Unit-4 Stream and Files

5.1 C++ Stream Classes,

5.2 Classes for File Stream Operations

5.3 Opening File,

5.4 Closing File

5.5 Detection of End of Files and File Modes

5.6 Reading from File

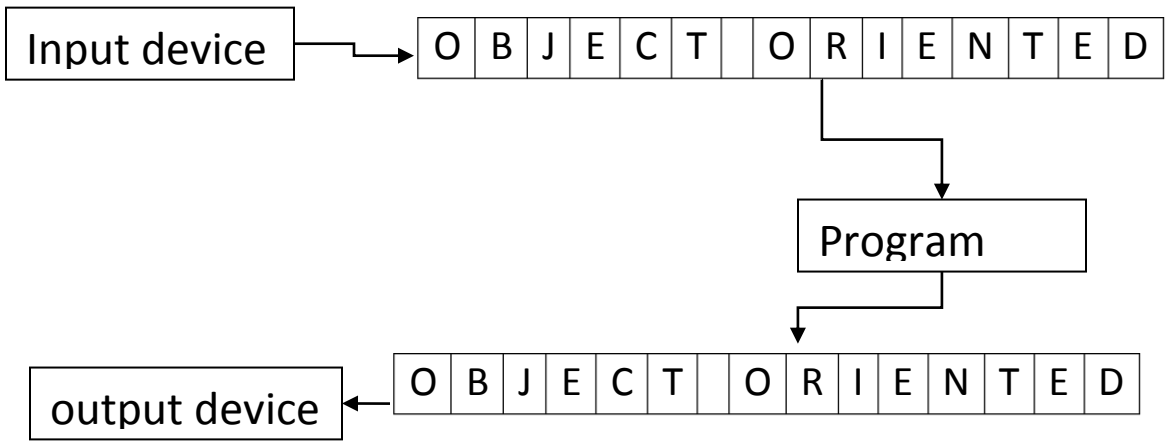
5.7 Writing to files

5.1 C++ Stream Classes

5.1.1 Stream: -

A stream is a sequence of bytes. It acts either as a score from which the input data can be obtained or as a destination to which the output data can be sent.

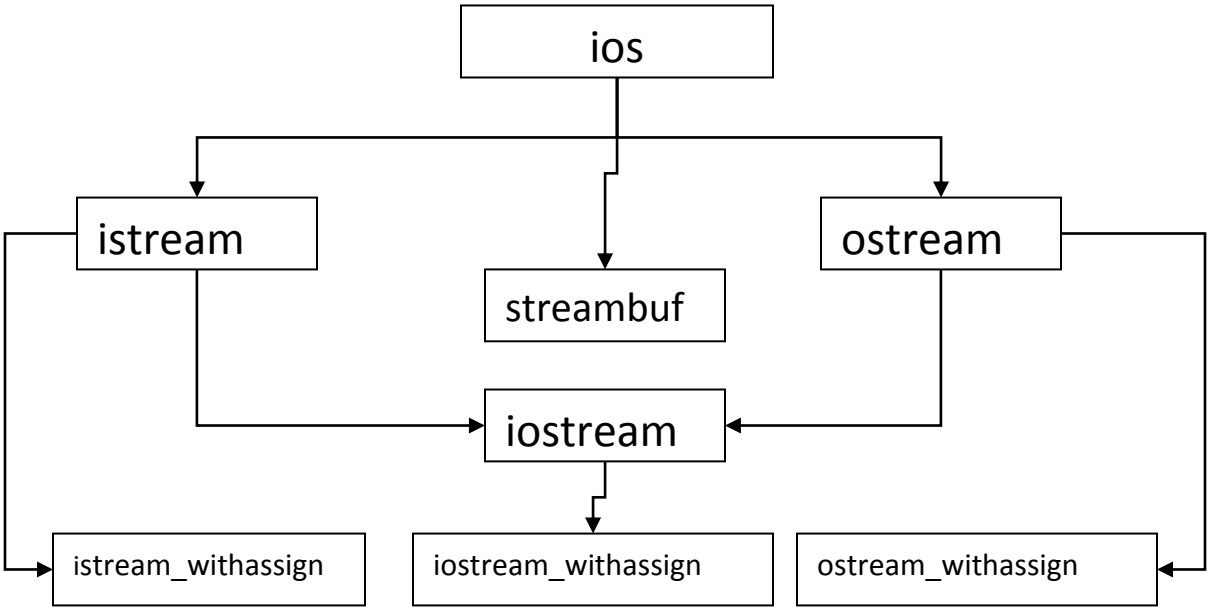
- The score stream that provide data to the program is called as the input stream.
- The destination stream that receives output from the program is called as the output stream.



