CHAPTER 7

FILES

Structure of the Unit

- File Stream Operations
- File I/O
- Sequential Input And Output Operations
- Random Access Files
- Input And Output Operations With Binary Files
- Reading And Writing Objects In A Binary File
- Error Handling In File Operations

7.1 Introduction

We all know C++ I/O operates on streams hence it is very important to know about streams supported by C++. Regarding streams this chapter starts the discussions from knowing the hierarchy of C++ I/O classes. You will be learning handling sequential files, random access files and binary files in detail. This chapter introduces you to write programs in C++ to handle the above mentioned files. Finally the chapter also discusses the error handling mechanisms supported by c++ when working with files.

7.2 Learning Objectives

- To introduce file stream operations
- To introduce file I/O
- To discuss sequential input and output operations
- To present the concept of random access files
- To show how input output operations are performed on binary files
- To introduce error handling mechanisms in files.

7.3 File Stream Operations

C++ I/O system operates on streams. A stream is a common, logical interface to various devices of a computer system. A stream either produces or consumes information, and is linked to a physical device by the C++ I/O system. All streams behave in the same manner.

There are two types of streams: text and binary. A text stream is used with characters. When a text stream is being used, some character translations may take place (e.g. newline to carriage-return/linefeed combination). A binary stream uses binary format for representing data in the memory. A binary stream can be used with any type of data. No character translation will occur. Thus always the binary file contains exact data sent by the stream.

C++ contains several predefined streams that are automatically opened. These are cin, cout, cerr and clog. By default, cin is linked to the keyboard, while the others are linked to the screen. The difference between cout, cerr and clog is, that cout is used for normal" output, cerr and clog are used for error messages.

The I/O system of C++ has many classes and file handling functions. All these classes are derived from the base class ios.The hierarchy of C++ I/O classes are given below.

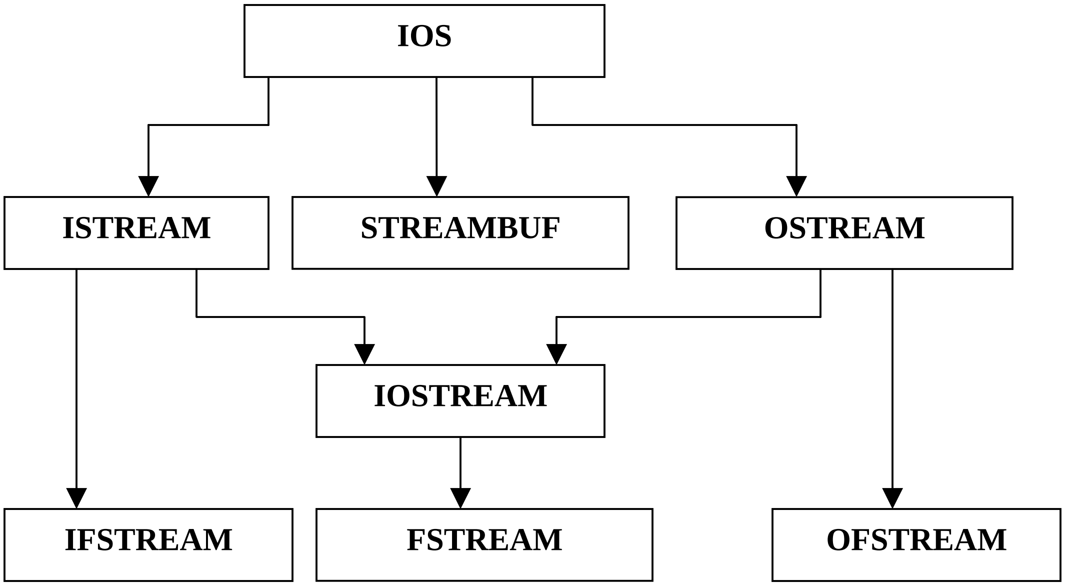

Fig.7.1. Hierarchy of C++ I/O classes

Have you Understood Questions?

