- Every program takes some data as input and generates processed data as an output following the familiar input process output cycle.
- It is essential to know how to provide the input data and present the results in the desired form.
- The use of the **cin** and **cout** is already known with the operator **>>** and **<<** for the input and output operations.
- C++ supports a rich set of I/O functions and operations. These functions use the advanced features of C++ such as classes, derived classes, and virtual functions.
- It also supports all of C's set of I/O functions and therefore can be used in C++ programs, but their use should be restrained due to two reasons.

1. I/O methods in C++ support the concept of OOPs.
2. I/O methods in C cannot handle the user-define data type such as class and object.

- It uses the concept of stream and stream classes to implement its I/O operations with the console and disk files.

C++ Stream

- The I/O system in C++ is designed to work with a wide variety of devices including terminals, disks, and tape drives.
- Although each device is very different, the I/O system supplies an interface to the programmer that is independent of the actual device being accessed. This interface is known as the **stream.**
- It comes with libraries that provide us with many ways for performing input and output.
- In C++ input and output are performed in the form of a sequence of bytes or more commonly known as **streams**.
- **Input Stream:** If the direction of flow of bytes is from the device(for example, Keyboard) to the main memory then this process is called input.
- **Output Stream:** If the direction of flow of bytes is opposite, i.e. from main memory to device(display screen) then this process is called output.
- **Header files available in C++ for Input/Output operations are:**
- **iostream**: iostream stands for standard input-output stream. This header file contains definitions of objects like cin, cout, cerr, etc.
- **iomanip**: iomanip stands for input-output manipulators. The methods declared in these files are used for manipulating streams. This file contains definitions of setw, setprecision, etc.
- **fstream**: This header file mainly describes the file stream. This header file is used to handle the data being read from a file as input or data being written into the file as output.
- **bits/stdc++:** This header file 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. To know more about this header file refer this article.

- In C++ after the header files, we often use '*using namespace std;*'. The reason behind it is that all of the standard library definitions are inside the namespace std. As the library functions are not defined at global scope, so in order to use them we use *namespace std*. So, that we don't need to write STD:: at every line (eg. STD::cout etc.).
- The two instances **cout in C++** and **cin in C++** of iostream class are used very often for printing outputs and taking inputs respectively. These two are the most basic methods of taking input and printing output in C++. To use cin and cout in C++ one must include the header file *iostream* in the program.
- The C++ **cout** statement is the instance of the **ostream** class.
- cin statement is the instance of the class **istream**
- Un-buffered standard error stream (cerr): The C++ cerr is the standard error stream that is used to output the errors.
- This is also an instance of the iostream class.
- As cerr in C++ is un-buffered so it is used when one needs to display the error message immediately.
- It does not have any buffer to store the error message and display it later.
- The main difference between cerr and cout comes when you would like to redirect output using "cout" that gets redirected to file if you use "cerr" the error doesn't get stored in file.(This is what un-buffered means ..It cant store the message)

```
int main()
{
   cerr << "An error occurred";
   return 0;
}
```

- **buffered standard error stream (clog)**: This is also an instance of ostream class and used to display errors but unlike cerr the error is first inserted into a buffer and is stored in the buffer until it is not fully filled. or the buffer is not explicitly flushed (using flush()). The error message will be displayed on the screen too.

```
int main()
{
   clog << "An error occurred";
   return 0;
}
```

- Although each device is very different, the I/O system supplies an interface to the programmer that is independent of the actual device being accessed. This interface is known as the **stream.**
- stream refers to the stream of characters that are transferred between the program thread and i/o.
- In C++ there are number of stream classes for defining various streams related with files and for doing input-output operations.
- All these classes are defined in the file **iostream.h**. Figure given below shows the hierarchy of these classes.
- **ios class** is topmost class in the stream classes hierarchy. It is the base class for **istream, ostream,** and **streambuf** class.
- **istream** and **ostream** serves the base classes for **iostream** class. The class **istream** is used for input and **ostream** for the output.
- Class **ios** is indirectly inherited to **iostream** class using **istream** and **ostream**. To avoid the duplicity of data and member functions of **ios** class, it is declared as virtual base class when inheriting in **istream** and **ostream** as

```
class istream: virtual public ios
{
};
class ostream: virtual public ios
{
};
```

- The **_withassign classes** are provided with extra functionality for the assignment operations that's why **_withassign** classes.

Facilities provided by these stream classes.

- **The ios class:** The ios class is responsible for providing all input and output facilities to all other stream classes.
- **The istream class:** This class is responsible for handling input stream. It provides number of function for handling chars, strings and objects such as **get, getline, read, ignore, putback** etc..

```
#include <iostream>
using namespace std;
```

```
int main()
{
   char x;

   // used to scan a single char
   cin.get(x);

   cout << x;
}
```

- **The ostream class:** This class is responsible for handling output stream. It provides number of function for handling chars, strings and objects such as **write, put** etc..
- #include <iostream>
- using namespace std;

```
int main()
{
   char x;

   // used to scan a single char
   cin.get(x);

   // used to put a single char onto the screen.
   cout.put(x);
}
```

- **The iostream:** This class is responsible for handling both input and output stream as both **istream class** and o**stream class** is inherited into it. It provides function of both **istream class** and o**stream class** for handling chars, strings and objects such as **get, getline, read, ignore, putback, put, write** etc..

```
#include <iostream>
using namespace std;
```

```
int main()
{
   // this function display
   // ncount character from array
   cout.write(“Hello World", 7);   // Hello W
}
```

- **istream_withassign class:** This class is variant of **istream** that allows object assignment.
- The predefined object **cin** is an object of this class and thus may be reassigned at run time to a different **istream** object.
- **ostream_withassign class:** This class is variant of **ostream** that allows object assignment.
- The predefined objects **cout, cerr, clog** are objects of this class and thus may be reassigned at run time to a different **ostream** object.
- Objects used to communicate through the standard input and output devices are declared in <iostream> file and are available in to std namespace. Objects cin, cout, cerr and clog are declared for narrow character streams; whereas objects wcin, wcout, wcerr and wclog for wide-character streams.

