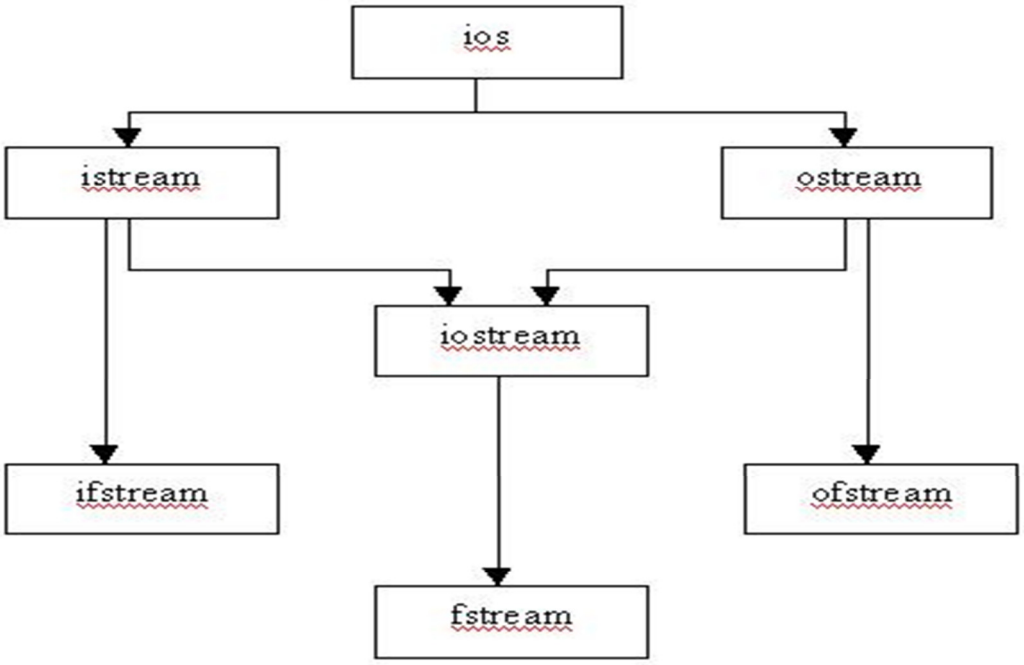
Unit-V File operations

What is stream?

- C++ IO are based on *streams*, which are sequence of bytes flowing in and out of the programs. A C++ stream is a flow of data into or out of a program, such as the data written to cout or read from cin.
- 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 handle input and output with the disk files.

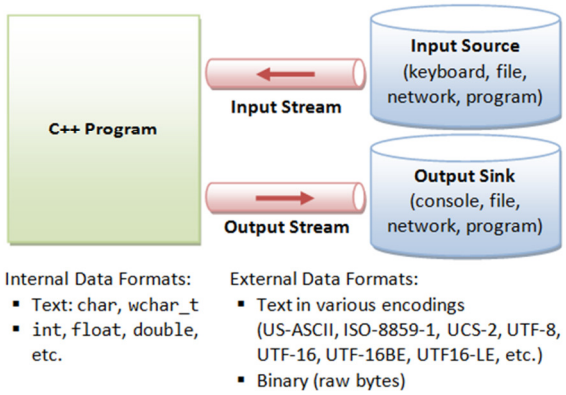
The fstream.h header file contains a declaration of ifstream, ofstream and fstream classes. The iostream.h file contains istream, ostream and iostream classes and included in the program while doing disk I/O operations.

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 arrange a space for keeping input data and for sending output data.

C++ stream classes

In input operations, data bytes flow from an *input source* (such as keyboard, file, network or another program) into the program. In output operations, data bytes flow

from the program to an *output sink* (such as console, file, network or another program). Streams acts as an intermediaries between the programs and the actual IO devices, in such the way that frees the programmers from handling the actual devices, so as to archive device independent IO operations.



C++ provides both the formatted and unformatted IO functions. In formatted or high-level IO, bytes are grouped and converted to types such as int, double, string or user-defined types. In unformatted or low-level IO, bytes are treated as raw bytes and unconverted. Formatted IO operations are supported via overloading the stream insertion (<<) and stream extraction (>>) operators, which presents a consistent public IO interface.

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.

Classes for file stream operations

In Files we store data i.e. text or binary data permanently and use these data to read or write in the form of input output operations by transferring bytes of data. So we use the term File Streams/File handling. We use the header file <fstream.h>

- **ofstream:** It represents output Stream and this is used for writing in files.
- **ifstream:** It represents input Stream and this is used for reading from files.
- **fstream:** It represents both output Stream and input Stream. So it can read from files and write to files.

Input/Output Streams

- The **iostream** standard library provides **cin** and **cout** object.

