**CHAPTER-6**

 **OVERVIEW OF C PROGRAMMING LANGUAGE**

**C language** Tutorial with programming approach for beginners and professionals, helps you to understand the C language tutorial easily. Our C tutorial explains each topic with programs.

The C Language is developed for creating system applications that directly interact with the hardware devices such as drivers, kernels, etc.

C programming is considered as the base for other programming languages, that is why it is known as mother language.

It can be defined by the following ways:

1. Mother language
2. System programming language
3. Procedure-oriented programming language
4. Structured programming language
5. Mid-level programming language

1) C as a mother language

C language is considered as the mother language of all the modern programming languages because **most of the compilers, JVMs, Kernels, etc. are written in C language**, and most of the programming languages follow C syntax, for example, C++, Java, C#, etc.

It provides the core concepts like the array, strings, functions, file handling, etc. that are being used in many languages like C++, Java, C#, etc.

2) C as a system programming language

A system programming language is used to create system software. C language is a system programming language because it **can be used to do low-level programming (for example driver and kernel)**. It is generally used to create hardware devices, OS, drivers, kernels, etc. For example, Linux kernel is written in C.

It can't be used for internet programming like Java, .Net, PHP, etc.

3) C as a procedural language

A procedure is known as a function, method, routine, subroutine, etc. A procedural language **specifies a series of steps for the program to solve the problem**.

A procedural language breaks the program into functions, data structures, etc.

C is a procedural language. In C, variables and function prototypes must be declared before being used.

4) C as a structured programming language

A structured programming language is a subset of the procedural language. **Structure means to break a program into parts or blocks** so that it may be easy to understand.

In the C language, we break the program into parts using functions. It makes the program easier to understand and modify.

5) C as a mid-level programming language

C is considered as a middle-level language because it **supports the feature of both low-level and high-level languages**. C language program is converted into assembly code, it supports pointer arithmetic (low-level), but it is machine independent (a feature of high-level).

A **Low-level language** is specific to one machine, i.e., machine dependent. It is machine dependent, fast to run. But it is not easy to understand.

A **High-Level language** is not specific to one machine, i.e., machine independent. It is easy to understand.

C Program

In this tutorial, all C programs are given with C compiler so that you can quickly change the C program code.

File: main.c

1. #include <stdio.h>
2. **int** main() {
3. printf("good morning");
4. **return** 0;
5. }

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **History of C Language**Dennis Ritchie - founder of C language**History of C language** is interesting to know. Here we are going to discuss a brief history of the c language.**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 |

 |
| **Features of C Language**C featuresC is the widely used language. It provides many features that are given below.1. Simple
2. Machine Independent or Portable
3. Mid-level programming language
4. structured programming language
5. Rich Library
6. Memory Management
7. Fast Speed
8. Pointers
9. Recursion
10. Extensible

**1) Simple**C is a simple language in the sense that it provides a structured approach (to break the problem into parts), the rich set of library functions, data types, etc.**2) Machine Independent or Portable**Unlike assembly language, c programs can be executed on different machines with some machine specific changes. Therefore, C is a machine independent language.**3) Mid-level programming language**Although, C is intended to do low-level programming. It is used to develop system applications such as kernel, driver, etc. It also supports the features of a high-level language. That is why it is known as mid-level language.**4) Structured programming language**C is a structured programming language in the sense that we can break the program into parts using functions. So, it is easy to understand and modify. Functions also provide code reusability.**5) Rich Library**C provides a lot of inbuilt functions that make the development fast.**6) Memory Management**It supports the feature of dynamic memory allocation. In C language, we can free the allocated memory at any time by calling the free() function.**7) Speed**The compilation and execution time of C language is fast since there are lesser inbuilt functions and hence the lesser overhead.**8) Pointer**C provides the feature of pointers. We can directly interact with the memory by using the pointers. We can use pointers for memory, structures, functions, array, etc.**9) Recursion**In C, we can call the function within the function. It provides code reusability for every function. Recursion enables us to use the approach of backtracking.10) ExtensibleC language is extensible because it can easily adopt new features. |