- As in figure the data in the input stream can come from keyboard or any other storage device.
- Similarly the data in the output stream can go to the monitor screen or any other storage device.

5.1.2 C++ stream classes: -

The C++ input output system contain a hierarchy of classes that are used to define various stream to deal with both the console and disk disk file(text file). These class are called as stream classes.



- Figure shows the hierarchy of stream classes used for input and output operation with the console unit.
- These classes are declared in the header file iostream thus this file should be included in all the programs that communicate with the console unit.
- As in above figure ios is the virtual base class for isteam and ostream derived classes which are in turn base classes for iostream.
- The class ios provides the basic support for formatted and unformatted input output operations.
- The class istream provides the facilities for formatted and unformatted facilities for formatted and unformatted input and class ostream provides the facilities for formatted and unformatted output.
- The class iostream provides the facilities for handling both input and output streams.
- Three class namely istream_withassign, ostream_withassign and iostream_withassign add assignment operator to these classes.
- Streambuf provides as interface to physical devices through buffers.

5.1.3 Unformatted input output operator-

1] get() and put() function-

- The classes istream and ostream define two member function get( ) and put( ) respectively to hand a single character input / output operations.
- Get( ) reads a single character at a time and put( ) prints a single character at a time.
    - Syntax- cin.get(char var);
- The get( ) is called using cin. It read a single character and stored into given character variable.
    - Syntax- cout.put(char var);
- The put( ) is called by using cout. It writes a single character variable on the screen.
- Example: program to read a line of text.

```
void main()
{
 char ch;
 cout<<"enter text.";
 cin(ch);
 while (ch!="\n")
 {
  cout.put(ch);
  cin.get(ch);
 }
 getch();
}
```

output:-

```
 enter text oriented
 oriented
```

2] getline() and write() function-

- We can read and display a line of text using the line oriented input / output function getline( ) and write( ).

- The getline( ) reads a whole line of text that ends with a new line character.
    - Syntax- cin.getline(line,size);
- This function can be invoked by using the object cin.
- As in syntax this function call or invokes the function getline which reads character input into the variable line.
- The reading is terminated as soon as either the new line char '\n' is encountered or size-1 characters are read(whichever occurs first)
- The write function displays as entire line on the screen.
    - Syntax- cout.write(line,size);
- To first argument line represent the name of the string to be displayed and the second argument size indicates the number of characters to be displayed.
- It does not stop displaying the character automatically when the null character is encountered.
- If the size is greater than the length of line then it displays bounds the size of line.

Example:

```
void main()
{
 char name[25];
 cout<<"Enter name\n";
 cin.getline(name,25);
 cout<<"name=\t";
 cout.write(name,25);
 getch();
}
```

5.1.4 Formatted console input output operation-

C++ supports a number of features that could be used for formatting the output. These features includes-

1] IOS class functions and flags.

2] Manipulators.

3] User define output functions.

1] IOS class functions and flags-

A] width( )-

- We can use width( ) to define a width of the field necessary for the output of an item.
- Syntax- cout.width (w)
- Where 'w' is the field width that is number of columns.
- The output will be printed in a field of 'w' characters and at the right end of the field.
- The width( ) can specify the width for only item at a time.after printing ane item it will set back to default.
- e.g. cout.width(5);
    - cout<<543;
    - cout<<12;
- will produce the following output.

		5	4	3	1	2			

- The value 543 is printed right justified in the frist 5 columns.
- The width (5) does not keep the setting for the printing the number 12 to do this rewrite the above example as follows-

e.g. cout.width(5);

cout<<543;

cout.width(5);

cout<<12;

will produce the following output.

		5	4	3				1	2

B] precision( )

- By default floating point numbers are printed with six digits after the decimal point.
- However we can specify the number of digits to be displayed after the decimal point which printing the floating point number.
- Syntax- cout.presion(d);
- Where d is the no of digits to be displayed after the decimal point.

e.g. cout.precision(3);

cout<<3.14159;

cout<<2.4013;

cout<<2.50032;

output- 3.141

2.401

2.5 // no trailing zeros

Unlike the function width precision keeps the setting to all the item after it until it is reset.

