Paper Name:OOPS using C++
Lesson name: Working with C++ files

Unit Structure

6.1 Introduction :-

When a large amount of data is to be handled in such situations floppy disk or hard disk are needed to store the data .The data is stored in these devices using the concept of files. A file is a collection of related data stored in a particular area on a disk. Programs can be designed to perform the read and write operations on these files .The I/O system of C++ handles file operations which are very much similar to the console input and output operations .It uses file streams as an interface between the programs and files. The stream that supplies data to the program is called input stream and the one that receives data from the program is called output stream. In other words input stream extracts data from the file and output stream inserts data to the file. The input operation involves the creation of an input stream and linking it with the program and input file. Similarly, the output operation involves establishing an output stream with the necessary links with the program and output file.

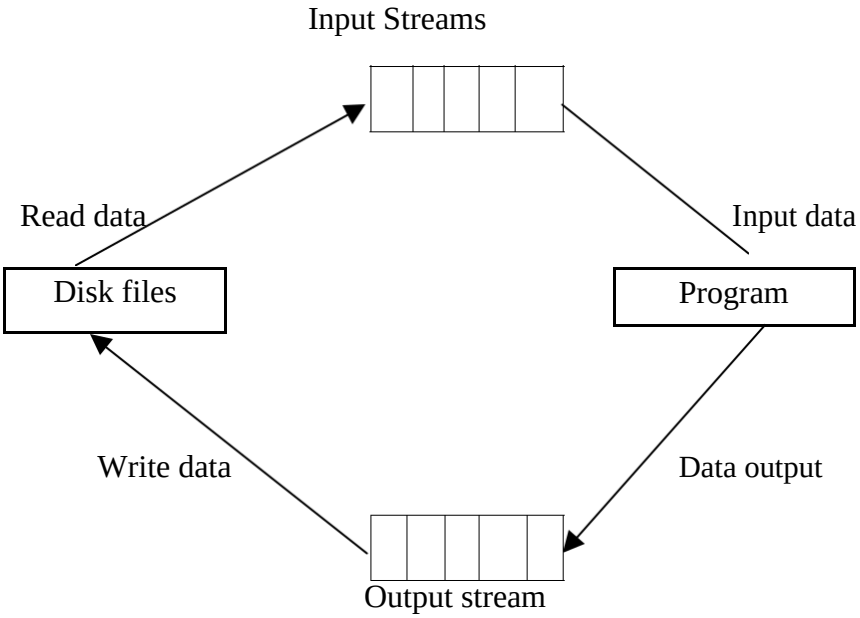

Figure 6.1 File Input and output stream

6.2 File Stream Classes:-

The I/O system of C++ contains a set of classes that defines the file handling methods. These include ifstream, ofstream and fstream.These classes are derived from fstreambase and form the corresponding iostream class. These classes ,designed to manage the disk files are declared in fstream and therefore this file is included in any program that uses files.