C Language Introduction

[C](https://www.geeksforgeeks.org/c-programming-language/) is a procedural programming language. It was initially developed by Dennis Ritchie in the year 1972. It was mainly developed as a system programming language to write an operating system. The main features of C language include low-level access to memory, a simple set of keywords, and clean style, these features make C language suitable for system programmings like an operating system or compiler development.
Many later languages have borrowed syntax/features directly or indirectly from C language. Like syntax of Java, PHP, JavaScript, and many other languages are mainly based on C language. C++ is nearly a superset of C language (There are few programs that may compile in C, but not in C++).

**Beginning with C programming:**

1. **Structure of a C program**
After the above discussion, we can formally assess the structure of a C program. By structure, it is meant that any program can be written in this structure only. Writing a C program in any other structure will hence lead to a Compilation Error.

The structure of a C program is as follows: 

The components of the above structure are:

* 1. **Header Files Inclusion**: 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.

Some of C Header files:

* + - stdio.h – Defines core input and output functions
		- stdlib.h – Defines numeric conversion functions, pseudo-random network generator, memory allocation
		- string.h – Defines string handling functions
		- math.h – Defines common mathematical functions
		- stddef.h – Defines several useful types and macros.
		- stdint.h – Defines exact width integer types.

**Syntax to include a header file in C:**

#include

* 1. **Main Method Declaration:** The next part of a C program is to declare the main() function. The syntax to declare the main function is:

**Syntax to Declare main method:**

int main()

{}

* 1. **Variable Declaration:** The next part of any C program is the variable declaration. It refers to the variables that are to be used in the function. Please note that in the C program, no variable can be used without being declared. Also in a C program, the variables are to be declared before any operation in the function.

**Example:**

int main()

{

 **int a;**

.

.

* 1. **Body:** Body of a function in C program, refers to the operations that are performed in the functions. It can be anything like manipulations, searching, sorting, printing, etc.

**Example:**

int main()

{

 int a;

 **printf("%d", a);**

.

.

* 1. **Return Statement:** The last part in any C program is the return statement. The return statement refers to the returning of the values from a function. This return statement and return value depend upon the return type of the function. For example, if the return type is void, then there will be no return statement. In any other case, there will be a return statement and the return value will be of the type of the specified return type.

**Example:**

int main()

{

 int a;

 printf("%d", a);

 **return 0;**

}

1. **Writing first program:**
Following is first program in C

|  |
| --- |
| #include <stdio.h> int main(void) {     printf("hello");     return 0; }  |

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

**Line 2 [ int main(void) ]** There must to be starting point from where execution of compiled C program begins. In C, the execution typically begins with first line of main(). The void written in brackets indicates that the main doesn’t take any parameter (See [this](https://www.geeksforgeeks.org/difference-int-main-int-mainvoid/)for more details). main() can be written to take parameters also. We will be covering that in future posts.
The int written before main indicates return type of main(). The value returned by main indicates status of program termination. See [this](https://www.geeksforgeeks.org/fine-write-void-main-cc/)post for more details on return type.

**Line 3 and 6: [ { and } ]** In C language, a pair of curly brackets define a scope and mainly used in functions and control statements like if, else, loops. All functions must start and end with curly brackets.

**Line 4 [ printf(“GeeksQuiz”); ]** [printf()](http://www.cplusplus.com/reference/cstdio/printf/%22%20%5Ct%20%22_blank) is a standard library function to print something on standard output. The semicolon at the end of printf indicates line termination. In C, semicolon is always used to indicate end of statement.

**Line 5 [ return 0; ]** The return statement returns the value from main(). The returned value may be used by operating system to know termination status of your program. The value 0 typically means successful termination.