C] Fill( )-

- We had print the valued using must large width then the specified value.

e.g. cout.width(5);

cout<12;

			1	2

- Here first three fields filled with white spaces by default.
- However we can use fill() to fill these unused positions by any described character.
- Syntax- int fill (ch);
- Where ch represents the character which is used for filling the unused positions.

e.g. cout.fill(*);

cout.width(5);

cout<<12;

*	*	*	1	2

D] Formatting flags, bit-fields and setf( )

- We have seen that when the function width( ) is used, the value is printed right-justified in the field width created. But it is a usual pratice to print the text left-justified.
- How do we get a value printed left-justified? Or, how do we get a floating-point number printed in the scientific notation?
- The setf( ), a member function of the ios class,can provide answers to these many other formatting questions. The setf( ) can be used as follows-

- Syntax- cout.setf(arg1,arg2);
- The arg1 is one of the formatting flag defined in the class ios. The formatting flag specifies the format action required for the output.
- Arg2 known as bit field specified the group to which the formatting flag belongs.

The table shows the bit fields, flags and their formatted actions.

Format required	Flag(arg1)	Bit-field(arg2)
Left-justified output	Ios :: left	Ios :: adjustfield
Right-justified output	Ios :: right	Ios :: adjustfield
Padding after sign or base Indicator(like +##20)	Ios :: internal	Ios :: adjustfield
Scientific notation	Ios :: scientific	Ios :: floatfield
Fixed point notation	Ios :: fixed	Ios :: floatfield
Decimal base	Ios :: dec	Ios :: basefield
Octal base	Ios :: oct	Ios :: basefield
Hexadecimal base	Ios :: hex	Ios :: basefield

e.g. 1] cout.setf(ios :: left, ios :: adjustified);

cout.setf(ios :: scientific, ios :: floatfield);

2] cout.fill('*');

Cout.setf(ios :: left, ios :: adjustfield);

Cout.width(15);

Cout<< "Table 1"<<"\n";

This will produce the output :-

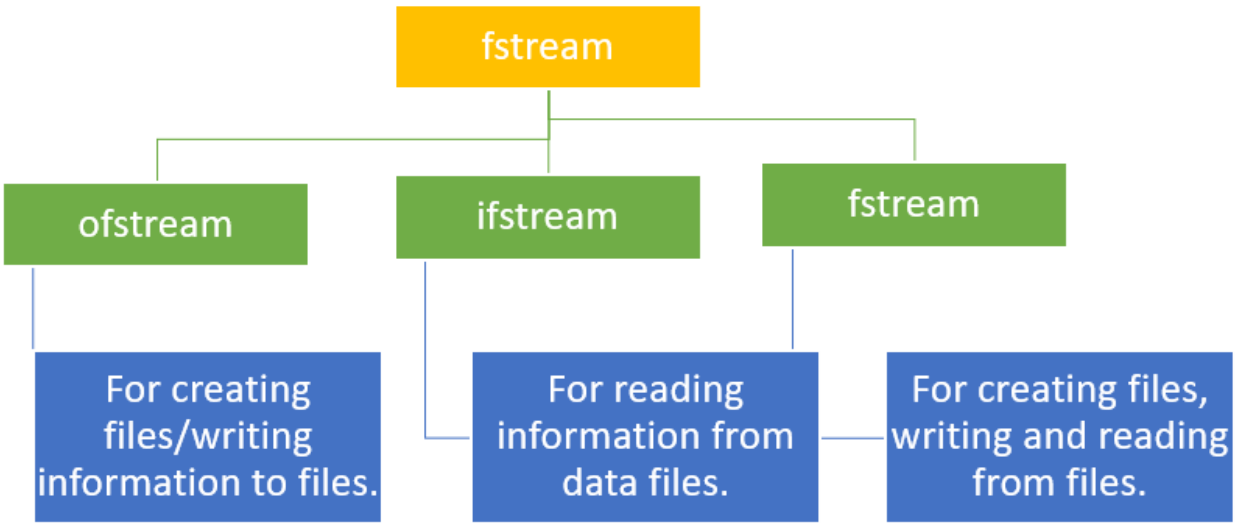
	T	A	B	L	E		1	*	*	*	*	*	*	*	*

5.2 Classes for File Stream Operations

So far, we have been using the iostream standard library, which provides cin and cout methods for reading from standard input and writing to standard output respectively.