- Input stream uses **cin** object for **reading the data** from standard input and Output stream uses **cout** object for **displaying the data** on the screen or writing to standard output.
- The **cin** and **cout** are pre-defined streams for input and output data.
  Syntax:
  cin>>variable_name;
  cout<<variable_name;
- The **cin** object uses **extraction operator (>>)** before a variable name while the **cout** object uses **insertion operator (<<)** before a variable name.
- The cin object is used to read the data through the input device like keyboard etc. while the cout object is used to perform console write operation.

Operations in File Handling:

- Creating a file: `open()`
- Reading data: `read()`
- Writing new data: `write()`
- Closing a file: `close()`

Creating/Opening a File

We create/open a file by specifying new path of the file and mode of operation. Operations can be reading, writing, appending and truncating. Syntax for file creation: `FilePointer.open("Path",ios::mode);`

- Example of file opened for writing: `st.open("E:\study.txt",ios::out);`
- Example of file opened for reading: `st.open("E:\study.txt",ios::in);`
- Example of file opened for appending: `st.open("E:\study.txt",ios::app);`
- Example of file opened for truncating: `st.open("E:\study.txt",ios::trunc);`

```
#include<iostream.h>
#include<conio.h>
#include <fstream.h>

using namespace std;
```

```
int main()
{
   fstream st; // Step 1: Creating object of fstream class
   st.open("E:\study.txt",ios::out);  // Step 2: Creating new file
   if(!st) // Step 3: Checking whether file exist
   {
      cout<<"File creation failed";
   }
   else
   {
      cout<<"New file created";
      st.close(); // Step 4: Closing file
   }
   getch();
   return 0;
}
```

Writing to a File

```
#include <iostream.h>
#include<conio.h>
#include <fstream.h>

using namespace std;

int main()
{
   fstream st; // Step 1: Creating object of fstream class
   st.open("E:\study.txt",ios::out);  // Step 2: Creating new file
   if(!st) // Step 3: Checking whether file exist
   {
      cout<<"File creation failed";
   }
```

```
   else
   {
      cout<<"New file created";
      st<<"Hello";    // Step 4: Writing to file
      st.close(); // Step 5: Closing file
   }
   getch();
   return 0;
}
```

Here we are sending output to a file. So, we use ios::out. As given in the program, information typed inside the quotes after **"FilePointer <<"** will be passed to output file.

Reading from a File

```
#include <iostream.h>
#include<conio.h>
#include <fstream.h>

using namespace std;

int main()
{
   fstream st; // step 1: Creating object of fstream class
   st.open("E:\study.txt",ios::in);   // Step 2: Creating new file
   if(!st) // Step 3: Checking whether file exist
   {
      cout<<"No such file";
   }
   else
   {
      char ch;
      while (!st.eof())
      {
         st >>ch;  // Step 4: Reading from file
```

```
        cout << ch;   // Message Read from file
      }
      st.close(); // Step 5: Closing file
   }
   getch();
   return 0;
}
```

Here we are reading input from a file. So, we use ios::in. As given in the program, information from the output file is obtained with the help of following syntax **"FilePointer >>variable"**.

Close a File

It is done by FilePointer.close().

```
#include <iostream.h>
#include<conio.h>
#include <fstream.h>

using namespace std;

int main()
{
   fstream st; // Step 1: Creating object of fstream class
   st.open("E:\study.txt",ios::out);  // Step 2: Creating new file
   st.close(); // Step 4: Closing file
   getch();
   return 0;
}
```

File Modes - Reading and Writing Files

In order to open a file with a stream object open() member function is used.
open (filename, mode);
Where filename is a null-terminated character sequence of type const char * (the same

type that string literals have) representing the name of the file to be opened, and mode is an optional parameter with a combination of the following flags:

Mode	Description
ios::ate	Write all output to the end of file (even if file position pointer is moved with **seekp**)
ios::app	Open a file for output and move to the end of the existing data (normally used to append data to a file, but data can be written anywhere in the file
ios::in	The original file (if it exists) will not be truncated
ios::out	Open a file for output (default for **ofstream** objects)
ios::trunc	Discard the file's contents if it exists (this is also the default action for **ios::out**, if **ios::ate**, **ios::app**, or **ios::in** are not specified)
ios::binary	Opens the file in binary mode (the default is text mode)
ios::nocreate	Open fails if the file does not exist
ios::noreplace	Open files if the file already exists.

Detection of end of file

C++ provides a special function, eof( ), that returns nonzero (meaning TRUE) when there are no more data to be read from an input file stream, and zero (meaning FALSE) otherwise.

Rules for using end-of-file (eof( )):
1. Always test for the end-of-file condition **before** processing data read from an input file stream.
   a. use a priming input statement before starting the loop
   b. repeat the input statement at the bottom of the loop body
2. Use a while loop for getting data from an input file stream. A for loop is desirable only when you know the exact number of data items in the file, which we do not know.

Special operations in a File

There are few important functions to be used with file streams like:

- tellp() - It tells the current position of the put pointer.

  Syntax: filepointer.tellp()

- tellg() - It tells the current position of the get pointer.

Syntax: filepointer.tellg()

- seekp() - It moves the put pointer to mentioned location.