**How to execute the above program:**
Inorder to execute the above program, we need to have a compiler to compile and run our programs. There are certain online compilers like <https://ide.geeksforgeeks.org/>, <http://ideone.com/> or <http://codepad.org/> that can be used to start C without installing a compiler.
**Windows:** There are many compilers available freely for compilation of C programs like [Code Blocks](http://www.codeblocks.org/downloads/26) and [Dev-CPP](http://www.bloodshed.net/download.html).   We strongly recommend Code Blocks.

**Linux:** For Linux, [gcc](http://en.wikipedia.org/wiki/GNU_Compiler_Collection%22%20%5Ct%20%22_blank)comes bundled with the linux,  Code Blocks can also be used with Linux.

**Flow of C Program**

The C program follows many steps in execution.

To understand the flow of C program well,

 let us see a simple program first.

File: simple.c

1. #include <stdio.h>
2. **int** main(){
3. printf("Hello C Language");
4. **return** 0;
5. }

Execution Flow

Let's try to understand the flow of above program by the figure given below.

1) C program (source code) is sent to preprocessor first. The preprocessor is responsible to convert preprocessor directives into their respective values. The preprocessor generates an expanded source code.

2) Expanded source code is sent to compiler which compiles the code and converts it into assembly code.

3) The assembly code is sent to assembler which assembles the code and converts it into object code. Now a simple.obj file is generated.

4) The object code is sent to linker which links it to the library such as header files. Then it is converted into executable code. A simple.exe file is generated.

5) The executable code is sent to loader which loads it into memory and then it is executed. After execution, output is sent to console.

Constants in C

A constant is a value or variable that can't be changed in the program, for example: 10, 20, 'a', 3.4, "c programming" etc.

There are different types of constants in C programming.

List of Constants in C

|  |  |
| --- | --- |
| **Constant** | **Example** |
| Decimal Constant | 10, 20, 450 etc. |
| Real or Floating-point Constant | 10.3, 20.2, 450.6 etc. |
| Octal Constant | 021, 033, 046 etc. |
| Hexadecimal Constant | 0x2a, 0x7b, 0xaa etc. |
| Character Constant | 'a', 'b', 'x' etc. |
| String Constant | "c", "c program” etc. |

2 ways to define constant in C

There are two ways to define constant in C programming.

1. const keyword
2. #define preprocessor
3. C const keyword

The const keyword is used to define constant in C programming.

1. **const** **float** PI=3.14;

Now, the value of PI variable can't be changed.

1. #include<stdio.h>
2. **int** main(){
3. **const** **float** PI=3.14;
4. printf("The value of PI is: %f",PI);
5. **return** 0;
6. }

Output:

The value of PI is: 3.140000

If you try to change the the value of PI, it will render compile time error.

1. #include<stdio.h>
2. **int** main(){
3. **const** **float** PI=3.14;
4. PI=4.5;
5. printf("The value of PI is: %f",PI);
6. **return** 0;
7. }

Output: Compile Time Error: Cannot modify a const object

2) C #define preprocessor

The #define preprocessor is also used to define constant.

Constants are like a variable, except that their value never changes during execution once defined.

**C Constants**

C Constants is the most fundamental and essential part of the C programming language. Constants in C are the fixed values that are used in a program, and its value remains the same during the entire execution of the program.

* Constants are also called literals.
* Constants can be any of the [data types](https://www.w3schools.in/c-tutorial/data-types/).
* It is considered best practice to define constants using only upper-case names.

Table of Contents

[1. Constant Definition in C](https://www.w3schools.in/c-tutorial/constants/#Constant_Definition_in_C)

[2. Constant Types in C](https://www.w3schools.in/c-tutorial/constants/#Constant_Types_in_C)

[3. Integer Constant](https://www.w3schools.in/c-tutorial/constants/#Integer_Constant)

[4. Real constant](https://www.w3schools.in/c-tutorial/constants/#Real_constant)

[5. Single Character Constants](https://www.w3schools.in/c-tutorial/constants/#Single_Character_Constants)

[6. String Constants](https://www.w3schools.in/c-tutorial/constants/#String_Constants)

[7. Backslash character constant](https://www.w3schools.in/c-tutorial/constants/#Backslash_character_constant)

## Constant Definition in C

Syntax:

const type constant\_name;

const keyword defines a constant in C.