To read and write from a file we requires another standard C++ library called fstream, which defines three new data types − fstream, ifstream, ofstream.

The fstream library provides C++ programmers with three classes for working with files. These classes include:



Ofstream: This data type represents the output file stream and is used to create files and to write information to files.

Ifstream: This data type represents the input file stream and is used to read information from files.

Fstream: This data type represents the file stream generally, and has the capabilities of both ofstream and ifstream which means it can create files, write information to files, and read information from files.

5.3 Opening File (open() function)

- 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.
- Before performing any operation on a file, you must first open it.
- If you need to **write** to the file, open it using fstream or **ofstream** objects.
- If you only need to **read** from the file, open it using the **ifstream** object.
- The three objects, that is, fstream, ofstream, and ifstream, have the **open()** function defined in them.
- open() function is a member of fstream, ifstream, and ofstream objects.
- **Open() function Syntax:**
    - **open(const char *filename, ios::mode);**
    - Here, the first argument specifies the name and location of the file to be opened
    - The second argument of the open() member function defines the mode in which the file should be opened.
- **File Opening Modes**
    - **ios::app** : Append mode. All output to that file to be appended/written at the end.
    - **ios::ate** : Open a file for output and move the read/write control to the end of the file.
    - **ios::in** : Open a file for reading.
    - **ios::out** : Open a file for writing if file not exist then new file is created.
    - **ios::trunc :** If the file already exists, its contents will be truncated before opening the file.
- It is possible to use two modes at the same time. You combine them using the | (OR) operator.
    - 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);

Example:

```
#include <iostream>
#include <fstream>
using namespace std;
int main() {
	fstream my_file;
	my_file.open("my_file", ios::out);
	if (!my_file) {
		cout << "File not created!";
	}
	else {
		cout << "File created successfully!";
		my_file.close();
	}
	return 0;
}
```

5.4 Closing a File (close() function)

- Once a C++ program terminates, it automatically
  - flushes the streams
  - releases the allocated memory
  - closes opened files.
- However, as a programmer, you should learn to close open files before the program terminates.
- The fstream, ofstream, and ifstream objects have the **close()** function for closing files.
- syntax:

  int close(FILE* stream);

This function is used to close a file stream. The data that is buffered but not written is flushed to the OS and all unread buffered data is discarded.

Example:

```
	fstream my_file;
```

```
my_file.open("my_file", ios::out);
my_file.close();
```

5.5 Detecting End of File (eof)

The eof function is used to check whether file pointer reaches to the end of file or not. This function returns true (nonzero) if end of file is encountered while reading; otherwise return false(zero).

Example:

```
ifstream myfile;
myfile.eof();
```

While reading a file, a situation can arise when we do not know the number of objects to be read from the file i.e. we do not know where the file is going to end?

A simple method of detecting end of file (eof) is by testing the stream in a while loop as shown below:

```
while (<stream>)
{
    :
}
```

The condition <stream> will evaluate to 1 as long as the end of file is not reached and it will return 0 as soon as end of file is detected.

5.6 Reading from a File

1. Reading From File using extraction operator (>>)

- You can read information from files into your C++ program. This is possible using **stream extraction operator (>>).**
- You use the operator in the same way you use it to read user input from the keyboard. However, instead of using the cin object, you use the ifstream/ fstream object.

Example:

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

int main () {
   char data[100];
   // open a file in read mode.
   ifstream infile;
   infile.open("afile.dat",”ios::in”);

   // Reading From a file
   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;
}
```

2. Reading From File using get() and getline() function:

C++ provides get( ) and getline( ) functions as input member functions of the ifstream class.

1. **get() :** It is used to read a character from a file.

   Syntax: <stream>.get(char);

Example : myfile.get(ch);

2. **getline():** It is used to read a line of text from a file.

Syntax: getline(<stream>,String);

First argument specifies the file stream object points to a file from which contents are reading. Second argument is string variable in which line of text is stored.

Example: getline (myfile, myText)

Write a C++ program to read and display contents of file. (Using get() function)

```
/*Read data From File using get() function : It Reads one char at a time from file
ifstream: open a file to read data*/
#include<iostream>
#include<conio.h>
#include<fstream>      //File Handling Header File
using namespace std;
int main()
{
        char ch;
        ifstream myfile;
        myfile.open("data.txt",ios::in);
        if(myfile)
        {
                cout<<"File Opened Successfully";
        }
        else
        {
                cout<<"File Not Present";
        }
        cout<<"Reading From File Start"<<endl;
        while(myfile)
        {
                myfile.get(ch);
                cout<<ch;
        }
```

