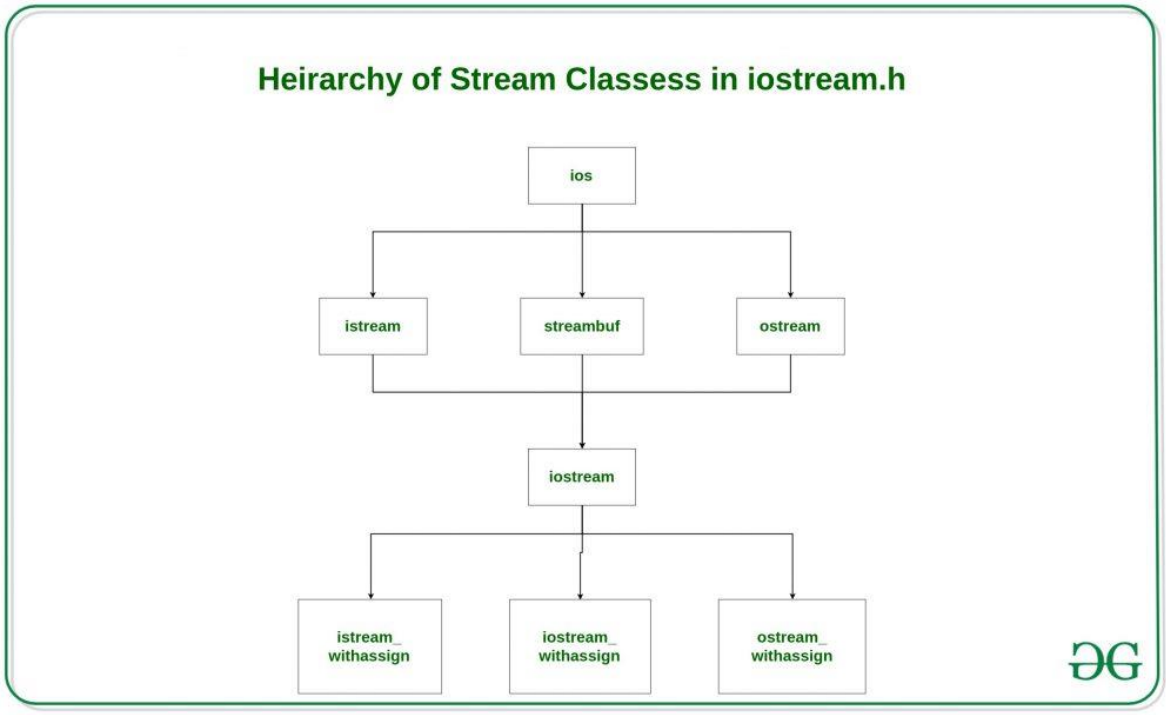
5.1 Stream Classes Structure

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.



1. **ios class** is topmost class in the stream classes hierarchy. It is the base class for **istream, ostream,** and **streambuf** class.

2. **istream** and **ostream** serves the base classes for **iostream** class. The class **istream** is used for input and **ostream** for the output.

3. 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

{

};
```

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

Facilities provided by these stream classes.

1. **The ios class:** The ios class is responsible for providing all input and output facilities to all other stream classes.

2. **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.**

   Example:

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

int main()
{
    char x;

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

    cout << x;
}
```

   Input:

```
g
```

   Output:

```
g
```

3. **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..
   Example:

```
#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);
}
```

   Input:

```
g
```

Output:

```
g
```

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

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

int main()
{

    // this function display
    // ncount character from array
    cout.write("geeksforgeeks", 5);
}
```

Output:

```
geeks
```

5. **istream_withassign class:** This class is variant of **istream** that allows object assigment. The predefined object **cin** is an object of this class and thus may be reassigned at run time to a different **istream** object.
Example: To show that **cin** is object of **istream** class.

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

class demo {
public:
    int dx, dy;

    // operator overloading using friend function
    friend void operator>>(demo& d, istream& mycin)
    {
        // cin assigned to another object mycin
        mycin >> d.dx >> d.dy;
    }
};

int main()
{
    demo d;
    cout << "Enter two numbers dx and dy\n";
```

```
    // calls operator >> function and
    // pass d and cin as reference
    d >> cin; // can also be written as operator >> (d, cin)
;

    cout << "dx = " << d.dx << "\tdy = " << d.dy;
}
```

Input:

```
4 5
```

Output:

```
Enter two numbers dx and dy

4 5

dx = 4  dy = 5
```

6. **ostream_withassign class:** This class is variant of **ostream** that allows object assigment. The predefined objects **cout, cerr, clog** are objects of this class and thus may be reassigned at run time to a different **ostream** object. **Example:**To show that **cout** is object of **ostream** class.

