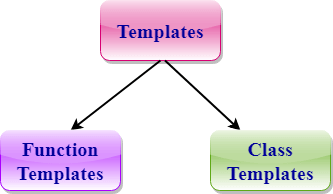
**C++ Templates**

**A C++ template is a powerful feature added to C++. It allows you to define the generic classes and generic functions and thus provides support for generic programming. Generic programming is a technique where generic types are used as parameters in algorithms so that they can work for a variety of data types.**

**Templates can be represented in two ways:**

* **Function templates**
* **Class templates**

****

**Function Templates:**

**We can define a template for a function. For example, if we have an add() function, we can create versions of the add function for adding the int, float or double type values.**

**Java Try Catch**

**Class Template:**

**We can define a template for a class. For example, a class template can be created for the array class that can accept the array of various types such as int array, float array or double array.**

**Function Template**

* **Generic functions use the concept of a function template. Generic functions define a set of operations that can be applied to the various types of data.**
* **The type of the data that the function will operate on depends on the type of the data passed as a parameter.**
* **For example, Quick sorting algorithm is implemented using a generic function, it can be implemented to an array of integers or array of floats.**
* **A Generic function is created by using the keyword template. The template defines what function will do.**

**Syntax of Function Template**

1. **template < class Ttype>**
2. **ret\_type func\_name(parameter\_list)**
3. **{**
4. **// body of function.**
5. **}**

**Where Ttype: It is a placeholder name for a data type used by the function. It is used within the function definition. It is only a placeholder that the compiler will automatically replace this placeholder with the actual data type.**

**class: A class keyword is used to specify a generic type in a template declaration.**

**Let's see a simple example of a function template:**

1. **#include <iostream>**
2. **using namespace std;**
3. **template<class T>**
4. **T add(T a,T b)**
5. **{**
6. **T result = a+b;**
7. **return result;**
9. **}**
10. **int main()**
11. **{**
12. **int i =2;**
13. **int j =3;**
14. **float m = 2.3;**
15. **float n = 1.2;**
16. **cout<<"Addition of i and j is :"<<add(i,j);**
17. **cout<<'\n';**
18. **cout<<"Addition of m and n is :"<<add(m,n);**
19. **return 0;**
20. **}**

**Output:**

**Addition of i and j is :5**

**Addition of m and n is :3.5**

**In the above example, we create the function template which can perform the addition operation on any type either it can be integer, float or double.**

**Function Templates with Multiple Parameters**

**We can use more than one generic type in the template function by using the comma to separate the list.**

**Syntax**

1. **template<class T1, class T2,.....>**
2. **return\_type function\_name (arguments of type T1, T2....)**
3. **{**
4. **// body of function.**
5. **}**

**In the above syntax, we have seen that the template function can accept any number of arguments of a different type.**

**Let's see a simple example:**

1. **#include <iostream>**
2. **using namespace std;**
3. **template<class X,class Y> void fun(X a,Y b)**
4. **{**
5. **std::cout << "Value of a is : " <<a<< std::endl;**
6. **std::cout << "Value of b is : " <<b<< std::endl;**
7. **}**
9. **int main()**
10. **{**
11. **fun(15,12.3);**
13. **return 0;**
14. **}**

**Output:**

**Value of a is : 15**

**Value of b is : 12.3**

**In the above example, we use two generic types in the template function, i.e., X and Y.**

**CLASS TEMPLATE**

**Class Template can also be defined similarly to the Function Template. When a class uses the concept of Template, then the class is known as generic class.**

**Syntax**

1. **template<class Ttype>**
2. **class class\_name**
3. **{**
4. **.**
5. **.**
6. **}**

**Ttype is a placeholder name which will be determined when the class is instantiated. We can define more than one generic data type using a comma-separated list. The Ttype can be used inside the class body.**

**Now, we create an instance of a class**

1. **class\_name<type> ob;**

**where class\_name: It is the name of the class.**

**type: It is the type of the data that the class is operating on.**

**ob: It is the name of the object.**

**Let's see a simple example:**

1. **#include <iostream>**
2. **using namespace std;**
3. **template<class T>**
4. **class A**
5. **{**
6. **public:**
7. **T num1 = 5;**
8. **T num2 = 6;**
9. **void add()**
10. **{**
11. **std::cout << "Addition of num1 and num2 : " << num1+num2<<std::endl;**
12. **}**
14. **};**
16. **int main()**
17. **{**
18. **A<int> d;**
19. **d.add();**
20. **return 0;**
21. **}**