Example:

#include<stdio.h>

main()

{

  const int SIDE = 10;

  int area;

  area = SIDE\*SIDE;

  printf("The area of the square with side: %d is: %d sq. units"

  , SIDE, area);

}

Program Output:



Putting const either before or after the type is possible.

int const SIDE = 10;

or

const int SIDE = 10;

## Constant Types in C

Constants are categorized into two basic types, and each of these types has its subtypes/categories. These are:

Primary Constants

1. Numeric Constants
	* Integer Constants
	* Real Constants
2. Character Constants
	* Single Character Constants
	* String Constants
	* Backslash Character Constants

### Integer Constant

It's referring to a sequence of digits. Integers are of three types viz:

1. Decimal Integer
2. Octal Integer
3. Hexadecimal Integer

Example:

15, -265, 0, 99818, +25, 045, 0X6

### Real constant

The numbers containing fractional parts like 99.25 are called real or floating points constant.

### Single Character Constants

It simply contains a single character enclosed within ' and ' (a pair of single quote). It is to be noted that the character '**8**' is not the same as **8**. Character constants have a specific set of integer values known as ASCII values (American Standard Code for Information Interchange).

Example:

'X', '5', ';'

### String Constants

These are a sequence of characters enclosed in double quotes, and they may include letters, digits, special characters, and blank spaces. It is again to be noted that "**G**" and '**G**' are different - because "G" represents a string as it is enclosed within a pair of double quotes whereas 'G' represents a single character.

Example:

"Hello!", "2015", "2+1"

### Backslash character constant

C supports some character constants having a backslash in front of it. The lists of backslash characters have a specific meaning which is known to the compiler. They are also termed as "Escape Sequence".

For Example:

\t is used to give a tab

\n is used to give a new line

|  |  |
| --- | --- |
| **Constants** | **Meaning** |
| \a | beep sound |
| \b | backspace |
| \f | form feed |
| \n | new line |
| \r | carriage return |
| \t | horizontal tab |
| \v | vertical tab |
| \' | single quote |
| \" | double quote |
| \\ | backslash |
| \0 | null |

Secondary Constant

* [Array](https://www.w3schools.in/c-tutorial/arrays/)
* [Pointer](https://www.w3schools.in/c-tutorial/pointers/)
* [Structure](https://www.w3schools.in/c-tutorial/structures/)
* [Union](https://www.w3schools.in/c-tutorial/unions/)
* Enum

C - Constants and Literals

Constants refer to fixed values that the program may not alter during its execution. These fixed values are also called **literals**.

Constants can be of any of the basic data types like *an integer constant, a floating constant, a character constant, or a string literal*. There are enumeration constants as well.

Constants are treated just like regular variables except that their values cannot be modified after their definition.

Integer Literals

An integer literal can be a decimal, octal, or hexadecimal constant. A prefix specifies the base or radix: 0x or 0X for hexadecimal, 0 for octal, and nothing for decimal.

An integer literal can also have a suffix that is a combination of U and L, for unsigned and long, respectively. The suffix can be uppercase or lowercase and can be in any order.

Here are some examples of integer literals −

212 /\* Legal \*/

215u /\* Legal \*/

0xFeeL /\* Legal \*/

078 /\* Illegal: 8 is not an octal digit \*/

032UU /\* Illegal: cannot repeat a suffix \*/

Following are other examples of various types of integer literals −

85 /\* decimal \*/

0213 /\* octal \*/

0x4b /\* hexadecimal \*/

30 /\* int \*/

30u /\* unsigned int \*/

30l /\* long \*/

30ul /\* unsigned long \*/

Floating-point Literals

A floating-point literal has an integer part, a decimal point, a fractional part, and an exponent part. You can represent floating point literals either in decimal form or exponential form.

While representing decimal form, you must include the decimal point, the exponent, or both; and while representing exponential form, you must include the integer part, the fractional part, or both. The signed exponent is introduced by e or E.