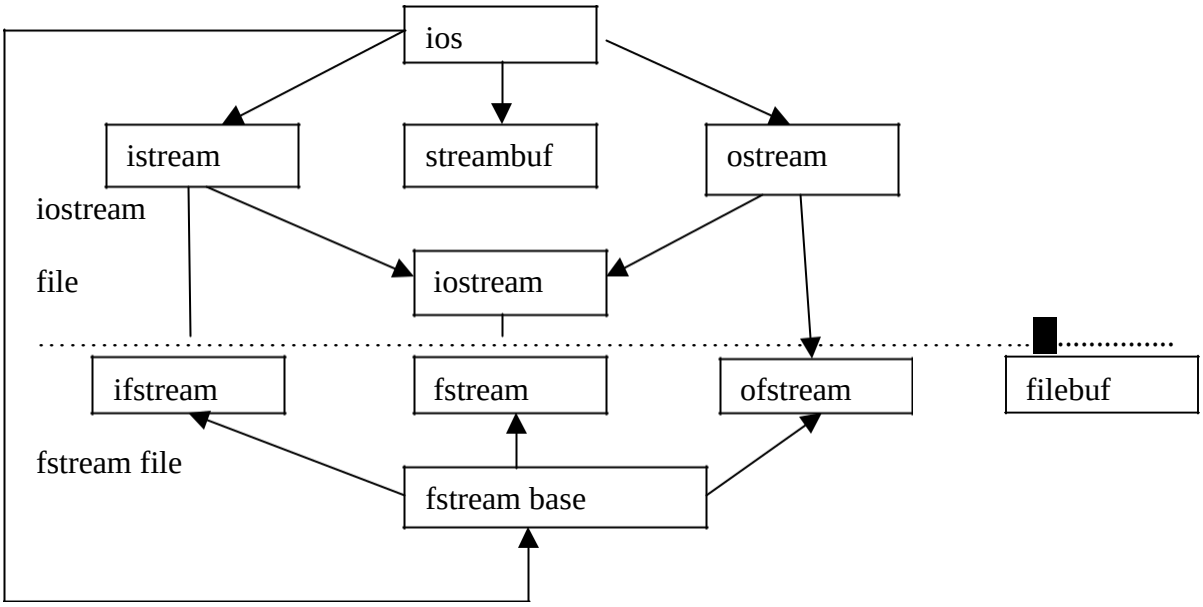

Figure 6.2 Stream classes for file operations

6.3 Steps of File Operations:-

For using a disk file the following things are necessary

1. Suitable name of file
2. Data type and structure
3. Purpose
4. Opening Method

Table 6.1 Detail of file stream classes

Class	Contents
filebuf	Its purpose is to set the file buffers to read and write. Contains **Openprot** constant used in the **open()** of file stream classes.Also contain **close()** and **open()** as members.
fstreambase	Provides operations common to file streams.Serves as a base for **fstream,ifstream** and **ofstream** class.Contains **open()** and **close()** functions.
ifstream	Provides input operations.Contains **open()** with default input mode.Inherits the functions **get(),getline(),read(),seekg(),tellg()** functions from istream.
ofstream	Provides output operations.Contains **open()** with default output mode.Inherits **put(),seekp(),tellp()** and **write()** functions from **ostream**.
fstream	Provides support for simultaneous input and output operations.Contains open with default input mode.Inherits all the functions from **istream** and **ostream** classes through **iostream**.

The filename is a string of characters that makeup a valid filename for the operating system. It may contain two parts ,primary name and optional period with extension. Examples are Input.data, Test.doc etc. For opening a file firstly a file stream is created and then it is linked to the filename.A file stream can be defined using the classes ifstream**, ofstream** and fstream that contained in the header file **fstream**.The class to be used depends upon the purpose whether the write data or read data operation is to be performed on the file.A file can be opened in two ways:

(a) Using the constructor function of class.

(b) Using the member function **open()** of the class.

The first method is useful only when one file is used in the stream.The second method is used when multiple files are to be managed using one stream.

6.3.1Opening Files using Constructor:

While using constructor for opening files,filename is used to initialize the file stream object.This involves the following steps

(i) Create a file stream object to manage the stream using the appropriate class

i.e the class ofstream is used to create the output stream and the class **ifstream** to create the input stream.

(ii) Initialize the file object using desired file name.

For example, the following statement opens a file named "results" for output:

ofstream outfile("results"); //output only

This create outfile as an **ofstream** object that manages the output stream. Similarly ,the following statement declares infile as an **ifstream** object and attaches it to the file data for reading (input).

ifstream infile ("data"); //input only

The same file name can be used for both reading and writing data.For example

Program1

……..

…….

ofstream outfile ("salary"); //creates outfile and connects salary to it

……

……..

Program 2

……

……

ifstream infile ("salary"); //creates infile and connects salary to it

……..

…….

The connection with a file is closed automatically when the stream object expires i.e when a program terminates.In the above statement ,when the program 1 is terminated,the salary file is disconnected from the outfile stream.The same thing happens when program 2 terminates.

Instead of using two programs,one for writing data and another for reading data ,a single program can be used to do both operations on a file.

……

…….

outfile.close(); //disconnect salary from outfile and connect to infile

ifstream infile ("salary");

…….

…….

infile.close();

The following program uses a single file for both reading and writing the data .First it take data from the keyboard and writes it to file.After the writing is completed the file is closed.The program again opens the same file read the information already written to it and displays the same on the screen.