1. What is a stream ?
2. What are the two types of streams?

7.4 File I/O

File I/O and console I/O are closely related. To perform file I/O, you must include <fstream> in your program. It defines several classes, including ifstream, ofstream and fstream. These classes are derived from ios, so ifstream, ofstream and fstream have access to all operations defined by ios. In C++, a file is opened by linking it to a stream. There are three types of streams: input, output and input/output. Before you can open a file, you must first obtain a stream.

- To create an input stream, declare an object of type **ifstream**.
- To create an output stream, declare an object of type **ofstream**.
- To create an input/output stream, declare an object of type **fstream**.

Examples

ofstream out(“emp.dat”);   //Opens emp.dat file in output mode
ifstream in(“emp.dat”);   //Opens emp.dat file in input mode
fstream fin(“emp.dat”);   //Opens emp.dat file in input/output mode

The prototype of each member function is given below.

- void ifstream::open(const char * *filename*,openmode)
- void ifstream::open(const char * *filename*,openmode)
- void ifstream::open(const char * *filename*,openmode)

Here filename is the name of the file to be processed, the filename also includes the path specifier. The value of the mode specifies how the file has to be opened.C++ supports the following file opening modes

- ios::app
- ios::ate
- ios::binary
- ios::in
- ios::out
- ios::trunk

Note that the file opening modes can be combined using | symbol. For example
ofstream out(“emp.dat” ,ios::out | ios::trunk)

- **ios::app**: causes all output to that file to be appended to the end. Only with files capable of output.
- **ios::ate**: causes a seek to the end of the file to occur when the file is opened.
- **ios::out**: specify that the file is capable of output.
- **ios::in**: specify that the file is capable of input.
- **ios::binary**: causes the file to be opened in binary mode. By default, all files are opened in text mode. In text mode, various character translations might take place, such as carriage return/linefeed sequences being converted into newlines. However, when a file is opened in binary mode, no such character translations will occur

- **ios::trunc**: causes the contents of a pre-existing file by the same name to be destroyed and the file to be truncated to zero length. When you create an output **ios::trunc**: causes the contents of a pre-existing file by the same name to be destroyed and the file to be truncated to zero length. When you create an output stream using **ofstream**, any pre-existing file is automatically truncated.

Once the file is opened we can read from or write into the file based on the mode you have opened. After processing the file it has to be closed. To close a file we have to use the member function called close, for example

```
in.close()  //this will close file pointed by the stream in.
```

The close function will not take any parameters and will not return any value.

Example 1:

The following program writes name and rollno into a file called student.

```
#include<iostream.h>
#include<fstream.h>
int main()
{
        ofstream out(“Student”); //open the file student in output mode
        char name[30];
        int rollno;
        cout<<”ENTER YOUR NAME”<<endl;
        cin>>name;
        cout<<”ENTER YOUR ROLL NO”<<endl;
        cin>>rollno;
        out<<name <<”\t” ; // write name to the file student
        out<<rollno;        // write rollno to the file student
        out.close();
}
```

Output of the Program

```
ENTER YOUR NAME
MOHAMED
ENTER YOUR ROLL NO
1101
```

To view the outputs from file go to command prompt and type the command given below.

```
c:\tc\bin>type student (press enter key)

AHAMED      1101
```

In this example we have opened the file student in output mode. Since the *openmode* parameter to open( ) defaults to a value appropriate to the type of stream being opened, there is no need to specify its value in the preceding example.

Example 2:

The following example reads the content of the file student.

```
#include<iostream.h>
#include<fstream.h>
int main()
{
        ifstream in(“Student”); //open the file student in input mode
        char name[30];
        int rollno;
        in>>name; // read name from the file student
        in>>rollno; // read rollno from the file student
        cout<<”NAME IS “<<name<<endl;
        cout<<”ROLL NO IS “<<rollno ;
        in.close();
}
```

Output of the Program

```
NAME IS MOHAMED
ROLL NO 1101
```

In the preceding examples we are working with one file at a time. Now lets learn now let us learn how to work with multiple files.

