str);

Example:

```
// C program to implement

// the gets() function
#include <conio.h>
```

```
#include <stdio.h>

// Driver code
void main()
{
   // Declaring a char type array
   // of length 50 characters
   char name[50];

   printf("Please enter some texts: ");

   // Reading a line of character or
   // a string
   gets(name);

   // Displaying this line of character
   // or a string
   getch();
}
```
Output:
Please enter some texts: geeks for geeks

You have entered: geeks for geeks

puts():

In C programming puts(HYPERLINK "https://www.geeksforgeeks.org/puts-vs-printf-for-printing-a-string/" HYPERLINK "https://www.geeksforgeeks.org/puts-vs-printf-for-printing-a-string/" HYPERLINK "https://www.geeksforgeeks.org/puts-vs-printf-for-printing-a-string/" HYPERLINK "https://www.geeksforgeeks.org/puts-vs-printf-for-printing-a-string/" HYPERLINK "https://www.geeksforgeeks.org/puts-vs-printf-for-printing-a-string/" HYPERLINK "https://www.geeksforgeeks.org/puts-vs-printf-for-printing-a-string/" HYPERLINK "https://www.geeksforgeeks.org/puts-vs-printf-for-printing-a-string/") function is used to display a group of characters or strings which is already stored in a character array.
This function is declared in stdio.h(header file).

Syntax:

puts(identifier_name);

Example:

-

```
// C program to implement
// the puts() function

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

// Driver code
void main()
{
   char name[50];
   printf("Enter your text: ");

   // Reads string from user
   gets(name);


   // Displays string
   puts(name);

   getch();
}
```

Output:

Enter your text: GeeksforGeeks

Your text is: GeeksforGeeks

putch():

putch() function is used to display a single character which is given by the user and that character prints at the current cursor location.

This function is declared in conio.h(header file)

Syntax:

putch(variable_name);

Example:

```
// C program to implement
// the putch() functions
#include <conio.h>

#include <stdio.h>
```

```
// Driver code
void main()
{
    char ch;
    printf("Enter any character:\n ");

    // Reads a character from the keyboard
    ch = getch();


    // Displays that character on the console
    putch(ch);
    getch();
}
```

Output:

```
Enter any character:

Entered character is: d
```