Here are some examples of floating-point literals −

3.14159 /\* Legal \*/

314159E-5L /\* Legal \*/

510E /\* Illegal: incomplete exponent \*/

210f /\* Illegal: no decimal or exponent \*/

.e55 /\* Illegal: missing integer or fraction \*/

**Character Constants**

Character literals are enclosed in single quotes, e.g., 'x' can be stored in a simple variable of **char** type.

A character literal can be a plain character (e.g., 'x'), an escape sequence (e.g., '\t'), or a universal character (e.g., '\u02C0').

There are certain characters in C that represent special meaning when preceded by a backslash for example, newline (\n) or tab (\t).

Following is the example to show a few escape sequence characters −

#include <stdio.h>

int main() {

 printf("Hello\tWorld\n\n");

 return 0;

}

When the above code is compiled and executed, it produces the following result −

Hello World

String Literals

String literals or constants are enclosed in double quotes "". A string contains characters that are similar to character literals: plain characters, escape sequences, and universal characters.

You can break a long line into multiple lines using string literals and separating them using white spaces.

Here are some examples of string literals. All the three forms are identical strings.

"hello, dear"

"hello, \

dear"

"hello, " "d" "ear"

Defining Constants

There are two simple ways in C to define constants −

* Using **#define** preprocessor.
* Using **const** keyword.

The #define Preprocessor

Given below is the form to use #define preprocessor to define a constant −

#define identifier value

The following example explains it in detail −

#include <stdio.h>

#define LENGTH 10

#define WIDTH 5

#define NEWLINE '\n'

int main() {

 int area;

 area = LENGTH \* WIDTH;

 printf("value of area : %d", area);

 printf("%c", NEWLINE);

 return 0;

}

When the above code is compiled and executed, it produces the following result −

value of area : 50

The const Keyword

You can use **const** prefix to declare constants with a specific type as follows −

const type variable = value;

The following example explains it in detail −

#include <stdio.h>

int main() {

 const int LENGTH = 10;

 const int WIDTH = 5;

 const char NEWLINE = '\n';

 int area;

 area = LENGTH \* WIDTH;

 printf("value of area : %d", area);

 printf("%c", NEWLINE);

 return 0;

}

When the above code is compiled and executed, it produces the following result −

value of area : 50

Note that it is a good programming practice to define constants in CAPITALS.

**Variables in C**

Variables are memory locations(storage area) in the C programming language.

The primary purpose of variables is to store data in memory for later use. Unlike [constants](https://www.w3schools.in/c-tutorial/constants/) which do not change during the program execution, variables value may change during execution. If you declare a variable in C, that means you are asking to the operating system for reserve a piece of memory with that variable name.

Table of Contents

[1. Variable Definition in C](https://www.w3schools.in/c-tutorial/variables/#Variable_Definition_in_C)

[2. Variable Definition and Initialization](https://www.w3schools.in/c-tutorial/variables/#Variable_Definition_and_Initialization)

[3. Variable Assignment](https://www.w3schools.in/c-tutorial/variables/#Variable_Assignment)

[4. There are some rules on choosing variable names](https://www.w3schools.in/c-tutorial/variables/#There_are_some_rules_on_choosing_variable_names)

[5. C Program to Print Value of a Variable](https://www.w3schools.in/c-tutorial/variables/#C_Program_to_Print_Value_of_a_Variable)

## Variable Definition in C

Syntax:

type variable\_name;

or

type variable\_name, variable\_name, variable\_name;

## Variable Definition and Initialization

Example:

int width, height=5;

char letter='A';

float age, area;

double d;

width = 10; /\* actual initialization \*/

age = 26.5;

## Variable Assignment

Variable assignment is a process of assigning a value to a variable.

Example:

int width = 60;

int age = 31;

## There are some rules on choosing variable names

* A variable name can consist of Capital letters A-Z, lowercase letters a-z, digits 0-9, and the underscore character.
* The first character must be a letter or underscore.
* Blank spaces cannot be used in variable names.
* Special characters like #, $ are not allowed.
* C keywords cannot be used as variable names.
* Variable names are case sensitive.
* Values of the variables can be numeric or alphabetic.
* Variable type can be char, int, float, double or void.