Example 3:

This example works with more than one file.

```
#include<iostream.h>
#include<fstream.h>
#include<iomanip.h>
int main()
{
        ofstream out;
        out.open(“book”);

        out<<”C++”;
        out<<”JAVA”;
        out.close();

        out.open(“author”);
        out<<”H.SCHILDT”;
        out<<”BALAGURUSAMY”;
        out.close();

        out.open(“publisher”);
        out<<”OSBORNE”;
        out<<”TATA MCGRAW”;
        out.close();

        ifstream in;
        char disp[100];   //create a buffer to store the data  read from the file
        in.open(“book”);

        cout<<”BOOK DETAILS “<<endl;
        while(in)  //iterate till end of file
        {
                getline(in,disp); //read a line from the file and put in the buffer
                cout<<disp;
        }
        in.close();

        cout<<”AUTHOR DETAILS”<<endl;
        in.open(“author”);
        while(in)  //iterate till end of file
        {
                getline(in,disp); //read a line from the file and put in the buffer
                cout<<disp;
        }
        in.close();

        cout<<”PUBLISHER DETAILS”<<endl;
        in.open(“publisher”);
        while(in)  //iterate till end of file
        {
                getline(in,disp);  //read a line from the file and put in the buffer
                cout<<disp;
        }
        in.close();

        return 0;
}
```

Output of the Program

```
BOOK DETAILS
C++ JAVA

AUTHOR DETAILS
H.SCHILDT  BALAGURUSAMY


PUBLISHER DETAILS
OSBORNE TATA MCGRAW
```

Here three files book, author, publisher are opened in output mode and the contents are written and then same files are opened in input mode, the contents are read from the file and displayed.

In this example we have used the getline function, which is used to read a line from the file pointed by the stream and store in a buffer .The second parameter in the getline function indicates buffer name.

Have you Understood Questions?

1. List some classes available in fstream header file?
2. if we want have a input and output operations on a file which type of object we create?
3. Write any two file opening modes?

7.5 Sequential Input And Output Operations

Sequential input and output operations are performed on sequential files. A sequential file is one in which every record is accessed serially, in the order in which they are written hence to process any i^{th} record all the previous i-1 records has to be processed.

In C++ every sequential file (including Random access file) will be associated with two file pointers. They are

- get() pointer
- put() pointer

The get pointer is an input pointer; it is used to read the content of the file. The get pointer will point content of the file to be read. When the file is opened in input mode the get pointer will point the first location of the file.

The put pointer is an output pointer; it is used to write content to the file. The put pointer will point to location where the content has to be written. When the file is opened in output mode the put pointer will point the first location of the file. However when the file is opened in append mode (ios::app) the put pointer will point the last location of the file.

Example 4:

This example program uses put pointer to write content to the sequential file.

```
#include<iostream.h>
#include<fstream.h>
int main()
{
        fstream out(“test”,ios::out); //opens “test” file in output mode
        cout<<”ENTER A STRING AT END PRESS #“<<endl;
        do
        {
                cin.get(ch);
                out.put(ch); //write a character to the file
         } while (ch! ='#')
        out.close();
}
```

Output of the Program

```
ENTER A STRING AT END PRESS#
ANNA UNIVERSITY#

c:\tc\bin>type test
ANNA UNIVERSITY#
```

Example 5:

This is an example program that uses get pointer to read content from the sequential file.

```
#include<iostream.h>
#include<fstream.h>
int main()
{
        fstream in (“test”,ios::in); //opens “test” file in input mode
        char ch;
        cout<<”READING CONTENT FROM THE FILE….“<<endl;
        while (in)
        {
                in.get(ch); //get a character to the file
                cout<<ch;
         }
        in.close();
}
```

Output of the Program

```
READING CONTENT FROM THE FILE….
ANNA UNIVERSITY#
```

Have you Understood Questions?

1. What is sequential file ?
2. write the 2 pointers associated with sequential file ?

7.6 Random Access Files

Random access file is one that allows accessing records randomly in any order hence any ith record can be accessed directly. In order to perform random access to a file C++ supports the following functions in addition to get() and put() pointers

seekg()	Moves get pointer to specified position
seekp()	Moves put pointer to specified position
tellg()	Returns the position of get pointer
tellp()	Returns the position of get pointer

The tell pointer is use to give the current position of the file. The functions tellg() and tellp() have the following prototype

- position tellg()
- position tellp()

Here position is the integer value that is capable of holding the largest value that defines the file position.

