# **CPP**

|  |
| --- |
| // C++ program to display "Hello World"  // Header file for input output functions  #include <iostream>  **using** **namespace** std;    // Main() function: where the execution of program begins  **int** main()  {      // prints hello world      cout << "Hello World";    **return** 0;  } |

**Output**

Hello World

Let us now understand every line and the terminologies of the above program:

**1) // C++ program to display “Hello World”:** This line is a comment line. A comment is used to display additional information about the program. A comment does not contain any programming logic. 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. Click to know [**More about Comments.**](https://www.geeksforgeeks.org/comments-in-c-c/)

**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. Click to Know [**More on Preprocessors.**](https://www.geeksforgeeks.org/cc-preprocessors/)

**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. When we import a namespace we are essentially pulling all type definitions into the current scope. The std namespace is huge. The alternative to this statement is to specify the namespace to which the identifier belongs using the scope operator(::) each time we declare a type. Click to know [**More about using namespace std.**](https://www.geeksforgeeks.org/using-namespace-std-considered-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. 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. Click to know [**More about the main() function.**](https://www.geeksforgeeks.org/executing-main-in-c-behind-the-scene/)

**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. 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.

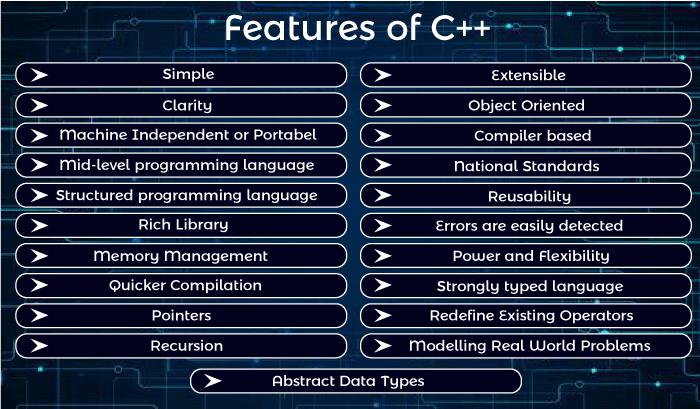
The std::cout is used to identify the standard character output device which is usually the desktop screen. Everything followed by the character “<<” is displayed to the output device. Click to know [**More on Input/Output.**](https://www.geeksforgeeks.org/basic-input-output-c/)

**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. This statement is basically used in functions to return the results of the operations performed by a function.

**8) Indentation**: As you can see the cout 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 hold much relevance, but as the programs become more complex, it makes the code more readable, less error-prone. Therefore, you must always use indentations and comments to make the code more readable. Must read the [**FAQ on the style of writing programs.**](https://www.geeksforgeeks.org/facts-and-question-related-to-style-of-writing-programs-in-c-c/)

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

1. 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**.
2. The execution of code begins from the **main()** function.
3. It is a good practice to use **Indentation** and **comments** in programs for easy understanding.
4. **cout** is used to print statements and **cin** is used to take inputs.



**#include<iostream.h>** includes the **standard input output** library functions. It provides **cin** and **cout** methods for reading from input and writing to output respectively.

**#include <conio.h>** includes the **console input output** library functions. The getch() function is defined in conio.h file.

|  |  |
| --- | --- |
| **Header File** | **Function and Description** |
| <iostream> | It is used to define the **cout, cin and cerr** objects, which correspond to standard output stream, standard input stream and standard error stream, respectively. |
| <iomanip> | It is used to declare services useful for performing formatted I/O, such as **setprecision and setw.** |
| <fstream> | It is used to declare services for user-controlled file processing. |

#include <iostream>

using namespace std;

int main()

{

std::cout << "Hello world" << std::endl;

}

Table

Description automatically generated

Whenever a variable is defined in C++, the compiler allocates some memory for that variable based on the data-type with which it is declared.

|  |  |  |
| --- | --- | --- |
| **Data Type** | **Size (in bytes)** | **Range** |
| short int | 2 | -32,768 to 32,767 |
| unsigned short int | 2 | 0 to 65,535 |
| unsigned int | 4 | 0 to 4,294,967,295 |
| int | 4 | -2,147,483,648 to 2,147,483,647 |
| long int | 4 | -2,147,483,648 to 2,147,483,647 |
| unsigned long int | 4 | 0 to 4,294,967,295 |
| long long int | 8 | -(2^63) to (2^63)-1 |
| unsigned long long int | 8 | 0 to 18,446,744,073,709,551,615 |
| signed char | 1 | -128 to 127 |
| unsigned char | 1 | 0 to 255 |
| float | 4 |  |
| double | 8 |  |
| long double | 12 |  |
| wchar\_t | 2 or 4 | 1 wide character |

**Note**: Above values may vary from compiler to compiler. In the above example, we have considered GCC 32 bit.

**Array**

An array is a collection of items stored at contiguous memory locations. The idea is to store multiple items of the same type together. This makes it easier to calculate the position of each element by simply adding an offset to a base value, i.e., the memory location of the first element of the array (generally denoted by the name of the array).