**Output:**

**Addition of num1 and num2 : 11**

**In the above example, we create a template for class A. Inside the main() method, we create the instance of class A named as, 'd'.**

**CLASS TEMPLATE WITH MULTIPLE PARAMETERS**

**We can use more than one generic data type in a class template, and each generic data type is separated by the comma.**

**Syntax**

1. **template<class T1, class T2, ......>**
2. **class class\_name**
3. **{**
4. **// Body of the class.**
5. **}**

**Let's see a simple example when class template contains two generic data types.**

1. **#include <iostream>**
2. **using namespace std;**
3. **template<class T1, class T2>**
4. **class A**
5. **{**
6. **T1 a;**
7. **T2 b;**
8. **public:**
9. **A(T1 x,T2 y)**
10. **{**
11. **a = x;**
12. **b = y;**
13. **}**
14. **void display()**
15. **{**
16. **std::cout << "Values of a and b are : " << a<<" ,"<<b<<std::endl;**
17. **}**
18. **};**
20. **int main()**
21. **{**
22. **A<int,float> d(5,6.5);**
23. **d.display();**
24. **return 0;**
25. **}**

**Output:**

**Values of a and b are : 5,6.5**

**Nontype Template Arguments**

**The template can contain multiple arguments, and we can also use the non-type arguments In addition to the type T argument, we can also use other types of arguments such as strings, function names, constant expression and built-in types. Let' s see the following example:**

1. **template<class T, int size>**
2. **class array**
3. **{**
4. **T arr[size];           // automatic array initialization.**
5. **};**

**In the above case, the nontype template argument is size and therefore, template supplies the size of the array as an argument.**

**Arguments are specified when the objects of a class are created:**

1. **array<int, 15> t1;                        // array of 15 integers.**
2. **array<float, 10> t2;                    // array of 10 floats.**
3. **array<char, 4> t3;                      // array of 4 chars.**

**Let's see a simple example of nontype template arguments.**

1. **#include <iostream>**
2. **using namespace std;**
3. **template<class T, int size>**
4. **class A**
5. **{**
6. **public:**
7. **T arr[size];**
8. **void insert()**
9. **{**
10. **int i =1;**
11. **for (int j=0;j<size;j++)**
12. **{**
13. **arr[j] = i;**
14. **i++;**
15. **}**
16. **}**
18. **void display()**
19. **{**
20. **for(int i=0;i<size;i++)**
21. **{**
22. **std::cout << arr[i] << " ";**
23. **}**
24. **}**
25. **};**
26. **int main()**
27. **{**
28. **A<int,10> t1;**
29. **t1.insert();**
30. **t1.display();**
31. **return 0;**
32. **}**

**Output:**

**1 2 3 4 5 6 7 8 9 10**

**In the above example, the class template is created which contains the nontype template argument, i.e., size. It is specified when the object of class 'A' is created.**

**Points to Remember**

* **C++ supports a powerful feature known as a template to implement the concept of generic programming.**
* **A template allows us to create a family of classes or family of functions to handle different data types.**
* **Template classes and functions eliminate the code duplication of different data types and thus makes the development easier and faster.**
* **Multiple parameters can be used in both class and function template.**
* **Template functions can also be overloaded.**
* **We can also use nontype arguments such as built-in or derived data types as template arguments.**

**C++ Exception Handling**

**Exception Handling in C++ is a process to handle runtime errors. We perform exception handling so the normal flow of the application can be maintained even after runtime errors.**

**In C++, exception is an event or object which is thrown at runtime. All exceptions are derived from std::exception class. It is a runtime error which can be handled. If we don't handle the exception, it prints exception message and terminates the program.**

**Advantage**

**It maintains the normal flow of the application. In such case, rest of the code is executed even after exception.**

**C++ Exception Handling Keywords**

**In C++, we use 3 keywords to perform exception handling:**

* **try**
* **catch, and**
* **throw**

**try/catch**

**In C++ programming, exception handling is performed using try/catch statement. The C++ try block is used to place the code that may occur exception. The catch block is used to handle the exception.**

**C++ example without try/catch**

1. **#include <iostream>**
2. **using namespace std;**
3. **float division(int x, int y) {**
4. **return (x/y);**
5. **}**
6. **int main () {**
7. **int i = 50;**
8. **int j = 0;**
9. **float k = 0;**
10. **k = division(i, j);**
11. **cout << k << endl;**
12. **return 0;**
13. **}**

