**Important Points to Note while Writing a C++ Program:**

* Always include the necessary header files for the smooth execution of functions. For example, **<iostream>** must be included to use **std::cin** and **std::cout**.
* The execution of code begins from the **main()** function.
* It is a good practice to use **Indentation** and **comments** in programs for easy understanding.
* **cout** is used to print statements and **cin** is used to take inputs.

1) **// C++ program to display :** This line is a comment line. When a comment is encountered by a compiler, the compiler **simply skips that line of code**. Any line beginning with **‘//’** without quotes OR in between **/\*…\*/** in C++ is comment.

2) **#include:** In C++, all lines that start with pound (#) sign are called directives and are processed by a preprocessor which is a program invoked by the compiler. The **#include** directive tells the compiler to include a file and **#include<iostream>**. It tells the compiler to include the standard iostream file which contains declarations of all the standard input/output library functions.

3) **using namespace std;** This is used to import the entirety of the std namespace into the current namespace of the program. The statement using namespace std is generally considered a bad practice.

4) **int main():** 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.

5) **{ 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.

6) **std::cout<<“Hello World”;**: 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. Semi-colon character at the end of the statement is used to indicate that the statement is ending there. The **std::cout** is used to identify the standard character output device which is usually the desktop screen. Everything followed by the character “**<<**” is displayed to the output device.

7) **return 0;** : This is also a statement. This statement is used to return a value from a function and indicates the finishing of a function.

**Is it fine to write void main() or main() in C/C++?**

In C, **void main()** has no defined(legit) usage, and it can sometimes throw garbage results or an error. However, **main()** is used to denote the main function which takes no arguments and returns an integer data type.

***Note****: Even if your compiler accepts* ***void main()*** *avoid it, or risk being considered ignorant by C and C++ programmers. In C++,* ***main()*** *need not contain an explicit return statement. In that case, the value returned is 0, meaning successful execution.*

#include <iostream>

int main()

{

    std::cout<<"This program returns the integer value 0\n";

}

**Difference between “int main()” and “int main(void)” in C/C++?**

Consider the following two definitions of main().

EX.1  int main()

{

   /\*  \*/

   return 0;

}

and

EX.2  int main(void)

{

   /\*  \*/

   return 0;

}

**C/C++ Tokens**

A token is the smallest element of a program that is meaningful to the compiler. Tokens can be classified as follows:

* Keywords
* Identifiers
* Constants
* Strings
* Special Symbols
* Operators

**1.Keywords:**

Keywords are pre-defined or reserved words in a programming language. Each keyword is meant to perform a specific function in a program. **C** **language** supports **32** **keywords** which are given below:

auto double int struct

break else long switch

case enum register typedef

char extern return union

const float short unsigned

continue for signed void

default goto sizeof volatile

do if static while

While in **C++** there are **31 additional keywords** other than C Keywords they are:

asm bool catch class

const\_cast delete dynamic\_cast explicit

export false friend inline

mutable namespace new operator

private protected public reinterpret\_cast

static\_cast template this throw

true try typeid typename

using virtual wchar\_t

**2.Identifiers:**

Identifiers are used as the general terminology for the naming of variables, functions and arrays. These are user-defined names consisting of an arbitrarily long sequence of letters and digits with either a letter or the underscore(\_) as a first character. Identifier names must differ in spelling and case from any keywords. You cannot use keywords as identifiers; they are reserved for special use. Once declared, you can use the identifier in later program statements to refer to the associated value. A special kind of identifier, called a statement label, can be used in goto statements.

**There are certain rules that should be followed while naming c identifiers:**

* They must begin with a letter or underscore(\_).
* They must consist of only letters, digits, or underscore. No other special character is allowed.
* It should not be a keyword.
* It must not contain white space.
* It should be up to 31 characters long as only the first 31 characters are significant.
* **main**: method name.
* **a**: variable name.

**3.Constants:**

Constants are also like normal variables. But, the only difference is, their values can not be modified by the program once they are defined. Constants refer to fixed values. They are also called literals.

Constants may belong to any of the data type

**Syntax:**

const data\_type variable\_name;   (or)

const data\_type \*variable\_name;

**Types of Constants:**

1. Integer constants – Example: 0, 1, 1218, 12482
2. Real or Floating-point constants – Example: 0.0, 1203.03, 30486.184
3. Octal & Hexadecimal constants – Ex: octal: (013 )8 = (11)10, Hexadecimal: (013)16 = (19)10
4. Character constants -Example: ‘a’, ‘A’, ‘z’
5. String constants -Example: “GeeksforGeeks”

**4.Strings:**

Strings are nothing but an array of characters ended with a null character (‘\0’). This null character indicates the end of the string. Strings are always enclosed in double-quotes. Whereas, a character is enclosed in single quotes in C and C++.**Declarations for String**:

* char string[20] = {‘g’, ’e’, ‘e’, ‘k’, ‘s’, ‘f’, ‘o’, ‘r’, ‘g’, ’e’, ‘e’, ‘k’, ‘s’, ‘\0’};
* char string[20] = “geeksforgeeks”;
* char string [] = “geeksforgeeks”;
* char as “string[20]”, 20 bytes of memory space is allocated for holding the string value.
* When we declare char as “string[]”, memory space will be allocated as per the requirement during the execution of the program.