A two – dimensional array can be seen as a table with ‘x’ rows and ‘y’ columns where the row number ranges from 0 to (x-1) and the column number ranges from 0 to (y-1). A two – dimensional array ‘x’ with 3 rows and 3 columns is shown below:



Graphical user interface, text, application

Description automatically generated

Deference b/w class and method

| **S. No.** | **Class** | **Object** |
| --- | --- | --- |
| 1 | Class is used as a template for declaring and  creating the objects. | An object is an instance of a class. |
| 2 | When a class is created, no memory is allocated. | Objects are allocated memory space  whenever they are created. |
| 3 | The class has to be declared only once. | An object is created many times as per requirement. |
| 4 | A class cannot be manipulated as they are not available in the memory. | Objects can be manipulated. |
| 5 | A class is a logical entity. | An object is a physical entity. |
| 6 | It is declared with the class keyword | It is created with a class name in C++ and  with the new keywords in Java. |
| 7 | Class does not contain any values which  can be associated with the field. | Each object has its own values, which are associated with it. |
| 8 | A class is used to bind data as well as methods together as a single unit. | Objects are like a variable of the class. |
| 9 | Syntax for Declaring Class in C++:  class <classname> {  }; | Syntax for Instantiating an object for a Class in C++:  class Student {     public:        void put(){            cout<<“Function Called”<<endl;        }  };   // The class is declared here  int main(){           Student s1;   // Object created           s1.put();  } |
| 10 | Example: Bike | Example: Ducati, Suzuki, Kawasaki |

Method:

for swap two element

**swap**(a[0], b[0])

# **Array:**

Array problem list: <https://www.geeksforgeeks.org/c-programs-gq/array-programs-gq/>

int x[3][4]

The above array has 3 rows and 4 columns

**Vector in C++ STL:**

**#include <vector>**  
Vectors are the same as dynamic arrays with the **ability to resize itself automatically** when an element is inserted or deleted, with their storage being handled automatically by the container. Vector elements are placed in contiguous storage so that they can be accessed and traversed using iterators. In vectors, data is inserted at the end. Inserting at the end takes differential time, as sometimes the array may need to be extended. Removing the last element takes only constant time because no resizing happens. Inserting and erasing at the beginning or in the middle is linear in time.

**Advantages of Vector over arrays** :

1. Vector is ***template class*** and is ***C++ only construct*** whereas arrays are ***built-in language construct*** and present in both C and C++.
2. Vector are implemented as ***dynamic arrays* with *list interface*** whereas arrays can be implemented as ***statically or dynamically*** with ***primitive data type*** interface.

**#include <bits/stdc++.h>**

**using namespace std;**

**int main()**

**{**

**int array[100]; // Static Implementation**

**int\* arr = new int[100]; // Dynamic Implementation**

**vector<int> v; // Vector's Implementation**

**return 0;**

**}**

1. **Size of arrays are fixed** whereas the **vectors are resizable** i.e they can grow and shrink as vectors are allocated on heap memory.
2. Arrays **have to be deallocated explicitly** if defined dynamically whereas vectors are **automatically de-allocated** from heap memory.
3. Size of array **cannot be determined** if **dynamically allocated** whereas Size of the vector can be determined in **O(1) time**.
4. When arrays are passed to a function, a **separate parameter for size is also passed** whereas in case of passing a vector to a function, there is no such need as **vector maintains variables which keeps track of size of container at all times**.
5. When array becomes full and new elements are inserted; **no reallocation is done implicitly** whereas When vector becomes larger than its capacity, reallocation is done implicitly.
6. **Arrays *cannot be returned unless dynamically allocated* from a function** whereas v**ectors *can be returned* from a function**.
7. Arrays cannot be copied or assigned directly whereas Vectors can be copied or assigned directly.

**Some function of vector**

1. ‘[begin()](https://www.geeksforgeeks.org/vectorbegin-vectorend-c-stl/) – Returns an iterator pointing to the first element in the vector
2. [end()](https://www.geeksforgeeks.org/vectorbegin-vectorend-c-stl/) – Returns an iterator pointing to the theoretical element that follows the last element in the vector
3. [rbegin()](https://www.geeksforgeeks.org/vector-rbegin-and-rend-function-in-c-stl/) – Returns a reverse iterator pointing to the last element in the vector (reverse beginning). It moves from last to first element
4. [rend()](https://www.geeksforgeeks.org/vector-rbegin-and-rend-function-in-c-stl/) – Returns a reverse iterator pointing to the theoretical element preceding the first element in the vector (considered as reverse end)
5. [cbegin()](https://www.geeksforgeeks.org/vector-cbegin-vector-cend-c-stl/) – Returns a constant iterator pointing to the first element in the vector.
6. [cend()](https://www.geeksforgeeks.org/vector-cbegin-vector-cend-c-stl/) – Returns a constant iterator pointing to the theoretical element that follows the last element in the vector.
7. [crbegin()](https://www.geeksforgeeks.org/vectorcrend-vectorcrbegin-examples/) – Returns a constant reverse iterator pointing to the last element in the vector (reverse beginning). It moves from last to first element
8. [crend()](https://www.geeksforgeeks.org/vectorcrend-vectorcrbegin-examples/) – Returns a constant reverse iterator pointing to the theoretical element preceding the first element in the vector (considered as reverse end)

**std::string class in C++ :**

**Strings are slower** when compared to implementation than character array.

**getline():** This function is used to store a stream of characters as entered by the user in the object memory.

**push\_back():** This function is used to input a character at the end of the string.

**pop\_back() :** Introduced from C++11(for strings), this function is used to delete the last character from the string.

