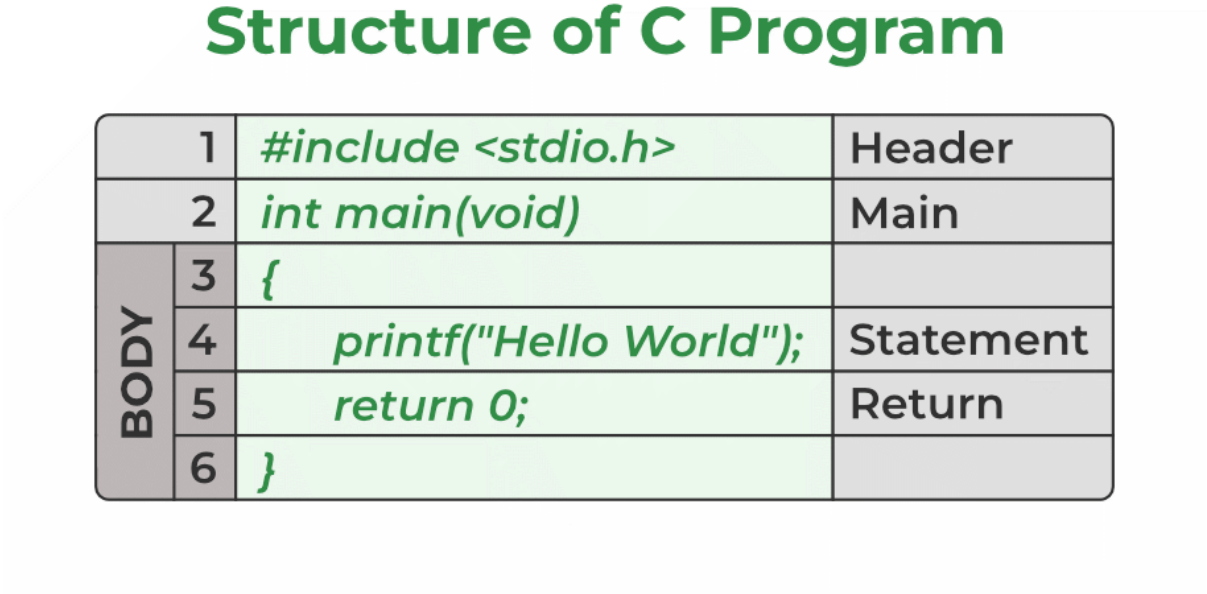



## 1. Header Files Inclusion – Line 1 [#include <stdio.h>]

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

Some of the C Header files:

stddef.h – Defines several useful types and macros.

stdint.h – Defines exact width integer types.

stdio.h – Defines core input and output functions

stdlib.h – Defines numeric conversion functions, pseudo-random number generator, and memory allocation

string.h – Defines string handling functions

math.h – Defines common mathematical functions.

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

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

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

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

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

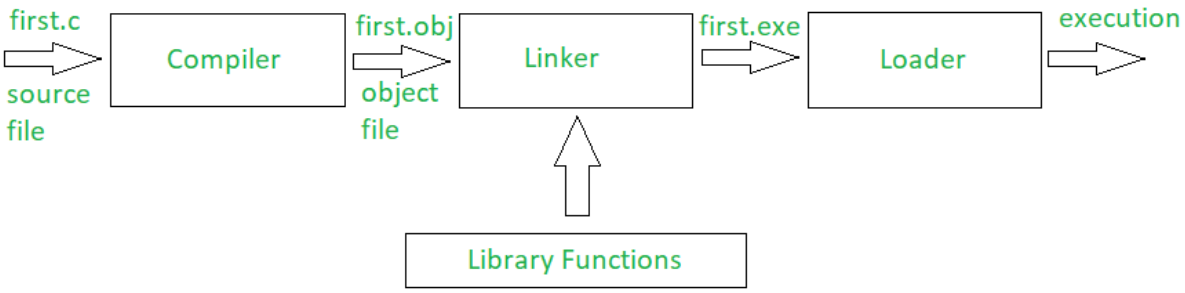
Statements are the instructions given to the compiler. In C, a statement is always terminated by a semicolon (;). In this particular case, we use printf() function to instruct the compiler to display "Hello World" text on the screen.

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

The last part of any C function is the return statement. The return statement refers to the return values from a function. This return statement and return value depend upon the return type of the function. The return statement in our program returns the value from main(). The returned value may be used by an operating

system to know the termination status of your program. The value 0 typically means successful termination.

## How does a C program execute?




Every file that contains a C program must be saved with '.c' extension. This is necessary for the compiler to understand that this is a C program file. Suppose a program file is named, first.c.