The functions seekg() and seekp() have the following prototype

- istream &seekg(offset,seekdirection);
- ostream &seekp(offset,seekdirection);

From the prototype we can note that both the functions have the same form. The first parameter offset specifies the number of bytes the file pointer has to move from the specified position. The second parameter seekdirection may take any one of the following values.

ios::beg	seek from the beginning
ios::cur	seek from the current position
ios::end	seek from end.

Consider the following examples

SEEK	POSITION OF THE FILE POINTER
in.seekg(0,ios::beg)	Moves the get pointer to the beginning of file pointed by the stream in
in.seekg(0,ios::end)	Moves the get pointer to the end of file pointed by stream in
in.seekg(10,ios::cur)	Moves the get pointer 10 bytes ahead from the current position of the file pointed by stream in
in.seekg(-k,ios::end)	Moves the get pointer m bytes before from the end of the file pointed by stream in
out.seekp(k,ios::beg)	Moves the put pointer k bytes ahead from the beginning of the file pointed by stream out

out.seekp(-k,ios:;end)	Moves the put pointer k bytes before the end of the file pointed by stream out

Example 6:

The following program opens a random access file in input and output mode and it will allow accessing the specified character in the file.

```
#include<iostream.h>
#include<fstream.h>
int main()
{
        fstream in(“test”,ios::in | ios :: out); //opens “test” file in input/output  mode
        char ch;
        int pos;
        cout<<”ENTER A STRING AT END PRESS #“<<endl;
        do
        {
                cin.get(ch);
                in.put(ch); //write a character to the file
         } while (ch! ='#');
        in.seekg(0) ;  // Goto the beginning of the file;
        cout<<”READING CONTENT FROM THE FILE…“<<endl;
        while (in)
        {
                out.get(ch); //get a character to the file
                cout<<ch;
         }
        cout<<”ENTER THE POSITION OF THE FILE TO READ”;
        cin>>pos;
        in.seekg(pos,ios::beg); //MOVES THE GET  POINTER TO THE POSITION
        while (in)
        {
                out.get(ch); //get a character to the file
                cout<<ch;
         }
        in.close();
}
```

Output of the Program

```
ENTER A STRING AT END PRESS#
ANNA UNIVERSITY#
READING CONTENT FROM THE FILE…
ANNA UNIVERSITY#
ENTER THE POSITION OF THE FILE TO READ 2
N
```

Have you Understood Questions?

1. What is a random access fle?
2. Write the two seek pointers available in random access files.
3. what do you mean by out.seekp(k,ios::beg) ?

7.7 Input and Output Operations with Binary Files

As mentioned earlier a binary file stores the content in binary format and a binary file can be used to represent any kind of data. Similar to the get and put functions of the text file we have read and write functions in a binary file. The syntax of read and write functions is given below.

```
        istream &read(char *buf, streamsize num);
        ostream &write(const char *buf, streamsize num);
```

The read( ) function reads *num* bytes from the stream and puts them in the buffer pointed to by *buf*. The write( ) function writes *num* bytes to the associated stream from the buffer pointed by *buf*. The streamsize type is some form of integer. An object of type streamsize is capable of holding the largest number of bytes that will be transferred in any I/O operation.

Example 7:

This program writes an integer number into a binary file.

```
#include<iostream.h>
#include<fstream.h>
int main()
{
        ofstream out;
        out.open(“number”,ios::binary); //opens a binary file
        int k=55;
        out.write((char *)&k,sizeof(k)); //writes integer to the file
        out.close();
        return 0;
}
```

In this example a binary file stream is created by specifying ios::binary in the open statement. To write an integer to the file we have used write function, the first parameter to the function is the address of the variable k and the second is the length in bytes.

Note:
The address of the variable must be casted to the type char *.