**capacity() :** This function returns the capacity allocated to the string, which can be equal to or more than the size of the string. Additional space is allocated so that when the new characters are added to the string, the operations can be done efficiently.

**resize() :** This function changes the size of the string, the size can be increased or decreased.

**length() :** This function finds the length of the string.

**shrink\_to\_fit():** This function decreases the capacity of the string and makes it equal to the minimum capacity of the string. This operation is useful to save additional memory if we are sure that no further addition of characters has to be made.

**begin():** This function returns an iterator to the beginning of the string.

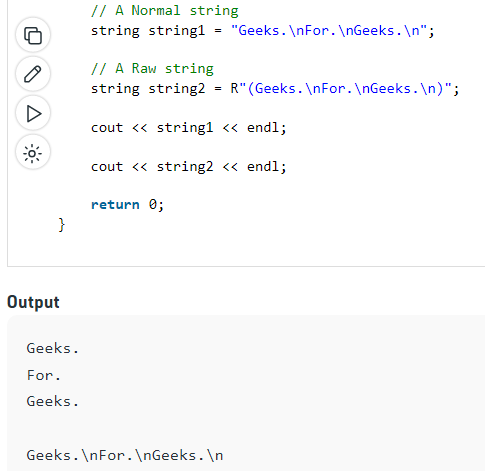
**end() :** his function returns an iterator to the end of the string.

**rbegin():** This function returns a reverse iterator pointing at the end of the string.

**rend():** This function returns a reverse iterator pointing at beginning of the string.

**Difference between an Ordinary String Literal and a Raw String Literal:**

| **Ordinary String Literal** | **Raw String Literal** |
| --- | --- |
| It does not need anything to be defined. | It needs a defined line{ parentheses ()} to start with the prefix **R.** |
| It does not allow/include nested characters. | It allows/includes nested character implementation. |
| It does not ignore any special meaning of character and implements their special characteristic. | It ignores all the special characters like **\n**and**\t** and treats them like normal text. |



# <bits/stdc++.h> in C++

It is basically a header file that includes every standard library. In programming contests, using this file is a good idea, when you want to reduce the time wasted in doing chores; especially when your rank is time sensitive.   
In programming contests, people do focus more on finding the algorithm to solve a problem than on software engineering. From, software engineering perspective, it is a good idea to minimize the include. If you use it actually includes a lot of files, which your program may not need, thus increases both compile time and program size unnecessarily.

**Disadvantages of bits/stdc++** 

* bits/stdc++.h is a non-standard header file of GNU C++ library. So, if you try to compile your code with some compiler other than GCC it might fail; e.g., MSVC do not have this header.
* Using it would include a lot of unnecessary stuff and increases compilation time.
* This header file is not part of the C++ standard and is therefore, non-portable, and should be avoided.
* Moreover, even if there were some catch-all headers in the standard, you would want to avoid it in lieu of specific headers, since the compiler has to actually read in and parse every included header (including recursively included headers) every single time that translation unit is compiled.

**Advantages of bits/stdc++** 

* In contests, using this file is a good idea, when you want to reduce the time wasted in doing chores; especially when your rank is time sensitive.
* This also reduces all the chores of writing all the necessary header files.
* You don’t have to remember all the STL of GNU C++ for every function you use.

# Converting Number/int to String in C++

There are***3 major methods to convert a number to a string***, which are as follows:

* **Using string Stream**
* **Using to\_string()**
* **Using boost lexical cast**

The function[**to\_string()**](https://www.geeksforgeeks.org/stdto_string-in-cpp/) accepts a number(which can be any data type) and returns the number in the desired string.

**String str = to\_string(Num);**

Now str is a string and stored Num values

# **Reverse the String**

Using the inbuilt “reverse” Function

There is a direct function in the “algorithm” header file for doing reverse that saves our time when programming.

// **Reverses elements in [begin, end]**

void reverse (BidirectionalIterator begin, BidirectionalIterator end);

# **Range-based for loop in C++**

Range-based for loop in C++ is added **since C++ 11**. It executes a for loop over a range.

**for ( range\_declaration : range\_expression )**

**loop\_statement**

**Parameters :**

**range\_declaration :**

a declaration of a named variable, whose type is the

type of the element of the sequence represented by

range\_expression, or a reference to that type.

Often uses the auto specifier for automatic type

deduction.

**range\_expression :**

any expression that represents a suitable sequence

or a braced-init-list.

**loop\_statement :**

any statement, typically a compound statement, which

is the body of the loop

**find how many times x occurs in vector v**

count(v.begin(),v.end(), x);