The file first.c is called the source file which keeps the code of the program. Now, when we compile the file, the C compiler looks for errors. If the C compiler reports no error, then it stores the file as a .obj file of the same name, called the object file. So, here it will create the first.obj. This .obj file is not executable.

The process is continued by the Linker which finally gives a .exe file which is executable.

**Linker:** First of all, let us know that library functions are not a part of any C program but of the C software. Thus, the compiler doesn't know the operation of any function, whether it be printf or scanf. **The definitions of these functions are stored in their**

**respective library which the compiler should be able to link.** This is what the Linker does. So, when we write #include, it includes stdio.h library which gives access to Standard Input and Output. The linker links the object files to the library functions and the program becomes a .exe file. Here, first.exe will be created which is in an executable format.

**Loader:** Whenever we give the command to execute a particular program, the loader comes into work. The loader will load the .exe file in RAM and inform the CPU with the starting point of the address where this program is loaded.

# Beginning with C programming:

## Writing the First Program in C

The following code is one of the simplest C programs that will help us the basic syntax structure of a C program.

```
#include <stdio.h>

int main() {

int a = 10;

printf("%d", a);

return 0;

}
```

**Output**

10

```
// Simple C program to display "Hello World"


// Header file for input output functions
```

```
#include <stdio.h>


// main function -
// where the execution of program begins
int main()
{


    // prints hello world
    printf("Hello World");


    return 0;
}
```

**Output**
Hello World

# Explanation of the Code

Let us now understand the terminologies of the above program:

## Line 1:

// Simple C program to display “Hello World”

1. This is a single comment line. A comment is used to display additional information about the program.
2. A comment does not contain any programming logic as it is not read by the compiler. When a comment is encountered by a compiler, the compiler simply skips that line of code.
3. Any line beginning with ‘//’ without quotes OR in between /*…*/ in C is a comment.

**More on Comments in C**

## Line 3:

#include

1. In C, all lines that start with the pound (#) sign are called directives. These statements are processed by preprocessor program invoked by the compiler.
2. The **#include** directive tells the compiler to include a file and **#include<stdio.h>** tells the compiler to include the header file for the Standard Input Output file which contains declarations of all the standard input/output library functions.

**More on Preprocessors in C.**

## Line 6:

int main()

1. This line is used to declare a function named "main" which returns data of integer type. A function is a group of statements that are designed to perform a specific task. Execution of every C program begins with the main() function, no matter where the function is located in the program. So, every C program must have a main() function and this is the function where the execution of the program begins.
2. **{ and }**: The opening braces '{' indicates the beginning of the main function and the closing braces '}' indicates the ending of the main function. Everything between these two comprises the body of the main function and are called the blocks.

## Line 10:

printf("Hello World");

1. This line tells the compiler to display the message "Hello World" on the screen. This line is called a statement in C. Every statement is meant to perform some task. A semi-colon ';' is used to end a statement. The semi-colon character at the end of the statement is used to indicate that the statement is ending there.
2. The *printf()* function is used to print a character stream of data on the stdout console. Everything within " " is displayed on the output device.

## Line 12:

return 0;

1. This is also a statement. This statement is used to return a value from a function and indicates the finishing of a function. This statement is basically used in functions to return the results of the operations performed by a function.
2. **Indentation**: As you can see the printf and the return statement have been indented or moved to the right side. This is done to make the code more readable. In a program as Hello World, it does not seem to hold much relevance but as the program

becomes more complex, it makes the code more readable and less error-prone. Therefore, one must always use indentations and comments to make the code more readable.

# C Comments

The **comments in C** are human-readable explanations or notes in the source code of a C program. A comment makes the program easier to read and understand. These are the statements that are not executed by the compiler or an interpreter.
It is considered to be a good practice to document our code using comments.

## When and Why to use Comments in C programming?

1. A person reading a large code will be bemused if comments are not provided about details of the program.
2. C Comments are a way to make a code more readable by providing more descriptions.
3. C Comments can include a description of an algorithm to make code understandable.
4. C Comments can be used to prevent the execution of some parts of the code.

## Types of comments in C

In C there are two types of comments in C language:

- **Single-line comment**
- **Multi-line comment**


## 1. Single-line Comment in C

A single-line comment in C starts with ( // ) double forward slash. It extends till the end of the line and we don't need to specify its end.