**5.Special Symbols:**

The following special symbols are used in C having some special meaning and thus, cannot be used for some other purpose**.[] () {}, ; \* = #**

* **Brackets[]:** Opening and closing brackets are used as array element reference. These indicate single and multidimensional subscripts.
* **Parentheses():** These special symbols are used to indicate function calls and function parameters.
* **Braces{}:** These opening and ending curly braces mark the start and end of a block of code containing more than one executable statement.
* **Comma (, ):** It is used to separate more than one statements like for separating parameters in function calls.
* **Colon(:):** It is an operator that essentially invokes something called an initialization list.
* **Semicolon(;):** It is known as a statement terminator. It indicates the end of one logical entity. That’s why each individual statement must be ended with a semicolon.
* **Asterisk (\*):** It is used to create a pointer variable and for the multiplication of variables.
* **Assignment operator(=):** It is used to assign values and for the logical operation validation.
* **Pre-processor (#):** The preprocessor is a macro processor that is used automatically by the compiler to transform your program before actual compilation.

**6.Operators:**

Operators are symbols that trigger an action when applied to C variables and other objects. The data items on which operators act upon are called operands. operators can be classified as follows:

* **Unary Operators:** Those operators that require only a single operand to act upon are known as unary operators. For Example increment and decrement operators
* **Binary Operators:** Those operators that require two operands to act upon are called binary operators. **Binary operators are classified into :**

1. Arithmetic operators
2. Relational Operators
3. Logical Operators
4. Assignment Operators
5. Bitwise Operator

* **Ternary Operator:** The operator that require three operands to act upon are called ternary operator. Conditional Operator(?) is also called ternary operator.

**Syntax**: (Expression1)? expression2: expression3;

**1. Arithmetic Operators:**

These operators are used to perform arithmetic/mathematical operations on operands.

Examples: (+, -, \*, /, %,++,–). Arithmetic operators are of two types:

**a) Unary Operators**: Operators that operate or work with a single operand are unary operators. For example: Increment(++) and Decrement(–) Operators

int val = 5;

++val; // 6

**b) Binary Operators**: Operators that operate or work with two operands are binary operators. For example: Addition(+), Subtraction(-), multiplication(\*), Division(/) operators

int a = 7;

int b = 2;

cout<<a+b; // 9

**2. Relational Operators:**

These are used for the comparison of the values of two operands. For example, checking if one operand is equal to the other operand or not, an operand is greater than the other operand or not, etc. Some of the relational operators are **(==, >= , <=** )(See this article for more reference).

int a = 3;

int b = 5;

a < b;

// operator to check if a is smaller than b

**3. Logical Operators:**

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

For example, the **logical AND** represented as **‘&&’ operator in C or C++** returns true when both the conditions under consideration are satisfied. Otherwise, it returns false. Therefore, a && b returns true when both a and b are true (i.e. non-zero)(See this article for more reference).

(4 != 5) **&&** (4 < 5); // true

**4. Bitwise Operators:**

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. The mathematical operations such as addition, subtraction, multiplication, etc. can be performed at bit-level for faster processing. For example, the **bitwise AND** represented as **& operator in C or C++** takes two numbers as operands and does AND on every bit of two numbers. The result of AND is 1 only if both bits are 1. (See this article for more reference).

*int a = 5, b = 9; // a = 5(00000101), b = 9(00001001)*

*cout << (a ^ b); // 00001100*

*cout <<(~a); // 11111010*

**5. Assignment Operators:**

Assignment operators are used to assigning 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.

**Different types of assignment operators are shown below:**

**a. “=”:** This is the simplest assignment operator. This operator is used to assign the value on the right to the variable on the left.

For example:

a = 10;

b = 20;

ch = 'y';

**b. “+=”:** This operator is combination of ‘+’ and ‘=’ operators. This operator first adds the current value of the variable on left to the value on the right and then assigns the result to the variable on the left.

(a += b) can be written as (a = a + b)

If initially value stored in a is 5. Then (a += 6) = 11.

**c. “-=”:** This operator is a combination of ‘-‘ and ‘=’ operators. This operator first subtracts the value on the right from the current value of the variable on left and then assigns the result to the variable on the left.

(a -= b) can be written as (a = a - b)

If initially value stored in a is 8. Then (a -= 6) = 2.

**d. “\*=”:** This operator is a combination of ‘\*’ and ‘=’ operators. This operator first multiplies the current value of the variable on left to the value on the right and then assigns the result to the variable on the left.

(a \*= b) can be written as (a = a \* b)

If initially, the value stored in a is 5. Then (a \*= 6) = 30.

**e. “/=”:** This operator is a combination of ‘/’ and ‘=’ operators. This operator first divides the current value of the variable on left by the value on the right and then assigns the result to the variable on the left.

(a /= b) can be written as (a = a / b)

If initially, the value stored in a is 6. Then (a /= 2) = 3.

**6. Other Operators:**

Apart from the above operators, there are some other operators available in C or C++ used to perform some specific tasks. Some of them are discussed here:

**a. sizeof operator:**

* sizeof is much used in the C/C++ programming language.
* It is a compile-time unary operator which can be used to compute the size of its operand.
* The result of sizeof is of the unsigned integral type which is usually denoted by size\_t.
* Basically, the sizeof the operator is used to compute the size of the variable.

**b. Comma Operator:**

* The comma operator (represented by the token) is a binary operator that evaluates its first operand and discards the result, it then evaluates the second operand and returns this value (and type).
* The comma operator has the lowest precedence of any C operator.
* Comma acts as both operator and separator.

**c. Conditional Operator:**

* The conditional operator is of the form Expression1? Expression2: Expression3.
* 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.