Example 8:

This program reads an integer number from the binary file.

```
#include<iostream.h>
#include<fstream.h>
int main()
{
        ifstream in;
        in.open("number",ios::binary); //opens a binary file
        int k;
        in.read((char *) &k,sizeof(k));
        cout<<k;
        out.close();
        return 0;
}
```

Output of the Program

```
55
```

7.7.1 Reading and Writing Objects in a Binary File

We can write a object to a binary file as we do with the primitive data type like int, float etc. Similarly we can also read an object from the binary file. We will still use the read() and write() functions to perform read and write operations. The program given below illustrates reading and writing an object from the binary file.

Example 9:

```
#include<iostream.h>
#include<fstream.h>
class book
{
        char bname[40];
        float cost;
        char aname[40];
        int pubid;
        public:
                void getdetails();
                void printdetails();
};
void book :: getdetails()
{
        cout<<”ENTER BOOK NAME”<<endl;
        cin>>bname;
        cout<<”ENTER AUTHOR NAME”<<endl;
        cin>>aname
        cout<<”ENTER COST OF THE BOOK”<<endl;
        cin>>cost;
        cout<<”ENTER PUBLISHER ID”<<endl;
        cin>>pubid;
}

void book :: printdetails()
{
        cout<<”BOOK NAME IS ”<<bname<<endl;
        cout<<”AUTHOR NAME IS”<<aname<<endl;
        cout<<”COST OF THE BOOK IS ”<<cost<<endl;
        cout<<”PUBLISHER ID IS ”<<pubid<<endl;
}
int main()
{
        fstream fin;
        fin.open(“book”,ios::in||ios::out);
        int i;

        book ptr[2];
        for(i=0;i<2;i++)
        {
                ptr[i].getdetails();
                fin.write((char *) &ptr[i],sizeof(ptr[i]));
        }
        fin.seekg(0);
```

```
    cout<<”PRINTING BOOK DETAILS”<<endl;
    for(i=0;i<2;i++)
    {
        fin.read((char *) &ptr[i],sizeof(ptr[i]));
        ptr[i].printdetails();
    }
    fin.close();
    return 0;
}
```

Output of the Program

```
ENTER BOOK NAME
C++
ENTER AUTHOR NAME
Balagurusamy
ENTER COST OF THE BOOK
200
ENTER PUBLISHER ID
101
ENTER BOOK NAME
JAVA
ENTER AUTHOR NAME
H.SCHILDT
ENTER COST OF THE BOOK
400
ENTER PUBLISHER ID
102
```

Output of the Program (Contd..)

```
PRINTING BOOK DETAILS
BOOK NAME IS C++
AUTHOR NAME IS Balagurusamy
COST OF THE BOOK IS 200
PUBLISHER ID IS 101
BOOK NAME IS JAVA
AUTHOR NAME IS H.SCHILDT
COST OF THE BOOK IS 400
PUBLISHER ID IS 102
```

This program creates an array of objects for the class book. The first for loop in the main program reads the details of 2 books. Note that we still use the write() function with the same syntax to write the object to the file. The second for loop reads the content from the file and displays the details.

Have you Understood Questions?

1. How will you perform read/write operations in binary files?

7.8 ERROR HANDLING IN FILE OPERATIONS

When we are working with files a number of errors may occur**.** The most common errors that may occur while working with files are listed below

- Attempting to perform read or write operation on a file that does not exist.
- Attempting to process the file even after the last byte file of the file is reached.
- Attempting to write information to a file when opened in read mode.
- Attempting to store information in file, when there is no disk space for storing more data.
- Attempting to create a new file with a file name that already exits.

To overcome from these errors The C++ I/O system maintains status information about the outcome of each I/O operation. The current I/O status of an I/O stream is described in an object of type iostate, which is an enumeration defined by ios that includes the members:

NAME	MEANING
goodbit	No errors occurred
failbit	A non fatal I/O error has occurred
eofbit	End of file has been encountered

There are two ways in which you can obtain the I/O status information. First, we call the rdstate( ) function, which is a member of **ios**. It has this prototype:

```
iostate rdstate( );
```

It returns the current status of the error flags. rdstate( ) returns goodbit when no error has occurred. Otherwise, an error flag is returned. The other way you can determine whether an error has occurred is by using one of these ios member functions:

```
bool bad( );
bool eof( );
bool fail( );
bool good( );
```

The eof( ) function returns true if end of file is reached. The bad( ) function returns true if badbit is set. The fail( ) function returns true if failbit is set. The good( ) function returns true if there are no errors. Otherwise, they return false.