### Syntax of Single Line C Comment

// This is a single line comment

### Example 1: C Program to illustrate single-line comment

```
// C program to illustrate
// use of single-line comment
#include <stdio.h>


int main(void)
{
```

```
    // This is a single-line comment
    printf("Welcome to GeeksforGeeks");
    return 0;
}
```

**Output:**

Welcome to GeeksforGeeks

### Comment at End of Code Line

We can also create a comment that displays at the end of a line of code using a single-line comment. But generally, it's better to practice putting the comment before the line of code.

```
// C program to demonstrate commenting after line of code
#include <stdio.h>

int main() {
    // single line comment here

    printf("Welcome to GeeksforGeeks"); // comment here
    return 0;
}
```

**Output**

Welcome to GeeksforGeeks

## 2. Multi-line Comment in C

The Multi-line comment in C starts with a forward slash and asterisk ( **/*** ) and ends with an asterisk and forward slash ( ***/*** ). Any text between **/*** and ***/*** is treated as a comment and is ignored by the compiler. It can apply comments to multiple lines in the program.

## Syntax of Multi-Line C Comment

```
/*Comment starts
   continues
   continues
   .
   .
   .
Comment ends*/
```

## Example 2:  C Program to illustrate the multi-line comment

```
/* C program to illustrate
use of
multi-line comment */
#include <stdio.h>
int main(void)
{
        /*
        This is a
        multi-line comment
        */

        /*
        This comment contains some code which
```

```
        will not be executed.
        printf("Code enclosed in Comment");
        */
        printf("Welcome to GeeksforGeeks");
        return 0;
}
```

**Output:**

Welcome to GeeksforGeeks

# C Variables

A **variable in C language** is the name associated with some memory location to store data of different types. There are many types of variables in C depending on the scope, storage class, lifetime, type of data they store, etc. A variable is the basic building block of a C program that can be used in expressions as a substitute in place of the value it stores.

## What is a variable in C?

*A **variable in C** is a memory location with some name that helps store some form of data and retrieves it when required. We can store different types of data in the variable and reuse the same variable for storing some other data any number of times.*

They can be viewed as the names given to the memory location so that we can refer to it without having to memorize the memory address. The size of the variable depends upon the data type it stores.

## C Variable Syntax

The syntax to declare a variable in C specifies the name and the type of the variable.

```
data_type variable_name = value;    // defining single variable
        or
data_type variable_name1, variable_name2;    // defining multiple variable
```

Here,

- **data_type:** Type of data that a variable can store.
- **variable_name:** Name of the variable given by the user.
- **value:** value assigned to the variable by the user.

**Example**

```
int var;    // integer variable
char a;     // character variable
float fff;  // float variables
```


**There are 3 aspects of defining a variable:**

1. Variable Declaration
2. Variable Definition
3. Variable Initialization

## 1. C Variable Declaration

Variable declaration in C tells the compiler about the existence of the variable with the given name and data type. When the variable is declared compiler automatically allocates the memory for it.

## 2. C Variable Definition

In the definition of a C variable, the compiler allocates some memory and some value to it. A defined variable will contain some random garbage value till it is not initialized.

**Example**

```
int var;
char var2;
```

## . C Variable Initialization

Initialization of a variable is the process where the user assigns some meaningful value to the variable.

**Example**

```
int var; // variable definition
var = 10; // initialization
    or
int var = 10; // variable declaration and definition
```

# How to use variables in C?

The below example demonstrates how the use variables in C language.

```
// C program to demonstrate the
// declaration, definition and
// initialization
#include <stdio.h>

int main()
{
        // declaration with definition
        int defined_var;

        printf("Defined_var: %d\n", defined_var);

        // initialization
        defined_var = 12;

        // declaration + definition + initialization
        int ini_var = 25;

        printf("Value of defined_var after initialization:
%d\n",defined_var);
        printf("Value of ini_var: %d", ini_var);

        return 0;
}
```