## C Program to Print Value of a Variable

Example:

#include<stdio.h>

void main()

{

 int age =5; /\* c program to print value of a variable \*/

 printf("I am %d years old.\n", age);

}

Program Output:

I am 5 years old.

# Data Types in C

A data type specifies the type of data that a variable can store such as integer, floating, character, etc.



There are the following data types in C language.

|  |  |
| --- | --- |
| **Types** | **Data Types** |
| Basic Data Type | int, char, float, double |
| Derived Data Type | array, pointer, structure, union |
| Enumeration Data Type | enum |
| Void Data Type | void |

## Basic Data Types

The basic data types are integer-based and floating-point based. C language supports both signed and unsigned literals.

The memory size of the basic data types may change according to 32 or 64-bit operating system.

Let's see the basic data types. Its size is given **according to 32-bit architecture**.

|  |  |  |
| --- | --- | --- |
| **Data Types** | **Memory Size** | **Range** |
| **char** | 1 byte | −128 to 127 |
| signed char | 1 byte | −128 to 127 |
| unsigned char | 1 byte | 0 to 255 |
| **short** | 2 byte | −32,768 to 32,767 |
| signed short | 2 byte | −32,768 to 32,767 |
| unsigned short | 2 byte | 0 to 65,535 |
| **int** | 2 byte | −32,768 to 32,767 |
| signed int | 2 byte | −32,768 to 32,767 |
| unsigned int | 2 byte | 0 to 65,535 |
| **short int** | 2 byte | −32,768 to 32,767 |
| signed short int | 2 byte | −32,768 to 32,767 |
| unsigned short int | 2 byte | 0 to 65,535 |
| **long int** | 4 byte | -2,147,483,648 to 2,147,483,647 |
| signed long int | 4 byte | -2,147,483,648 to 2,147,483,647 |
| unsigned long int | 4 byte | 0 to 4,294,967,295 |
| **float** | 4 byte |  |
| **double** | 8 byte |  |
| **long double** | 10 byte |  |

**C Identifiers**

Identifiers are names given to different names given to entities such as constants, variables, structures, functions etc.

Example:

int amount;

double totalbalance;

In the above example, amount and totalbalance are identifiers and int, and double are keywords.

## Rules for Naming Identifiers

* An identifier can only have alphanumeric characters (a-z , A-Z , 0-9) (i.e. letters & digits) and underscore( \_ ) symbol.
* Identifier names must be unique
* The first character must be an alphabet or underscore.
* You cannot use a keyword as identifiers.
* Only first thirty-one (31) characters are significant.
* Must not contain white spaces.
* Identifiers are case-sensitive.

# C Operators

An operator is simply a symbol that is used to perform operations. There can be many types of operations like arithmetic, logical, bitwise, etc.

There are following types of operators to perform different types of operations in C language.

* Arithmetic Operators
* Relational Operators
* Shift Operators
* Logical Operators
* Bitwise Operators
* Ternary or Conditional Operators
* Assignment Operator
* Misc Operator

## Precedence of Operators in C

The precedence of operator species that which operator will be evaluated first and next. The associativity specifies the operator direction to be evaluated; it may be left to right or right to left.

Let's understand the precedence by the example given below:

1. **int** value=10+20\*10;

The value variable will contain **210** because \* (multiplicative operator) is evaluated before + (additive operator).