PROGRAM 6.1

WORKING WITH SINGLE FILE

```
//Creating files with constructor function
#include <iostream.h>
#include <fstream.h>
int main()
{
      ofstream outf("ITEM");
     cout <<"enter item name: ";
    char name[30];
   cin >>name;
  outf <<name <<"\n";
```

```
  cout <<"enter item cost :";
 float cost;
cin >>cost;
outf <<cost <<"\n";
outf.close();
ifstream inf("item");
inf >>name;
inf >>cost;
cout <<"\n";
cout <<"item name : " << name <<"\n";
cout <<"item cost: " << cost <<"\n";
inf.close();
return 0;
}
```

6.3.2 Opening Files using open()

The function **open()** can be used to open multiple files that uses the same stream object. For example to process a set of files sequentially,in such case a single stream object can be created and can be used to open each file in turn. This can be done as follows;

File-stream-class stream-object;

stream-object.**open** ("filename");

The following example shows how to work simultaneously with multiple files

PROGRAM 6.2

```
WORKING WITH MULTIPLE FILES
//Creating files with open() function
#include <iostream.h>
#include<fstream.h>
int main()
```

```
{
  ofstream fout;
 fout.open("country");
fout<<"United states of America \n";
fout<<"United Kingdom";
fout<<"South korea";
fout.close();
fout.open("capital");
fout<<"Washington\n";
fout<<"London\n";
fout<<"Seoul \n";
fout.close();
const int N =80;
char line[N];
ifstream fin;
fin.open("country");
cout<<"contents of country file \n";
while (fin)
{
fin.getline(line,N);
cout<<line;
}
fin.close();
fin.open("capital");
cout<<"contents of capital file";
while(fin)
{
```

```
 fin.getline(line,N);
cout<<line;
}
fin.close();
return 0;
}
```

6.4 Finding End of File:

While reading a data from a file,it is necessary to find where the file ends i.e end of file.The programmer cannot predict the end of file,if the program does not detect end of file,the program drops in an infinite loop.To avoid this,it is necessary to provide correct instruction to the program that detects the end of file.Thus when end of file of file is detected,the process of reading data can be easily terminated. An **ifstream** object such as **fin** returns a value of 0 if any error occurs in the file operation including the end-of –file condition.Thus the while loop terminates when f**in** returns a value of zero on reaching the end-of –file condition.There is an another approach to detect the end of file condition.The statement

```
if(fin1.eof() !=0 )
 {
     exit(1);
  }
```

returns a non zero value if end of file condition is encountered and zero otherwise.Therefore the above statement terminates the program on reaching the end of file.

6.5 File Opening Modes:

The **ifstream** and **ofstream** constructors and the function **open()** are used to open the files.Upto now a single arguments a single argument is used that is filename.However,these functions can take two arguments, the second one for specifying the file mode.The general form of function open() with two arguments is:

stream-object.open(“filename”,mode);

The second argument mode specifies the purpose for which the file is opened.The prototype of these class member functions contain default values for second argument and therefore they use the default values in the absence of actual values.The default values are as follows :

ios::in for **ifstream** functions meaning open for reading only.

ios::out for **ofstream** functions meaning open for writing only.

The file mode parameter can take one of such constants defined in class **ios**.The following table lists the file mode parameter and their meanings.

Table 6.2 File Mode Operation

Parameter	**Meaning**
ios::app	Append to end-of-file
ios::ate	Go to end-of-file on opening
ios::binary	Binary file
ios::in	Open file for reading only
ios::nocreate	Open fails if file the file does not exist
ios::noreplace	Open fails if the file already exists
ios::out	Open file for writing only
ios::trunk	Delete the contents of the file if it exists

6.6 File Pointers and Manipulators:

Each file has two pointers known as file pointers,one is called the input pointer and the other is called output pointer..The input pointer is used for reading the contents of of a given file location and the output pointer is used for writing to a given file location.Each time an input or output operation takes place,the appropriate pointer is automatically advanced.

6.6.1 Default actions:

When a file is opened in read-only mode,the input pointer is automatically set at the beginning so that file can be read from the start.Similarly when a file is opened in write-only mode the existing contents are deleted and the output pointer is set at the beginning.This enables us to write the file from start.In case an existing file is to be opened in order to add more data,the file is opened in ‘append’ mode.This moves the pointer to the end of file.

6.6.2 Functions for Manipulations of File pointer:

All the actions on the file pointers takes place by default.For controlling the movement of file pointers file stream classes support the following functions

- **seekg()** Moves get pointer (input)to a specified location.