**Output**

Defined_var: 0

Value of defined_var after initialization: 12

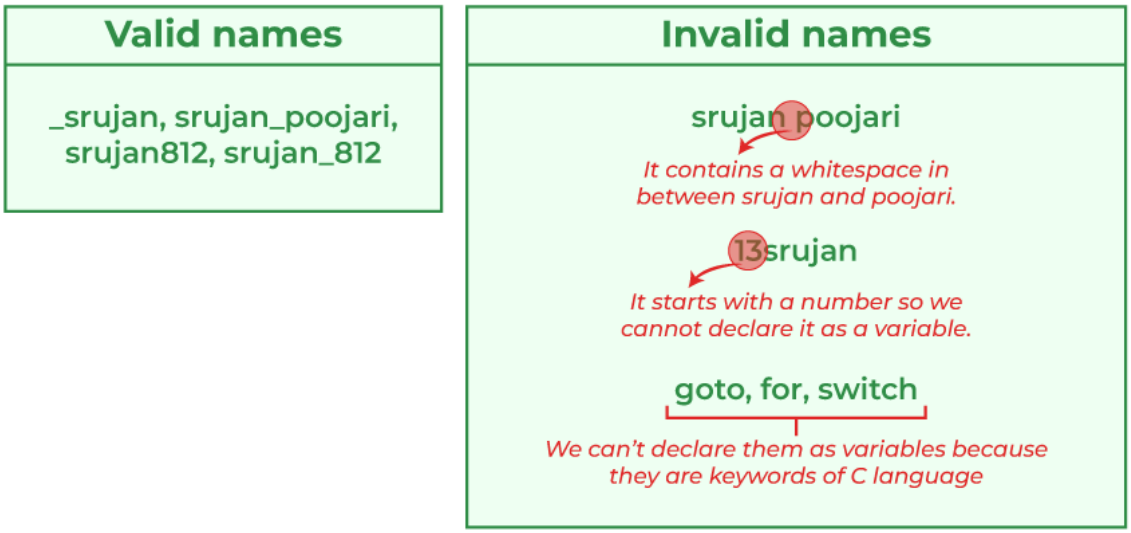
Value of ini_var: 25

# Rules for Naming Variables in C

**You can assign any name to the variable as long as it follows the following rules:**

1. A variable name must only contain alphabets, digits, and underscore.
2. A variable name must start with an alphabet or an underscore only. It cannot 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.




# C Variable Types

The C variables can be classified into the following types:

1. **Local Variables**
2. **Global Variables**
3. **Static Variables**

# 1. Local Variables in C

A **Local variable in C** is a variable that is declared inside a function or a block of code. Its scope is limited to the block or function in which it is declared.

## Example of Local Variable in C

```
// C program to declare and print local variable inside a
// function.
#include <stdio.h>

void function()
{
        int x = 10; // local variable
        printf("%d", x);
}

int main()
{
function();
 }
```

**Output**
10

In the above code, x can be used only in the scope of function(). Using it in the main function will give an error.

# 2. Global Variables in C

A **Global variable in C** is a variable that is declared outside the function or a block of code. Its scope is the whole program i.e. we can access the global variable anywhere in the C program after it is declared.

## Example of Global Variable in C

```
// C program to demonstrate use of global variable
#include <stdio.h>

int x = 20; // global variable

void function1() { printf("Function 1: %d\n", x); }

void function2() { printf("Function 2: %d\n", x); }

int main()
{

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

**Output**
Function 1: 20

Function 2: 20

In the above code, both functions can use the global variable as global variables are accessible by all the functions.

# 3. Static Variables in C

A **static variable in C** is a variable that is defined using the **static** keyword. It can be defined only once in a C program and its scope depends upon the region where it is declared (can be **global or local**).

The **default value** of static variables is **zero.**

## Syntax of Static Variable in C

static data_type variable_name = initial_value;

As its lifetime is till the end of the program, it can retain its value for multiple function calls as shown in the example.

## Example of Static Variable in C

```
// C program to demonstrate use of static variable
#include <stdio.h>

void function()
{
        int x = 20; // local variable
        static int y = 30; // static variable
        x = x + 10;
        y = y + 10;
        printf("\tLocal: %d\n\tStatic: %d\n", x, y);
}