The precedence and associativity of C operators is given below:

|  |  |  |
| --- | --- | --- |
| **Category** | **Operator** | **Associativity** |
| Postfix | () [] -> . ++ - - | Left to right |
| Unary | + - ! ~ ++ - - (type)\* & sizeof | Right to left |
| Multiplicative | \* / % | Left to right |
| Additive | + - | Left to right |
| Shift | << >> | Left to right |
| Relational | < <= > >= | Left to right |
| Equality | == != | Left to right |
| Bitwise AND | & | Left to right |
| Bitwise XOR | ^ | Left to right |
| Bitwise OR | | | Left to right |
| Logical AND | && | Left to right |
| Logical OR | || | Left to right |
| Conditional | ?: | Right to left |
| Assignment | = += -= \*= /= %=>>= <<= &= ^= |= | Right to left |
| Comma | , | Left to right |

**Arithmetic Operators**

Arithmetic Operators are used to performing mathematical calculations like addition (+), subtraction (-), multiplication (\*), division (/) and modulus (%).

|  |  |
| --- | --- |
| Operator | Description |
| + | Addition |
| - | Subtraction |
| \* | Multiplication |
| / | Division |
| % | Modulus |

### C Program to Add Two Numbers

Example:

#include <stdio.h>

void main()

{

 int i=3,j=7,k; /\* Variables Defining and Assign values \*/ k=i+j;

 printf("sum of two numbers is %d\n", k);

}

Program Output:



## Increment and Decrement Operators

Increment and Decrement Operators are useful operators generally used to minimize the calculation, i.e. ++x and x++ means x=x+1 or -x and x−−means x=x-1. But there is a slight difference between ++ or −− written before or after the operand. Applying the pre-increment first add one to the operand and then the result is assigned to the variable on the left whereas post-increment first assigns the value to the variable on the left and then increment the operand.

|  |  |
| --- | --- |
| Operator | Description |
| ++ | Increment |
| −− | Decrement |

Example: To Demonstrate prefix and postfix modes.

#include <stdio.h>//stdio.h is a header file used for input.output purpose.

void main()

{

 //set a and b both equal to 5.

 int a=5, b=5;

 //Print them and decrementing each time.

 //Use postfix mode for a and prefix mode for b.

 printf("\n%d %d",a--,--b);

 printf("\n%d %d",a--,--b);

 printf("\n%d %d",a--,--b);

 printf("\n%d %d",a--,--b);

 printf("\n%d %d",a--,--b);

}

Program Output:

5 4

4 3

3 2

2 1

1 0

## Relational Operators

Relational operators are used to comparing two quantities or values.

|  |  |
| --- | --- |
| Operator | Description |
| == | Is equal to |
| != | Is not equal to |
| > | Greater than |
| < | Less than |
| >= | Greater than or equal to |
| <= | Less than or equal to |

## Logical Operators

C provides three logical operators when we test more than one condition to make decisions. These are: && (meaning logical AND), || (meaning logical OR) and ! (meaning logical NOT).

|  |  |
| --- | --- |
| Operator | Description |
| && | And operator. It performs logical conjunction of two expressions. (if both expressions evaluate to True, result is True. If either expression evaluates to False, the result is False) |
| || | Or operator. It performs a logical disjunction on two expressions. (if either or both expressions evaluate to True, the result is True) |
| ! | Not operator. It performs logical negation on an expression. |

## Bitwise Operators

C provides a special operator for bit operation between two variables.

|  |  |
| --- | --- |
| Operator | Description |
| << | Binary Left Shift Operator |
| >> | Binary Right Shift Operator |
| ~ | Binary Ones Complement Operator |
| & | Binary AND Operator |
| ^ | Binary XOR Operator |
| | | Binary OR Operator |

## Assignment Operators

Assignment operators applied to assign the result of an expression to a variable. C has a collection of shorthand assignment operators.

|  |  |
| --- | --- |
| Operator | Description |
| = | Assign |
| += | Increments then assign |
| -= | Decrements then assign |
| \*= | Multiplies then assign |
| /= | Divides then assign |
| %= | Modulus then assign |
| <<= | Left shift and assign |
| >>= | Right shift and assign |
| &= | Bitwise AND assign |
| ^= | Bitwise exclusive OR and assign |
| |= | Bitwise inclusive OR and assign |

## Conditional Operator

C offers a ternary operator which is the conditional operator (?: in combination) to construct conditional expressions.