  Syntax: filepointer.seekp(no of bytes ,reference mode)

- seekg() - It moves get pointer(input) to a specified location.

  Syntax: filepointer.seekg((no of bytes, reference point)

- put() - It writes a single character to file.
- get() - It reads a single character from file.

Note: *For seekp and seekg three reference points are passed:*
ios::beg *- beginning of the file*
ios::cur *- current position in the file*
ios::end *- end of the file*

Below is a program to show importance of tellp, tellg, seekp and seekg:

```
#include <iostream.h>
#include<conio.h>
#include <fstream.h>

using namespace std;

int main()
{
   fstream st; // Creating object of fstream class
   st.open("E:\study.txt",ios::out);  // Creating new file
   if(!st) // Checking whether file exist
   {
      cout<<"File creation failed";
   }
   else
   {
      cout<<"New file created"<<endl;
      st<<"Hello Friends"; //Writing to file
```

```
    // Checking the file pointer position
    cout<<"File Pointer Position is "<<st.tellp()<<endl;

    st.seekp(-1, ios::cur); // Go one position back from current position

    //Checking the file pointer position
    cout<<"As per tellp File Pointer Position is "<<st.tellp()<<endl;

    st.close(); // closing file
  }
  st.open("E:\study.txt",ios::in);   // Opening file in read mode
  if(!st) //Checking whether file exist
  {
    cout<<"No such file";
  }
  else
  {
    char ch;
    st.seekg(5, ios::beg);  // Go to position 5 from begning.
    cout<<"As per tellg File Pointer Position is "<<st.tellg()<<endl; //Checking file pointer position
    cout<<endl;
    st.seekg(1, ios::cur); //Go to position 1 from beginning.
    cout<<"As per tellg File Pointer Position is "<<st.tellg()<<endl; //Checking file pointer position
    st.close(); //Closing file
  }
  getch();
  return 0;
}
```

New file created

File Pointer Position is 13

As per tellp File Pointer Position is 12

```
As per tellg File Pointer Position is 5
As per tellg File Pointer Position is 6
```

C++ write() function

The write() function is used to write object or record (sequence of bytes) to the file. A record may be an array, structure or class.

Syntax of write() function

```
fstream fout;
fout.write( (char *) &obj, sizeof(obj) );
```

The write() function takes two arguments.
&obj : Initial byte of an object stored in memory.
sizeof(obj) : size of object represents the total number of bytes to be written from initial byte.

Example of write() function

```
#include<fstream.h>
#include<conio.h>
class Student
{
    int roll;
    char name[25];
    float marks;
    void getdata()
    {
        cout<<"\n\nEnter Roll : ";
        cin>>roll;

        cout<<"\nEnter Name : ";
        cin>>name;

        cout<<"\nEnter Marks : ";
```

```
            cin>>marks;
        }
        public:
        void AddRecord()
        {
            fstream f;
            Student Stu;
            f.open("Student.dat",ios::app|ios::binary);
            Stu.getdata();
            f.write( (char *) &Stu, sizeof(Stu) );
            f.close();
        }
    };

    void main()
    {
        Student S;
        char ch='n';
        do
        {
            S.AddRecord();
            cout<<"\n\nDo you want to add another data (y/n) : ";
            ch = getche();
        } while(ch=='y' || ch=='Y');
        cout<<"\nData written successfully...";
    }
```

Output :

```
        Enter Roll : 1
```

```
        Enter Name : Ashish
        Enter Marks : 78.53
```

```
Do you want to add another data (y/n) : y

Enter Roll : 2
Enter Name : Kaushal
Enter Marks : 72.65

Do you want to add another data (y/n) : y

Enter Roll : 3
Enter Name : Vishwas
Enter Marks : 82.65

Do you want to add another data (y/n) : n

Data written successfully...
```

C++ read() function

The read() function is used to read object (sequence of bytes) to the file.

Syntax of read() function

```
fstream fin;
fin.read( (char *) &obj, sizeof(obj) );
```

The read() function takes two arguments.
&obj : Initial byte of an object stored in file.
sizeof(obj) : size of object represents the total number of bytes to be read from initial byte.

The read() function returns NULL if no data read.

Example of read() function

```
#include<fstream.h>
#include<conio.h>
```

```
class Student
{
	int roll;
	char name[25];
	float marks;
	void putdata()
	{

	   cout<<"\n\t"<<roll<<"\t"<<name<<"\t"<<marks;

	}
	public:
	void Display()
	{
		fstream f;
		Student Stu;
		f.open("Student.dat",ios::in|ios::binary);
		cout<<"\n\tRoll\tName\tMarks\n";
		while( (f.read((char*)&Stu,sizeof(Stu))) != NULL )
				Stu.putdata();
		f.close();
	}
};

void main()
{
	Student S;
	S.Display();
}
```

Output :

Roll	Name	Marks
1	Ashish	78.53
2	Kaushal	72.65
3	Vishwas	82.65