int main()
{
        printf("First Call\n");
        function();
        printf("Second Call\n");
        function();
        printf("Third Call\n");
        function();
        return 0;
}
```

**Output**

First Call

Local: 30

Static: 40

Second Call

Local: 30

Static: 50

Third Call

Local: 30

Static: 60

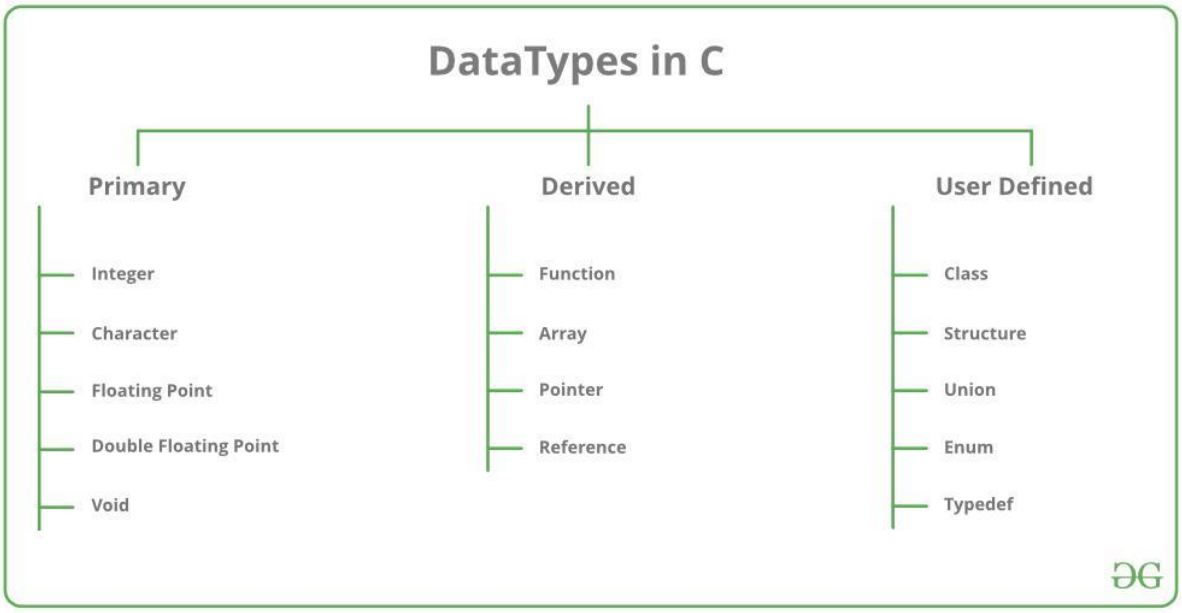
In the above example, we can see that the local variable will always print the same value whenever the function will be called whereas the static variable will print the incremented value in each function call.

# Data Types in C

Each variable in C has an associated data type. It specifies the type of data that the variable can store like integer, character, floating, double, etc. Each data type requires different amounts of memory and has some specific operations which can be performed over it. The data type is a collection of data with values having fixed values, meaning as well as its characteristics.

**The data types in C can be classified as follows:**

| 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 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 | Size (bytes) | Range | Format Specifier |
|---|---|---|---|
| **short int** | 2 | -32,768 to 32,767 | %hd |
| **unsigned short int** | 2 | 0 to 65,535 | %hu |
| **unsigned int** | 4 | 0 to 4,294,967,295 | %u |
| **int** | 4 | -2,147,483,648 to 2,147,483,647 | %d |

| Data Type | Size (bytes) | Range | Format Specifier |
|---|---|---|---|
| **long int** | 4 | -2,147,483,648 to 2,147,483,647 | %ld |
| **unsigned long int** | 4 | 0 to 4,294,967,295 | %lu |
| **long long int** | 8 | -(2^63) to (2^63)-1 | %lld |
| **unsigned long long int** | 8 | 0 to 18,446,744,073,709,551,615 | %llu |
| **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 |

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.

# Integer Data Type

The integer datatype in C is used to store the integer numbers(any number including positive, negative and zero without decimal part). Octal values, hexadecimal values, and decimal values can be stored in int data type in C.

- **Range:** -2,147,483,648 to 2,147,483,647
- **Size:** 4 bytes
- **Format Specifier:** %d

## Syntax of Integer

We use **int keyword** to declare the integer variable:
**int** *var_name;*

## Example of int

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

#include <stdio.h>


int 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);

    return 0;
}
```

**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

## Character Data Type

Character data type allows its variable to store only a single character. The size of the character is 1 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.

- **Range:** (-128 to 127) or (0 to 255)
- **Size:** 1 byte
- **Format Specifier:** %c

## Syntax of char

The **char keyword** is used to declare the variable of character type:

**char** *var_name;*

## Example of char

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

int main()
{
	char a = 'a';
	char c;

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

	a++;
	printf("Value of a after increment is: %c\n", a);

	// c is assigned ASCII values
	// which corresponds to the
	// character 'c'
	// a-->97 b-->98 c-->99
	// here c will be printed
	c = 99;

	printf("Value of c: %c", c);

	return 0;
}
```

**Output**

Value of a: a

Value of a after increment is: b

Value of c: c

# Float Data Type

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

## Syntax of float

The **float keyword** is used to declare the variable as a floating point:
**float** *var_name;*

## Example of Float

```
// C Program to demonstrate use
// of Floating types
#include <stdio.h>

int 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);

	return 0;
}
```

**Output**

9.000000

2.500000

0.000200

# Basic Input and Output in C

C language has standard libraries that allow input and output in a program. The **stdio.h** or **standard input output library** in C that has methods for input and output.

## scanf()

The scanf() method, in C, reads the value from the console as per the type specified. **Syntax:**

## printf()

The printf() method, in C, prints the value passed as the parameter to it, on the console screen. **Syntax:**
printf("%X", variableOfXType); where %X is the format specifier in C. It is a way to tell the compiler what type of data is in a variable and & is the address operator in C, which tells the compiler to change the real value of this variable, stored at this address in the memory.

## How to take input and output of basic types in C?

The basic type in C includes types like int, float, char, etc. Inorder to input or output the specific type, the **X** in the above syntax is changed with the specific format specifier of that type. The Syntax for input and output for these are:

- **Integer:**

**Input:** scanf("%d", &intVariable);
**Output:** printf("%d", intVariable);

- **Float:**

**Input:** scanf("%f", &floatVariable);
**Output:** printf("%f", floatVariable);

- **Character:**

**Input:** scanf("%c", &charVariable);
**Output:** printf("%c", charVariable);