**Output:**

**Floating point exception (core dumped)**

**C++ try/catch example**

**#include <iostream>**

**using namespace std;**

**float division(int x, int y) {**

**if( y == 0 ) {**

**throw "Attempted to divide by zero!";**

**}**

**return (x/y);**

**}**

**int main () {**

**int i = 25;**

**int j = 0;**

**float k = 0;**

**try {**

**k = division(i, j);**

**cout << k << endl;**

**}catch (const char\* e) {**

**cerr << e << endl;**

**}**

**return 0;**

**}**

**Output:**

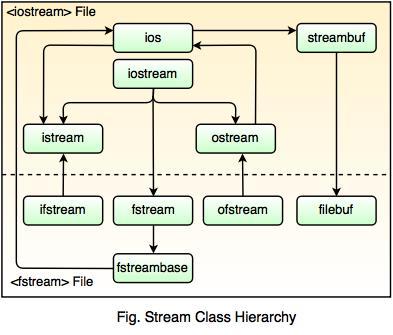
**Attempted to divide by zero!**

# Managing I/O Console using C++

**C++ Stream Classes**

**What is Stream?**

* **A stream is an abstraction. It is a sequence of bytes.**
* **It represents a device on which input and output operations are performed.**
* **It can be represented as a source or destination of characters of indefinite length.**
* **It is generally associated to a physical source or destination of characters like a disk file, keyboard or console.**
* **C++ provides standard iostream library to operate with streams.**
* **The iostream is an object-oriented library which provides Input/Output functionality using streams.**

**  
  
In the above figure, ios is the base class. The iostream class is derived from istream and ostream classes. The ifstream and ofstream are derived from istream and ostream, respectively. These classes handles input and output with the disk files.  
  
The fstream.h header file contains a declaration of ifstream, ofstream and fstream classes. The iostream.h file contains istream, ostream and iostream classes and included in the program while doing disk I/O operations.  
  
The filebuf class contains input and output operations with files. The streambuf class does not organize streams for input and output operations, only derived classes of streambuf performs I/O operations. These derived classes arranges a space for keeping input data and for sending output data.  
  
The istream and ostream invokes the filebuf functions to perform the insertion or extraction on the streams.**

|  |  |  |
| --- | --- | --- |
| **I/O Stream** | **Meaning** | **Description** |
| **istream** | **Input Stream** | **It reads and interprets input.** |
| **ostream** | **Output stream** | **It can write sequences of characters and represents other kinds of data.** |
| **ifstream** | **Input File Stream** | **The ifstream class is derived from fstreambase and istream by multiple inheritance.  This class accesses the member functions such as get(), getline(), seekg(), tellg() and read().  It provides open() function with the default input mode and allows input operations.** |
| **ofstream** | **Output File Stream** | **The ofstream class is derived from fstreambase and ostream classes.  This class accesses the member functions such as put(), seekp(), write() and tellp().  It provides the member function open() with the default output mode.** |
| **fstream** | **File Stream** | **The fstream allows input and output operations simultaneous on a filebuf.  It invokes the member function istream::getline() to read characters from the file.  This class provides the open() function with the default input mode.** |
| **fstreambase** | **File Stream Base** | **It acts as a base class for fstream, ifstream and ofstream. The open() and close() functions are defined in fstreambase.** |

**Advantages of Stream Classes**

* **Stream classes have good error handling capabilities.**
* **These classes work as an abstraction for the user that means the internal operation is encapsulated from the user.**
* **These classes are buffered and do not uses the memory disk space.**
* **These classes have various functions that make reading or writing a sequence of bytes easy for the programmer.**

# Unformatted input/output operations In C++

**In this article, we will discuss the unformatted Input/Output operations In C++. Using objects cin and cout for the input and the output of data of various types is possible because of overloading of operator >> and << to recognize all the basic C++ types. The operator >> is overloaded in the [istream class](https://www.geeksforgeeks.org/c-stream-classes-structure/) and operator << is overloaded in the ostream class.**

**The general format for reading data from the keyboard:**

***cin >> var1 >> var2 >> …. >> var\_n;***

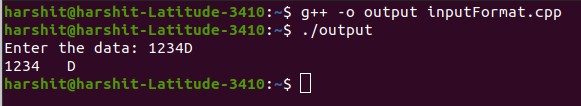
* **Here, var1, var2, ……, varn are the variable names that are declared already.**
* **The input data must be separated by white space characters and the data type of user input must be similar to the data types of the variables which are declared in the program.**
* **The operator >> reads the data character by character and assigns it to the indicated location.**
* **Reading of variables terminates when white space occurs or character type occurs that does not match the destination type.**