```
        myfile.close();
return(0);
}
```

Write a C++ program to read and display line by line contents of file. (Using getline())

```
/*Read data From File using get() function : It Reads one line at a time from file
ifstream: open a file to read data*/
#include<iostream>
#include<conio.h>
#include<fstream>      //File Handling Header File
using namespace std;
int main()
{
        string str;
        ifstream myfile;
        myfile.open("data.txt",ios::in);
        if(myfile)
        {
                cout<<"File Opened Successfully";
        }
        else
        {
```

```
            cout<<"File Not Present";
        }
        cout<<"Reading From File Start"<<endl;
        while(myfile)
        {
            getline(myfile,str);
            cout<<str;
        }
        myfile.close();
return(0);
}
```

5.7 Writing to a File

1. Writing into File using extraction operator (<<)

- 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.

Example:

```
//Writing Data into file using << insertion operator
#include<iostream>
#include<conio.h>
#include<fstream>   //File Handling Header File
using namespace std;
```

```
int main()
{
      string data;
      ofstream myfile;
      myfile.open("data1.txt",ios::app);
      cout<<"Enter Your Name";
      cin>>data;
      myfile<<data<<endl;
      cout<<"Enter Your College Name";
      cin>>data;
      myfile<<data<<endl;
      cout<<"Enter Your Rollno";
      cin>>data;
      myfile<<data<<endl;
      myfile.close();
return(0);
}
```

2. Writing into File using put() function:

The put() function writes a single character to the associated stream.

Syntax: <stream>.put(char);

Example : file.put(ch);

Program: Write a program that copies contents of one file into another file. OR

Write a C++ program to append data from abc.txt to xyz.txt file.

Assuming input file as abc.txt with contents "World" and output file named as xyz.txt with contents "Hello" have been already created.

```
#include <iostream.h>
#include<fstream.h>
int main()
{
fstream f;
ifstream fin;
```

```
fin.open("abc.txt",ios::in);
ofstream fout;
fout.open("xyz.txt", ios::app);
if (!fin)
 {
 cout<< "file not found";
 }
char ch;
fin.seekg(0);
 while (fin)
 {
        fin.get(ch);
        fout.put(ch);
 }
 f.close();
 return 0;
}
```

5.8 File Pointers Manipulation

All i/o streams objects have, at least, one internal stream pointer: ifstream, like istream, has a pointer known as the get pointer that points to the element to be read in the next input operation.

ofstream, like ostream, has a pointer known as the put pointer that points to the location where the next element has to be written.

Finally, fstream, inherits both, the get and the put pointers, from iostream (which is itself derived from both istream and ostream).

These internal stream pointers that point to the reading or writing locations within a stream can be manipulated using the following member functions:

seekg() moves get pointer(input) to a specified location

seekp() moves put pointer (output) to a specified location

tellg() gives the current position of the get pointer

tellp() gives the current position of the put pointer

The other prototype for these functions is:

seekg(offset, refposition);

seekp(offset, refposition);

The parameter offset represents the number of bytes the file pointer is to be moved from the location specified by the parameter refposition. The refposition takes one of the following three constants defined in the ios class.

ios::beg start of the file

ios::cur current position of the pointer

ios::end end of the file

Example:

file.seekg(-10, ios::cur);

file.seekg(0);

Write a C++ program to count number of spaces in text file.

(Note: Any other correct logic shall be considered)

Program:

```
#include<iostream.h>
#include<conio.h>
#include<fstream.h>
void main()
{
ifstream file;
int s=0;
char ch;
clrscr();
```

```
file.open("abc.txt");
while(file)
{
file.get(ch);
 if(ch==' ')
 {
 s++;
 }
}
cout<<"\nNumber of spaces in text file are:"<<s;
getch();
}
```

Write a program to count the number of lines in file.

```
#include<iostream.h>
#include<fstream.h>
#include<conio.h>
void main()
{
ifstream file;
char ch;
int n=0;
clrscr();
file.open("abc.txt");
while(file)
{
file.get(ch);
        if(ch=='\n')
                n++;
}
cout<<"\n Number of lines in a file are:"<<n;
file.close();
getch();
}
```