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

class demo {
public:
    int dx, dy;

    demo()
    {
        dx = 4;
        dy = 5;
    }

    // operator overloading using friend function
    friend void operator<<(demo& d, ostream& mycout)
    {
        // cout assigned to another object mycout
        mycout << "Value of dx and dy are \n";
        mycout << d.dx << " " << d.dy;
    }
};

int main()
{
    demo d; // default constructor is called
```

```
    // calls operator << function and
    // pass d and cout as reference
    d << cout; // can also be written as operator << (d,
cin) ;
}
```

Output:

```
Value of dx and dy are

4 5
```

Opening a File

A file must be opened before you can read from it or write to it. Either **ofstream** or **fstream** object may be used to open a file for writing. And ifstream object is used to open a file for reading purpose only.

Following is the standard syntax for open() function, which is a member of fstream, ifstream, and ofstream objects.

```
void open(const char *filename, ios::openmode mode);
```

Here, the first argument specifies the name and location of the file to be opened and the second argument of the **open()** member function defines the mode in which the file should be opened.

Sr.No	Mode Flag & Description
1	**ios::app** Append mode. All output to that file to be appended to the end.
2	**ios::ate** Open a file for output and move the read/write control to the end of the file.
3	**ios::in** Open a file for reading.
4	**ios::out** Open a file for writing.
5	**ios::trunc**

	If the file already exists, its contents will be truncated before opening the file.

You can combine two or more of these values by **OR**ing them together. For example if you want to open a file in write mode and want to truncate it in case that already exists, following will be the syntax −

```
ofstream outfile;

outfile.open("file.dat", ios::out | ios::trunc );
```

Similar way, you can open a file for reading and writing purpose as follows −

```
fstream  afile;

afile.open("file.dat", ios::out | ios::in );
```

Closing a File

When a C++ program terminates it automatically flushes all the streams, release all the allocated memory and close all the opened files. But it is always a good practice that a programmer should close all the opened files before program termination.

Following is the standard syntax for close() function, which is a member of fstream, ifstream, and ofstream objects.

```
void close();
```

Writing to a File

While doing C++ programming, you write information to a file from your program using the stream insertion operator (<<) just as you use that operator to output information to the screen. The only difference is that you use an **ofstream** or **fstream** object instead of the **cout** object.

Reading from a File

You read information from a file into your program using the stream extraction operator (>>) just as you use that operator to input information from the keyboard. The only difference is that you use an **ifstream** or **fstream** object instead of the **cin** object.

Read and Write Example

Following is the C++ program which opens a file in reading and writing mode. After writing information entered by the user to a file named afile.dat, the program reads information from the file and outputs it onto the screen −

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

int main () {
   char data[100];

   // open a file in write mode.
   ofstream outfile;
   outfile.open("afile.dat");

   cout << "Writing to the file" << endl;
   cout << "Enter your name: ";
   cin.getline(data, 100);

   // write inputted data into the file.
   outfile << data << endl;

   cout << "Enter your age: ";
   cin >> data;
   cin.ignore();

   // again write inputted data into the file.
   outfile << data << endl;

   // close the opened file.
   outfile.close();

   // open a file in read mode.
   ifstream infile;
   infile.open("afile.dat");

   cout << "Reading from the file" << endl;
   infile >> data;

   // write the data at the screen.
   cout << data << endl;

   // again read the data from the file and display it.
   infile >> data;
   cout << data << endl;
```

```
   // close the opened file.
   infile.close();

   return 0;
}
```

When the above code is compiled and executed, it produces the following sample input and output −

```
$./a.out
Writing to the file
Enter your name: Zara
Enter your age: 9
Reading from the file
Zara
9
```

Above examples make use of additional functions from cin object, like getline() function to read the line from outside and ignore() function to ignore the extra characters left by previous read statement.

File Position Pointers

Both **istream** and **ostream** provide member functions for repositioning the file-position pointer. These member functions are **seekg** ("seek get") for istream and **seekp** ("seek put") for ostream.

The argument to seekg and seekp normally is a long integer. A second argument can be specified to indicate the seek direction. The seek direction can be **ios::beg** (the default) for positioning relative to the beginning of a stream, **ios::cur** for positioning relative to the current position in a stream or **ios::end** for positioning relative to the end of a stream.

The file-position pointer is an integer value that specifies the location in the file as a number of bytes from the file's starting location. Some examples of positioning the "get" file-position pointer are −

```
// position to the nth byte of fileObject (assumes ios::beg)
fileObject.seekg( n );

// position n bytes forward in fileObject
fileObject.seekg( n, ios::cur );

// position n bytes back from end of fileObject
fileObject.seekg( n, ios::end );

// position at end of fileObject
fileObject.seekg( 0, ios::end );
```