```
// C program to show input and output

#include <stdio.h>

int main()
```

```
{
	// Declare the variables
	int num;
	char ch;
	float f;

	// --- Integer ---

	// Input the integer
	printf("Enter the integer: ");
	scanf("%d", &num);

	// Output the integer
	printf("\nEntered integer is: %d", num);

	// --- Float ---

	//For input Clearing buffer
	while((getchar()) != '\n');

	// Input the float
	printf("\n\nEnter the float: ");
	scanf("%f", &f);

	// Output the float
	printf("\nEntered float is: %f", f);

	// --- Character ---

	// Input the Character
	printf("\n\nEnter the Character: ");
	scanf("%c", &ch);

	// Output the Character
	printf("\nEntered character is: %c", ch);

	return 0;
}
```

**Output:**

Enter the integer: 10

Entered integer is: 10

Enter the float: 2.5

Entered float is: 2.500000

Enter the Character: A

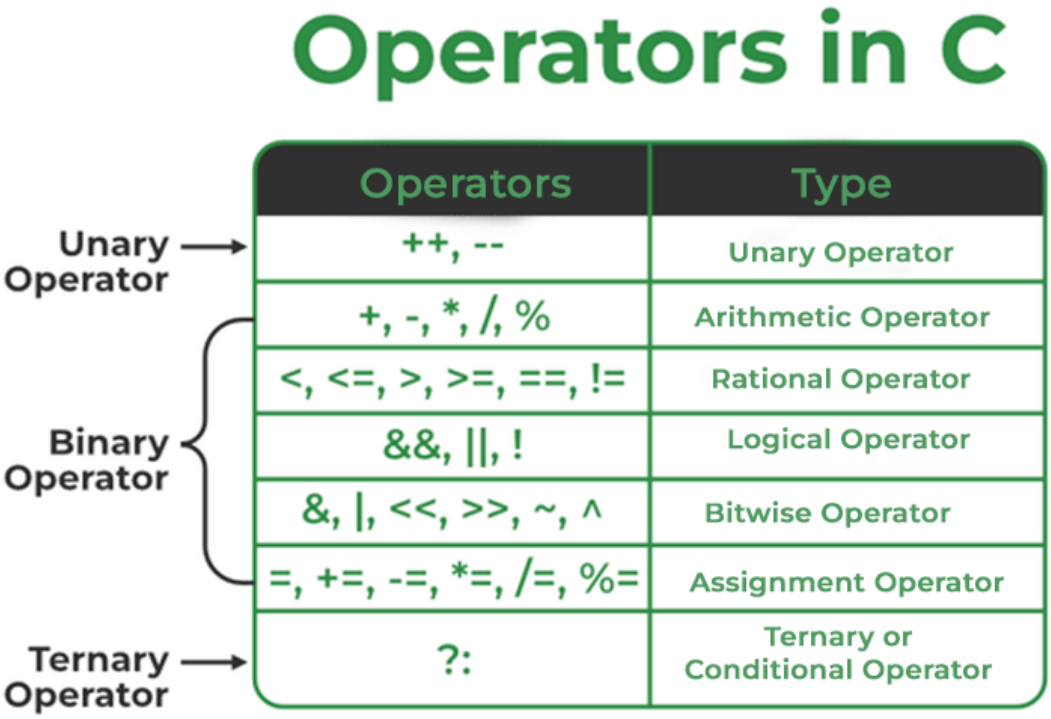
Entered Character is: A

# C Operator

In C language, operators are symbols that represent operations to be performed on one or more operands. They are the basic components of the C programming. In this article, we will learn about at all the built-in operators in C with examples.

## What is a C Operator?

An operator in C can be defined as the symbol that helps us to perform some specific mathematical, relational, bitwise, conditional, or logical computations on values and variables. The values and variables used with operators are called operands. So we can say that the operators are the symbols that perform operations on operands.




**For example,**
c = a + b;

Here, '+' is the operator known as the addition operator, and 'a' and 'b' are operands. The addition operator tells the compiler to add both of the operands 'a' and 'b'.

# Types of Operators in C

C language provides a wide range of operators that can be classified into 6 types based on their functionality:

1. Arithmetic Operators
2. Relational Operators
3. Logical Operators
4. Bitwise Operators
5. Assignment Operators
6. Other Operators

# 1. Arithmetic Operations in C

The arithmetic operators are used to perform arithmetic/mathematical operations on operands. There are 9 arithmetic operators in C language:

| S. No. | Symbol | Operator | Description | Syntax |
|---|---|---|---|---|
| 1 | + | **Plus** | Adds two numeric values. | **a + b** |
| 2 | – | **Minus** | Subtracts right operand from left operand. | **a – b** |
| 3 | * | **Multiply** | Multiply two numeric values. | **a * b** |
| 4 | / | **Divide** | Divide two numeric values. | **a / b** |
| 5 | % | **Modulus** | Returns the remainder after diving the left operand with the right operand. | **a % b** |
| 6 | + | **Unary Plus** | Used to specify the positive values. | **+a** |
| 7 | – | **Unary Minus** | Flips the sign of the value. | **-a** |
| 8 | ++ | **Increment** | Increases the value of the operand by 1. | a++ |

| S. No. | Symbol | Operator | Description | Syntax |
|---|---|---|---|---|
| 9 | — | **Decrement** | Decreases the value of the operand by 1. | a– |

## Example of C Arithmetic Operators