|  |  |
| --- | --- |
| Operator | Description |
| ? : |  Conditional Expression |

## Special Operators

C supports some special operators

|  |  |
| --- | --- |
| Operator | Description |
| sizeof() |  Returns the size of a memory location. |
| & |  Returns the address of a memory location. |
| \* |  Pointer to a variable. |

### Program to demonstrate the use of sizeof operator

Example:

#include <stdio.h>

void main()

{

 int i=10; /\* Variables Defining and Assign values \*/ printf("integer: %d\n", sizeof(i));

}

Program Output:



# C Type Conversion – Implicit & Explicit Type Conversion in C

 When variables and constants of different types are combined in an expression then they are converted to same data type. The process of converting one predefined type into another is called type conversion.

Type conversion in c can be classified into the following two types:

## Implicit Type Conversion

## When the type conversion is performed automatically by the compiler without programmer’s intervention, such type of conversion is known as **implicit type conversion** or **type promotion**.

The compiler converts all operands into the data type of the largest operand.

The sequence of rules that are applied while evaluating expressions are given below:

All short and char are automatically converted to int, then,

1. If either of the operand is of type long double, then others will be converted to long double and result will be long double.
2. Else, if either of the operand is double, then others are converted to double.
3. Else, if either of the operand is float, then others are converted to float.
4. Else, if either of the operand is unsigned long int, then others will be converted to unsigned long int.
5. Else, if one of the operand is long int, and the other is unsigned int, then
	1. if a long int can represent all values of an unsigned int, the unsigned int is converted to long int.
	2. otherwise, both operands are converted to unsigned long int.
6. Else, if either operand is long int then other will be converted to long int.
7. Else, if either operand is unsigned int then others will be converted to unsigned int.

It should be noted that the final result of expression is converted to type of variable on left side of assignment operator before assigning value to it.

Also, conversion of float to int causes truncation of fractional part, conversion of double to float causes rounding of digits and the conversion of long int to int causes dropping of excess higher order bits.

**bool -> char -> short int -> int ->**

 **unsigned int -> long -> unsigned ->**

 **long long -> float -> double -> long double**

**Example of Type Implicit Conversion:**

**// An example of implicit conversion**

#include<stdio.h>

int main()

{

    int x = 10;    // integer x

    char y = 'a';  // character c

    // y implicitly converted to int. ASCII

    // value of 'a' is 97

    x = x + y;

    // x is implicitly converted to float

    float z = x + 1.0;

    printf("x = %d, z = %f", x, z);

    return 0;

}

|  |
| --- |
|  |

Output:

x = 107, z = 108.000000

## Explicit Type Conversion

The type conversion performed by the programmer by posing the data type of the expression of specific type is known as explicit type conversion.

The explicit type conversion is also known as **type casting**.

Type casting in c is done in the following form:

**(data\_type)expression;**

where, data\_type is any valid c data type, and expression may be constant, variable or expression.

For example,

x=(int)a+b\*d;

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

The following rules have to be followed while converting the expression from one type to another to avoid the loss of information:

1. All integer types to be converted to float.
2. All float types to be converted to double.
3. All character types to be converted to integer.

**// C program to demonstrate explicit type casting**

#include<stdio.h>

int main()

{

    double x = 1.2;

    // Explicit conversion from double to int

    int sum = (int)x + 1;

    printf("sum = %d", sum);

    return 0;

}

Output:

sum = 2

**Type Casting**

Type Casting in C is used to convert a variable from one data type to another data type, and after type casting compiler treats the variable as of the new data type.

Syntax:

(type\_name) expression

## Without Type Casting

Example:

#include <stdio.h>

main ()

{

 int a;

 a = 15/6;

 printf("%d",a);

}

Program Output:

In the above C program, 15/6 alone will produce integer value as 2.



## After Type Casting

#include <stdio.h>

main ()

{

 float a;

 a = (float) 15/6;

 printf("%f",a);

}

Program Output:

After type cast is done before division to retain float value 2.500000.