Unformatted input/output operations In C++

- 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** and operator **<<** is overloaded in the ostream class.
- The general format for reading data from the keyboard:
- *cin >> var1 >> var2 >> …. >> var_n;*
- **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.
- *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);
- getline() and write() functions**:**
- 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**.
- *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.
- *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.

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.
- 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

- There are two ways to do formatting

1. Using the ios class or various ios member functions.
2. Using manipulators(special functions)

- **Formatting using the ios members:**

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

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.
- To access manipulators that take parameters (such as setw( )), you must include “iomanip” header file in your program.
- The standard manipulators are shown below:
- boolalpha: The boolalpha manipulator of stream manipulators in C++ is used to turn on bool alpha flag
- dec: The dec manipulator of stream manipulators in C++ is used to turn on the dec flag
- endl: The endl manipulator of stream manipulators in C++ is used to Output a newline character.
- and: The and manipulator of stream manipulators in C++ is used to Flush the stream
- ends: The ends manipulator of stream manipulators in C++ is used to Output a null
- fixed: The fixed manipulator of stream manipulators in C++ is used to Turns on the fixed flag
- flush: The flush manipulator of stream manipulators in C++ is used to Flush a stream
- hex: The hex manipulator of stream manipulators in C++ is used to Turns on hex flag
- internal: The internal manipulator of stream manipulators in C++ is used to Turns on internal flag
- left: The left manipulator of stream manipulators in C++ is used to Turns on the left flag

File

- Files are used to store data in a storage device permanently. File handling provides a mechanism to store the output of a program in a file and to perform various operations on it. A stream is an abstraction that represents a device on which operations of input and output are performed

File Handling through C++ Classes

- File handling is used to store data permanently in a computer. Using file handling we can store our data in secondary memory (Hard disk).
  How to achieve the File Handling
  For achieving file handling we need to follow the following steps:-
  STEP 1-Naming a file
  STEP 2-Opening a file
  STEP 3-Writing data into the file
  STEP 4-Reading data from the file
  STEP 5-Closing a file.

- We give input to the executing program and the execution program gives back the output. The sequence of bytes given as input to the executing program and the sequence of bytes that comes as output from the executing program are called stream. In other words, streams are nothing but the flow of data in a sequence.
- The input and output operation between the executing program and the devices like keyboard and monitor are known as “console I/O operation”. The input and output operation between the executing program and files are known as “disk I/O operation”.

Classes for File stream operations

- The I/O system of C++ contains a set of classes which define the file handling methods.
- These include ifstream, ofstream and fstream classes.
- These classes are derived from fstream and from the corresponding iostream class.
- These classes, designed to manage the disk files, are declared in fstream and therefore we must include this file in any program that uses files.
- ios:- ios stands for input output stream.
- This class is the base class for other classes in this class hierarchy.
- This class contains the necessary facilities that are used by all the other derived classes for input and output operations.
- istream:- istream stands for input stream.
- This class is derived from the class ‘ios’.
- This class handle input stream.
- The extraction operator(>>) is overloaded in this class to handle input streams from files to the program execution.
- This class declares input functions such as get(), getline() and read().
- ostream:- ostream stands for output stream.
- This class is derived from the class ‘ios’.
- This class handle output stream.
- The insertion operator(<<) is overloaded in this class to handle output streams to files from the program execution.
- This class declares output functions such as put() and write().
- streambuf:- This class contains a pointer which points to the buffer which is used to manage the input and output streams.
- fstreambase:- This class provides operations common to the file streams. Serves as a base for fstream, ifstream and ofstream class.
- This class contains open() and close() function.
- ifstream:- This class provides input operations.

- It contains open() function with default input mode.
- Inherits the functions get(), getline(), read(), seekg() and tellg() functions from the istream.
- ofstream:- This class provides output operations.
- It contains open() function with default output mode.
- Inherits the functions put(), write(), seekp() and tellp() functions from the ostream.
- fstream:- This class provides support for simultaneous input and output operations.
- Inherits all the functions from istream and ostream classes through iostream.
- filebuf:- Its purpose is to set the file buffers to read and write.
- We can also use file buffer member function to determine the length of the file.
- In C++, files are mainly dealt by using three classes fstream, ifstream, ofstream available in fstream headerfile.
- ofstream: Stream class to write on files
- ifstream: Stream class to read from files
- fstream: Stream class to both read and write from/to files.
- Now the first step to open the particular file for read or write operation. We can open file by
  1. passing file name in constructor at the time of object creation
  2. using the open method

For e.g.

- Open File by using constructor
- ifstream (const char* filename, ios_base::openmode mode = ios_base::in);
- ifstream fin(filename, openmode) by default openmode = ios::in
- ifstream fin("filename");
- Open File by using open method
- Calling of default constructor
- ifstream fin;
- fin.open(filename, openmode)
- fin.open("filename");