```
// C program to illustrate the arithmatic operators
#include <stdio.h>

int main()
{

    int a = 25, b = 5;

    // using operators and printing results
    printf("a + b = %d\n", a + b);
    printf("a - b = %d\n", a - b);
    printf("a * b = %d\n", a * b);
    printf("a / b = %d\n", a / b);
    printf("a % b = %d\n", a % b);
    printf("+a = %d\n", +a);
    printf("-a = %d\n", -a);
    printf("a++ = %d\n", a++);
    printf("a-- = %d\n", a--);

    return 0;
}
```

**Output**

a + b = 30

a - b = 20

a * b = 125

a / b = 5

a % b = 0

+a = 25

-a = -25

a++ = 25

a-- = 26

## 2. Relational Operators in C

The relational operators in C are used for the comparison of the two operands. All these operators are binary operators that return true or false values as the result of comparison.
These are a total of 6 relational operators in C:

| S. No. | Symbol | Operator | Description | Syntax |
|---|---|---|---|---|
| **1** | **<** | **Less than** | Returns true if the left operand is less than the right operand. Else false | **a < b** |
| **2** | **>** | **Greater than** | Returns true if the left operand is greater than the right operand. Else false | **a > b** |
| **3** | **<=** | **Less than or equal to** | Returns true if the left operand is less than or equal to the right operand. Else false | **a <= b** |

| S. No. | Symbol | Operator | Description | Syntax |
|---|---|---|---|---|
| 4 | >= | Greater than or equal to | Returns true if the left operand is greater than or equal to right operand. Else false | a >= b |
| 5 | == | Equal to | Returns true if both the operands are equal. | a == b |
| 6 | != | Not equal to | Returns true if both the operands are NOT equal. | a != b |

## Example of C Relational Operators

```
// C program to illustrate the relational operators
#include <stdio.h>

int main()
{

    int a = 25, b = 5;

    // using operators and printing results
    printf("a < b : %d\n", a < b);
    printf("a > b : %d\n", a > b);
    printf("a <= b: %d\n", a <= b);
    printf("a >= b: %d\n", a >= b);
    printf("a == b: %d\n", a == b);
    printf("a != b : %d\n", a != b);
```

```
    return 0;
}
```

**Output**

a < b  : 0

a > b  : 1

a <= b: 0

a >= b: 1

a == b: 0

a != b : 1

Here, 0 means false and 1 means true.

# 3. Logical Operator in C

Logical Operators are used to combine 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**.

| S. No. | Symbol | Operator | Description | Syntax |
|---|---|---|---|---|
| 1 | && | Logical AND | Returns true if both the operands are true. | a && b |
| 2 | \|\| | Logical OR | Returns true if both or any of the operand is true. | a \|\| b |
| 3 | ! | Logical NOT | Returns true if the operand is false. | !a |

## Example of Logical Operators in C

## // C program to illustrate the logical operators

```
#include <stdio.h>

int main()
{

    int a = 25, b = 5;