**Program 1:**

* **C++**

|  |
| --- |
| **// C++ program to illustrate the**  **// input and output of the data**  **// entered by user**  **#include <iostream>**  **using namespace std;**    **// Driver Code**  **int main()**  **{**  **int data;**  **char val;**    **// Input the data**  **cin >> data;**  **cin >> val;**    **// Print the data**  **cout << data << "   " << val;**    **return 0;**  **}** |

**Output:**

****

**Explanation: In the above program, 123 is stored in the variable val of integer, and B is passed to the next [cin object](https://www.geeksforgeeks.org/cin-in-c/) and stored in the data variable of character.**

### put() and get() functions:

**The class istream and ostream have predefined functions get() and put(), to handle single character input and output operations. The function get() can be used in two ways, such as get(char\*) and get(void) to fetch characters including blank spaces, newline characters, and tab. The function get(char\*) assigns the value to a variable and get(void) to return the value of the character.**

**Syntax:**

***char data;***

***// get() return the character value and assign to data variable  
data = cin.get();***

***// Display the value stored in data variable  
cout.put(data);***

**Example:**

***char c;***

***// directly assign value to c   
cin.get(c);***

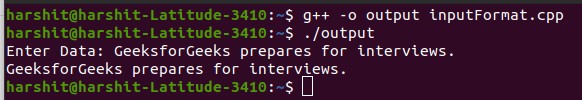
***// Display the value stored in c variable  
cout.put()***

**Program 2:**

* **C++**

|  |
| --- |
| **// C++ program to illustrate the**  **// input and output of data using**  **// get() and puts()**  **#include <iostream>**  **using namespace std;**    **// Driver Code**  **int main()**  **{**  **char data;**  **int count = 0;**    **cout << "Enter Data: ";**    **// Get the data**  **cin.get(data);**    **while (data != '\n') {**  **// Print the data**  **cout.put(data);**  **count++;**    **// Get the data again**  **cin.get(data);**  **}**    **return 0;**  **}** |

**Output:**

**[](https://media.geeksforgeeks.org/wp-content/cdn-uploads/20210401113733/Program2InputOutput.jpg)**

### getline() and write() functions:

**In C++, the function getline() and write() provide a more efficient way to handle line-oriented inputs and outputs. getline() function reads the complete line of text that ends with the new line character. This function can be invoked using the cin object.**

**Syntax:**

***cin.getline(variable\_to\_store\_line, size);***

**The reading is terminated by the ‘\n’ (newline) character. The new character is read by the function, but it does not display it, instead, it is replaced with a NULL character. After reading a particular string the cin automatically adds the newline character at end of the string.**

**The write() function displays the entire line in one go and its syntax is similar to the getline() function only that here cout object is used to invoke it.**

**Syntax:**

***cout.write(variable\_to\_store\_line, size);***

**The key point to remember is that the write() function does not stop displaying the string automatically when a NULL character occurs. If the size is greater than the length of the line then, the write() function displays beyond the bound of the line.**

**Program 3:**

* **C++**

|  |
| --- |
| **// C++ program to illustrate the**  **// input and output of file using**  **// getline() and write() function**  **#include <iostream>**  **#include <string>**  **using namespace std;**    **// Driver Code**  **int main()**  **{**  **char line[100];**    **// Get the input**  **cin.getline(line, 10);**    **// Print the data**  **cout.write(line, 5);**  **cout << endl;**    **// Print the data**  **cout.write(line, 20);**    **cout << endl;**    **return 0;**  **}** |

**Output:**

****

# Formatted I/O in C++

**C++ helps you to format the I/O operations like determining the number of digits to be displayed after the decimal point, specifying number base etc.**

**Example:**

* **If we want to add + sign as the prefix of out output, we can use the formatting to do so:**
* **stream.setf(ios::showpos)**

**If input=100, output will be +100**

* **If we want to add trailing zeros in out output to be shown when needed using the formatting:**
* **stream.setf(ios::showpoint)**

**If input=100.0, output will be 100.000**

**Note: Here, stream is referred to the streams defined in c++ like cin, cout, cerr, clog.**

**There are two ways to do so:**

1. **Using the ios class or various ios member functions.**
2. **Using manipulators(special functions)**
3. **Formatting using the ios members:**