- **seekp()** Moves put pointer (output) to a specified location.
- **tellg()** Give the current position of the get pointer.
- **tellp()** Give the current position of the put pointer.

For example, the statement

infile.seekg(10);

moves the pointer to the byte number 10.The bytes in a file are numbered beginning from zero.Therefore ,the pointer to the 11^{th} byte in the file.Consider the following statements:

ofstream fileout;

fileout.open("hello",ios::app);

int p=fileout.tellp();

On execution of these statements,the output pointer is moved to the end of file "hello"

And the value of p will represent the number of bytes in the file.

6.6.3 Specifying the Offset:

'Seek' functions **seekg()** and **seekp()** can also be used with two arguments as follows:

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 file
- **ios::cur**Current position of the pointer
- **ios::end** End of file

The **seekg()** function moves the associated file's 'get' pointer while the **seekp()** function moves the associated file's 'put 'pointer.The following table shows some sample pointer offset calls and their actions.**fout** is an **ofstream** object.

Table 6.3 Pointer offset calls

Seek call	Action
fout.seekg(o,ios::beg)	Go to start
fout.seekg(o,ios::cur)	Stay at the current position
fout.seekg(o,ios::end)	Go to the end of file
fout.seekg(m,ios::beg)	Move to (m+1)th byte in the file
fout.seekg(m,ios::cur)	Go forward by m byte from current position
fout.seekg(-m,ios::cur)	Go backward by m bytes from current position.
fout.seekg(-m,ios::end)	Go backward by m bytes from the end

6.7 Sequential Input and Output Operations:

The file stream classes support a number of member functions for performing the input and output operations on files.One pairs of functions,**put()** and **get()** are designed for handling a single character at a time.Another pair of functions, **write(),read()** are designed to write and read blocks of binary data.

6.7.1 put() and get() Functions:

The function **put()** writes a single character to the associated stream.Similarly,the function **get()** reads a single character from the associated stream.The following program illustrates how the functions work on a file.The program requests for a string.On receiving the string,the program writes it,character by character,to the file using the **put()** function in a for loop.The length of string is used to terminate the for loop.

The program then displays the contents of file on the screen.It uses the function **get()** to fetch a character from the file and continues to do so until the end –of –file condition is reached.The character read from the files is displayed on screen using the operator <<.

PROGRAM 6.3

I/O OPERATIONS ON CHARACTERS

```
#include <iostream.h>

#include <fstream.h>

#onclude<string.h>

int main()
```

```
{
  char string[80];
  cout<<"enter a string \n";
   cin>>string;
  int len =strlen(string);
  fstream file;
   file.open("TEXT". Ios::in | ios::out);
   for (int i=o;i<len;i++)
   file.put(string[i]);
    file .seekg(0);
    char ch;
     while(file)
     {
          file.get(ch);
           cout<<ch;
      }
       return 0;
 }
```

6.7.2 write() and read () functions:

The functions **write()** and **read(),**unlike the functions **put()** and **get()** ,handle the data in binary form.This means that the values are stored in the disk file in same format in which they are stored in the internal memory.An int character takes two bytes to store its value in the binary form,irrespective of its size.But a 4 digit int will take four bytes to store it in the character form.The binary input and output functions takes the following form:

infile.read ((char *) & V,sizeof (V));

outfile.write ((char *) & V ,sizeof (V));

These functions take two arguments.The first is the address of the variable V, and the second is the length of that variable in bytes.The address of the variable must be cast to type char*(i.e pointer to character type).The following program illustrates how these

two functions are used to save an array of floats numbers and then recover them for display on the screen.