    // using operators and printing results
    printf("a && b : %d\n", a && b);
    printf("a || b : %d\n", a || b);
    printf("!a: %d\n", !a);

    return 0;
}
```

**Output**

a && b : 1

a || b : 1

!a: 0

# 4. Bitwise Operators in C

The Bitwise operators are used to perform bit-level operations on the operands. The operators are first converted to bit-level and then the calculation is performed on the operands. Mathematical operations such as addition, subtraction, multiplication, etc. can be performed at the bit level for faster processing.

There are 6 bitwise operators in C:

| S. No. | Symbol | Operator | Description | Syntax |
|---|---|---|---|---|
| 1 | & | Bitwise AND | Performs bit-by-bit AND operation and returns the result. | a && b |
| 2 | \| | Bitwise OR | Performs bit-by-bit OR | a \|\| b |

| S. No. | Symbol | Operator | Description | Syntax |
|---|---|---|---|---|
| | | | operation and returns the result. | |
| 3 | ^ | **Bitwise XOR** | Performs bit-by-bit XOR operation and returns the result. | **a ^ b** |
| 4 | ~ | **Bitwise First Complement** | Flips all the set and unset bits on the number. | **~a** |
| 5 | << | **Bitwise Leftshift** | Shifts the number in binary form by one place in the operation and returns the result. | **a << b** |
| 6 | >> | **Bitwise Rightshilft** | Shifts the number in binary form by one place in the operation and returns the result. | **a >> b** |

## Example of Bitwise Operators

```
// C program to illustrate the bitwise operators
#include <stdio.h>
```

```
int main()
{

	int a = 25, b = 5;

	// using operators and printing results
	printf("a & b: %d\n", a & b);
	printf("a | b: %d\n", a | b);
	printf("a ^ b: %d\n", a ^ b);
	printf("~a: %d\n", ~a);
	printf("a >> b: %d\n", a >> b);
	printf("a << b: %d\n", a << b);

	return 0;
}
```

**Output**

a & b: 1

a | b: 29

a ^ b: 28

~a: -26

a >> b: 0

a << b: 800

# 5. Assignment Operators in C

Assignment operators are used to assign value to a variable. 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.

The assignment operators can be combined with some other operators in C to provide multiple operations using single operator. These operators are called compound operators.

In C, there are 11 assignment operators :

| S. No. | Symbol | Operator | Description | Syntax |
|---|---|---|---|---|
| 1 | = | **Simple Assignment** | Assign the value of the right operand to the left operand. | **a = b** |
| 2 | += | **Plus and assign** | Add the right operand and left operand and assign this value to the left operand. | **a += b** |
| 3 | -= | **Minus and assign** | Subtract the right operand and left operand and assign this value to the left operand. | **a -= b** |
| 4 | *= | **Multiply and assign** | Multiply the right operand and left operand and assign this value to the left operand. | **a *= b** |
| 5 | /= | **Divide and assign** | Divide the left operand with the right operand and assign this value to the left operand. | **a /= b** |

| S. No. | Symbol | Operator | Description | Syntax |
|---|---|---|---|---|
| 6 | %= | **Modulus and assign** | Assign the remainder in the division of left operand with the right operand to the left operand. | **a %= b** |
| 7 | &= | **AND and assign** | Performs bitwise AND and assigns this value to the left operand. | **a &= b** |
| 8 | \|= | **OR and assign** | Performs bitwise OR and assigns this value to the left operand. | **a \|= b** |
| 9 | ^= | **XOR and assign** | Performs bitwise XOR and assigns this value to the left operand. | **a ^= b** |
| 10 | >>= | **Rightshift and assign** | Performs bitwise Rightshift and assign this value to the left operand. | **a >>= b** |

| S. No. | Symbol | Operator | Description | Syntax |
|---|---|---|---|---|
| 11 | <<= | **Leftshift and assign** | Performs bitwise Leftshift and assign this value to the left operand. | **a <<= b** |

## Example of C Assignment Operators

```
// C program to illustrate the arithmatic operators
#include <stdio.h>

int main()
{

    int a = 25, b = 5;

    // using operators and printing results
    printf("a = b: %d\n", a = b);
    printf("a += b: %d\n", a += b);
    printf("a -= b: %d\n", a -= b);
    printf("a *= b: %d\n", a *= b);
    printf("a /= b: %d\n", a /= b);
    printf("a %= b: %d\n", a %= b);
    printf("a &= b: %d\n", a &= b);
    printf("a |= b: %d\n)", a |= b);
    printf("a >>= b: %d\n", a >> b);
    printf("a <<= b: %d\n", a << b);

    return 0;
}
```

**Output**

a = b: 5

a += b: 10

a -= b: 5

a *= b: 25

a /= b: 5

a %= b: 0

a &= b: 0

a |= b: 5

)a >>= b: 0

a <<= b: 160

## Conditional Operator ( ? : )

- The conditional operator is the only ternary operator in C++.
- Here, 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.
- We may replace the use of if..else statements with conditional operators.

**Syntax**

operand1 **?** operand2 **:** operand3;