**The stream has the format flags that control the way of formatting it means Using this setf function, we can set the flags, which allow us to display a value in a particular format. The ios class declares a bitmask enumeration called fmtflags in which the values(showbase, showpoint, oct, hex etc) are defined. These values are used to set or clear the format flags.**

**Few standard ios class functions are:**

* 1. **width(): The width method is used to set the required field width. The output will be displayed in the given width**
  2. **precision(): The precision method is used to set the number of the decimal point to a float value**
  3. **fill(): The fill method is used to set a character to fill in the blank space of a field**
  4. **setf(): The setf method is used to set various flags for formatting output**
  5. **unsetf(): The unsetf method is used To remove the flag setting**

**Working:**

|  |
| --- |
| **#include<bits/stdc++.h>**  **using namespace std;**    **// The width() function defines width**  **// of the next value to be displayed**  **// in the output at the console.**  **void IOS\_width()**  **{**  **cout << "--------------------------\n";**  **cout << "Implementing ios::width\n\n";**    **char c = 'A';**    **// Adjusting width will be 5.**  **cout.width(5);**    **cout << c <<"\n";**      **int temp = 10;**    **// Width of the next value to be**  **// displayed in the output will**  **// not be adjusted to 5 columns.**  **cout<<temp;**  **cout << "\n--------------------------\n";**  **}**    **void IOS\_precision()**  **{**  **cout << "\n--------------------------\n";**  **cout << "Implementing ios::precision\n\n";**  **cout << "Implementing ios::width";**  **cout.setf(ios::fixed, ios::floatfield);**  **cout.precision(2);**  **cout<<3.1422;**  **cout << "\n--------------------------\n";**  **}**    **// The fill() function fills the unused**  **// white spaces in a value (to be printed**  **// at the console), with a character of choice.**  **void IOS\_fill()**  **{**  **cout << "\n--------------------------\n";**  **cout << "Implementing ios::fill\n\n";**  **char ch = 'a';**    **// Calling the fill function to fill**  **// the white spaces in a value with a**  **// character our of choice.**  **cout.fill('\*');**    **cout.width(10);**  **cout<<ch <<"\n";**      **int i = 1;**    **// Once you call the fill() function,**  **// you don't have to call it again to**  **// fill the white space in a value with**  **// the same character.**  **cout.width(5);**  **cout<<i;**  **cout << "\n--------------------------\n";**  **}**    **void IOS\_setf()**  **{**  **cout << "\n--------------------------\n";**  **cout << "Implementing ios::setf\n\n";**  **int val1=100,val2=200;**  **cout.setf(ios::showpos);**  **cout<<val1<<" "<<val2;**  **cout << "\n--------------------------\n";**  **}**    **void IOS\_unsetf()**  **{**  **cout << "\n--------------------------\n";**  **cout << "Implementing ios::unsetf\n\n";**  **cout.setf(ios::showpos|ios::showpoint);**  **// Clear the showflag flag without**  **// affecting the showpoint flag**  **cout.unsetf(ios::showpos);**  **cout<<200.0;**  **cout << "\n--------------------------\n";**  **}**    **// Driver Method**  **int main()**  **{**  **IOS\_width();**  **IOS\_precision;**  **IOS\_fill();**  **IOS\_setf();**  **IOS\_unsetf();**  **return 0;**  **}** |