PROGRAM 6.4

```
// I/O OPERATIONS ON BINARY FILES
 #include <iostream.h>
#include <fstream.h>
#include <iomanip.h>
const char * filename =”Binary”;
int main()
{
  float height[4] ={ 175.5,153.0,167.25,160.70};
ofstream outfile;
outfile.open(filename);
outfile.write((char *) & height,sizeof(height));
outfile.close();
 for (int i=0;i<4;i++)
height[i]=0;
ifstream infile;
infile.open(filename);
infile.read ((char *) & height,sizeof (height));
for (i=0;i<4;i++)
{
    cout.setf(ios::showpoint);
cout<<setw(10)<<setprecision(2)<<height[i];
}
infile.close();
return 0;
}
```

6.8 Error Handling during File Operations:

There are many problems encounterd while dealing with files like

- a file which we are attempting to open for reading does not exist.
- The file name used for a new file may already exist.
- We are attempting an invalid operation such as reading past the end of file.
- There may not be any space in the disk for storing more data.
- We may use invalid file name.
- We may attempt to perform an operation when the file is not opened for that purpose.The C++ file stream inherits a ‘stream-state ‘member from the class ios.This member records information on the status of a file that is being currently used.The stream state member uses bit fields to store the status of error conditions stated above.The class ios support several member functions that can be used to read the status recorded in a file stream.

  Table 6.4 Error Handling Functions

Function	**Return value and meaning**
eof()	Returns true(non zero value) if end of file is encountered while reading otherwise returns false(zero).
fail()	Returns true when an input or output operation has failed .
bad()	Returns true if an invalid operation is attempted or any unrecoverable error has occurred.However,if it is false,it may be possible to recover from any other error reported and continues operation.
good()	Returns true if no error has occurred.This means all the above functions are false.For instance,if file.good() is true.all is well with the stream file and we can proceed to perform I/O operations.When it returns false,no further operations is carried out.

These functions can be used at the appropriate places in a program to locate the status of a file stream and thereby to take the necessary corrective measures.Example:

……

……..

ifstream infile;

```
infile.open("ABC");
while(!infile.fail())
{
……
…….. (process the file)
…….
}
if (infile.eof())
{
……(terminate the program normally)
}
else
  if (infile.bad())
{
     …….(report fatal error)
}
else
{
 infile.clear(); //clear error state
…….
…….
}
……..
……..
```

The function **clear()** resets the error state so that further operations can be attempted.

6.9 Command Line Arguments:-

Like C,C++ also support the feature of command line argument i.e passing the arguments at the time of invoking the program.They are typically used to pass the names of data files.Example:

C>exam data results

Here exam is the name of file containing the program to be executed and data and results are the filenames passed to program as command line arguments.The command line arguments are typed by the user and are delimited by a space.The first argument is alwayas the filename and contains the program to be executed.The **main()** functions which have been using upto now without any argument can take two arguments as shown below:

main(int argc,char * argv[])

The first argument **argc** repesents the number of argumnts in commandline.The second argument **argv** is an array of character type pointers that points to the the command line arguments.The size of this array will be equal to the value of argc.For instance,for command line

C>exam data results

The value of **argc** would be 3 and the **argv** would be an array of three pointers to string as shown:

argv[0] exam

argv[1] data

argv[2] results

The argv[0] alwayas represents the command name that invokes the program.The character pointer argv[1], and argv[2] can be used as file names in the file opening statements as shown:

……

……

inline.open(argv[1]); //open data file for reading

……..

……..

outfile.open(argv[2]); //open result file for writing

……..

……..

Summary

1.Stream is nothing but flow of data .In object oriented programming the streams are controlled using classes.

2.The istream and ostream classes control input and output functions respectively.

3.The iostream class is also a derived class .It is derived from istream and ostream classes.There are three more derived classes istream_withassign,ostream_withassign and iostream_withassign.They are derived from istream,ostream and iostream respectively.

4.There are two methods constructor of class and member function open() of the class for opening the file.

5.The class ostream creates output stream objects and ifstream creates input stream objects.

6.The close() member function closes the file.

7.When end of file is detected the process of readind data can be easily terminated.The eof() function is used for this purpose.The eof() stands for end of file.The eof() function returns 1 when end of file is detected.

8.The seekg () functions shifts the associated file's input file pointer and output file pointer.

9.The put() and get() functions are used for reading and writing a single character whereas write() and read() are used to read or write block of binary data.

Key Terms

argv	ios::in
clear()	ios::out
eof()	iostream
fail()	ofstream
filemode	open()
filebuf	put()
get()	read()
seekg()	seekp()

Exercises

6.1.What are input and output streams?

6.2.What are the various classes available for file operations.

6.3. What is a file mode ?describe the various file mode options available.

6.4.Describes the various approaches by which we can detect the end of file condition.

6.5.What do you mean by command line arguments?