We can use all these functions in our file handling program to minimize the errors.

Example 10:

This program given below contains all error handling mechanisms.

```
#include<iostream.h>
int main()
{
        ifstream in;
        in.open(“text”);
        char ch;
        if(!in)
        {
                cout<<”CANNOT OPEN FILE”<<endl;
                return 0;
        }
        if(in.bad())
        {
              cout<<”FATAL ERROR IN FILE”<<endl;
         }
         cout<<”READING CONTENT FROM THE FILE“<<endl;
        while (in)
        {
                in.get(ch); //get a character to the file
                cout<<ch;
         }
        in.close();
}
```

This program initially checks whether the file pointed by the stream in exits, if so it will check whether there is any fatal error by using the faction bad().If there is no error the entire content of the file is read and displayed.

Have you Understood Questions?

1. Which function we use to detect end of file?
2. When will the fail bit set ?

Summary

- A stream is a common, logical interface to various devices of a computer system.
- A text stream is used with characters. A binary stream can be used with any type of data. No character translation will occur.
- File I/O in C++ has many classes that include ifstream, ofstream and fstream all these classes are derived from ios.
- There are many file opening modes that include input,output,appending etc.,
- A sequential file is one in which every record is accessed serially.
- In C++ every sequential file (including Random access file) will be associated with two file pointers namely get() and put().
- The get pointer is an input pointer; it is used to read the content of the file.
- The put pointer is an output pointer; it is used to write content to the file.
- Random access file is one that allows accessing records randomly in any order
- The tell function is use to give the current position of the file.
- The seek function is used to move the file pointer to the specified position.
- The read and write functions are used perform I/O operations in a binary file.
- The C++ I/O system maintains status information about the outcome of each I/O operation.
- The eof( ) function returns true if end of file is reached.
- The bad( ) function returns true if badbit is set.
- The fail( ) function returns true if failbit is set.
- The good( ) function returns true if there are no errors.

Exercises

Short Questions

1. Draw the hierarchy of C++ IO streams
2. Mention the advantage of using binary files
3. List out the various file opening modes
4. ifstream in(“emp.dat”); opens the file in ______________________ mode
5. ___________ symbol is used to combine file opening modes
6. Mention the use of get() and put() pointers
7. Write the syntax of seekg and seep function.
8. Mention the members of the iostate object
9. To close a file we will call ________________ function.
10. ios :: end is used to position the file pointer at the __________ of the file.

Long Questions

1. Explain random access files also explain how the file pointer is manipulated
2. Write short notes on reading and writing objects in binary files.
3. Explain in detail about error handling functions in files
4. Write short motes on sequential file organization

Programming Exercises

1. Write a program to create a file called “emp.dat” and write employee details into that file
2. Create a file called “number.txt” and write some numbers to the file. Create files called “odd.txt” and “even.txt” and write all the odd numbers to file “odd.txt” and even numbers to the file “even.txt”.
3. Create a class called sales with your own data members and functions. Create object for the class and write the object to the binary file called “sales.dat”.
4. Modify program 1 by including all error handling facilities.

Answers to Have you Understood Questions

Section 7.3

1. A stream is a common, logical interface to various devices of a computer system
2. (1) Binary stream (2) Text stream

Section 7.4

1. ifstream, ofstream and fstream.
2. To create an input/output stream we create an object of type **fstream**.
3. ios::in , ios::out, ios::trunk

Section 7.5

1. A sequential file is one in which every record is accessed serially
2. (1) get (2) put pointers

Section 7.6

1. Random access file is one that allows accessing records randomly in any order
2. seekg and seep
3. Moves the put pointer k bytes ahead from the beginning of the file pointed by stream out

Section 7.7

1. Using read() and write() functions

Section 7.8

1. By using eof() function.
2. When a non fatal I/O error has occurred