**Output:**

**--------------------------**

**Implementing ios::width**

**A**

**10**

**--------------------------**

**--------------------------**

**Implementing ios::fill**

**\*\*\*\*\*\*\*\*\*a**

**\*\*\*\*1**

**--------------------------**

**--------------------------**

**Implementing ios::setf**

**+100 +200**

**--------------------------**

**--------------------------**

**Implementing ios::unsetf**

**200.000**

**--------------------------**

1. **Formatting using Manipulators  
   The second way you can alter the format parameters of a stream is through the use of special functions called manipulators that can be included in an I/O expression.  
   The standard manipulators are shown below:**
   1. **boolalpha: The boolalpha manipulator of stream manipulators in C++ is used to turn on bool alpha flag**
   2. **dec: The dec manipulator of stream manipulators in C++ is used to turn on the dec flag**
   3. **endl: The endl manipulator of stream manipulators in C++ is used to Output a newline character.**
   4. **and: The and manipulator of stream manipulators in C++ is used to Flush the stream**
   5. **ends: The ends manipulator of stream manipulators in C++ is used to Output a null**
   6. **fixed: The fixed manipulator of stream manipulators in C++ is used to Turns on the fixed flag**
   7. **flush: The flush manipulator of stream manipulators in C++ is used to Flush a stream**
   8. **hex: The hex manipulator of stream manipulators in C++ is used to Turns on hex flag**
   9. **internal: The internal manipulator of stream manipulators in C++ is used to Turns on internal flag**
   10. **left: The left manipulator of stream manipulators in C++ is used to Turns on the left flag**
   11. [**noboolalpha:**](https://www.geeksforgeeks.org/ios-manipulators-noboolapha-function-in-c/)**The noboolalpha manipulator of stream manipulators in C++ is used to Turns off bool alpha flag**
   12. [**noshowbase:**](https://www.geeksforgeeks.org/ios-manipulators-noboolapha-function-in-c/)**The noshowbase manipulator of stream manipulators in C++ is used to Turns off showcase flag**
   13. [**noshowpoint**](https://www.geeksforgeeks.org/ios-manipulators-noshowpoint-function-in-c/)**: The noshowpoint manipulator of stream manipulators in C++ is used to Turns off show point flag**
   14. **noshowpos: The noshowpos manipulator of stream manipulators in C++ is used to Turns off showpos flag**
   15. [**noskipws**](https://www.geeksforgeeks.org/ios-manipulators-showpos-function-in-c/)**: The noskipws manipulator of stream manipulators in C++ is used to Turns off skipws flag**
   16. **nounitbuf: The nounitbuf manipulator of stream manipulators in C++ is used to Turns off the unit buff flag**
   17. **nouppercase: The nouppercase manipulator of stream manipulators in C++ is used to Turns off the uppercase flag**
   18. **oct: The oct manipulator of stream manipulators in C++ is used to Turns on oct flag**
   19. **resetiosflags(fmtflags f): The resetiosflags manipulator of stream manipulators in C++ is used to Turns off the flag specified in f**
   20. **right: The right manipulator of stream manipulators in C++ is used to Turns on the right flag**
   21. **scientific: The scientific manipulator of stream manipulators in C++ is used to Turns on scientific flag**
   22. **setbase(int base): The setbase manipulator of stream manipulators in C++ is used to Set the number base to base**
   23. **setfill(int ch): The setfill manipulator of stream manipulators in C++ is used to Set the fill character to ch**
   24. **setiosflags(fmtflags f): The setiosflags manipulator of stream manipulators in C++ is used to Turns on the flag specified in f**
   25. **setprecision(int p): The setprecision manipulator of stream manipulators in C++ is used to Set the number of digits of precision**
   26. **setw(int w): The setw manipulator of stream manipulators in C++ is used to Set the field width to w**
   27. [**showbase:**](https://www.geeksforgeeks.org/ios-manipulators-showbase-function-in-c/)**The showbase manipulator of stream manipulators in C++ is used to Turns on showbase flag**
   28. [**showpoint:**](https://www.geeksforgeeks.org/ios-manipulators-noboolapha-function-in-c/)**The showpoint manipulator of stream manipulators in C++ is used to Turns on show point flag**
   29. **showpos: The showpos manipulator of stream manipulators in C++ is used to Turns on showpos flag**
   30. **skipws: The skipws manipulator of stream manipulators in C++ is used to Turns on skipws flag**
   31. **unitbuf: The unitbuf manipulator of stream manipulators in C++ is used to turn on unitbuf flag**
   32. **uppercase: The uppercase manipulator of stream manipulators in C++ is used to turn on the uppercase flag**
   33. **ws: The ws manipulator of stream manipulators in C++ is used to skip leading white space**

**To access manipulators that take parameters (such as setw( )), you must include “iomanip” header file in your program.**

**Example:**

|  |
| --- |
| **#include <iomanip>**  **#include <iostream>**  **using namespace std;**    **void Example()**  **{**  **// performs ssame as setf( )**  **cout << setiosflags(ios::showpos);**  **cout << 123<<"\n";**    **// hexadecimal base (i.e., radix 16)**  **cout << hex << 100 << endl;**    **// Set the field width**  **cout << setfill('\*') << setw(10) << 2343.0;**  **}**    **int main()**  **{**  **Example();**  **return 0;**  **}** |

**Output:**

**+123**

**64**

**\*\*\*\*\*+2343**

**The manipulator setiosflags( ) .**