11 Working with Files

LEARNING OBJECTIVES

After going through this chapter, you will be able to

- List the classes available in C++ for file stream operations
- Identify the methods used for opening and closing of files
- List the various file mode parameters of open () function
- Illustrate the use of file pointers and their manipulators
- Describe the various functions used for sequential input and output operations
- Explain the concept of random file access
- List the various error handling functions
- Discuss how file names are passed as command line arguments

11.1 INTRODUCTION

Many real-life problems handle large volumes of data and, in such situations, we need to use some secondary storage device such as a hard disk 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 the disk. Programs can be designed to perform the read and write operations on these files.

A program typically involves either or both of the following kinds of data communication:

1. Data transfer between the console unit and the program.
2. Data transfer between the program and a disk file.

This is illustrated in Fig. 11.1.

We have already discussed the technique of handling data communication between the console unit and the program. In this chapter, we will discuss various methods available for storing and retrieving the data from 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 the files. The stream that supplies data to the program is known as *input stream* and the one that receives data from the program is known as *output stream*. In other words, the input stream extracts (or reads) data from the file and the output stream inserts (or writes) data to the file. This is illustrated in Fig. 11.2.

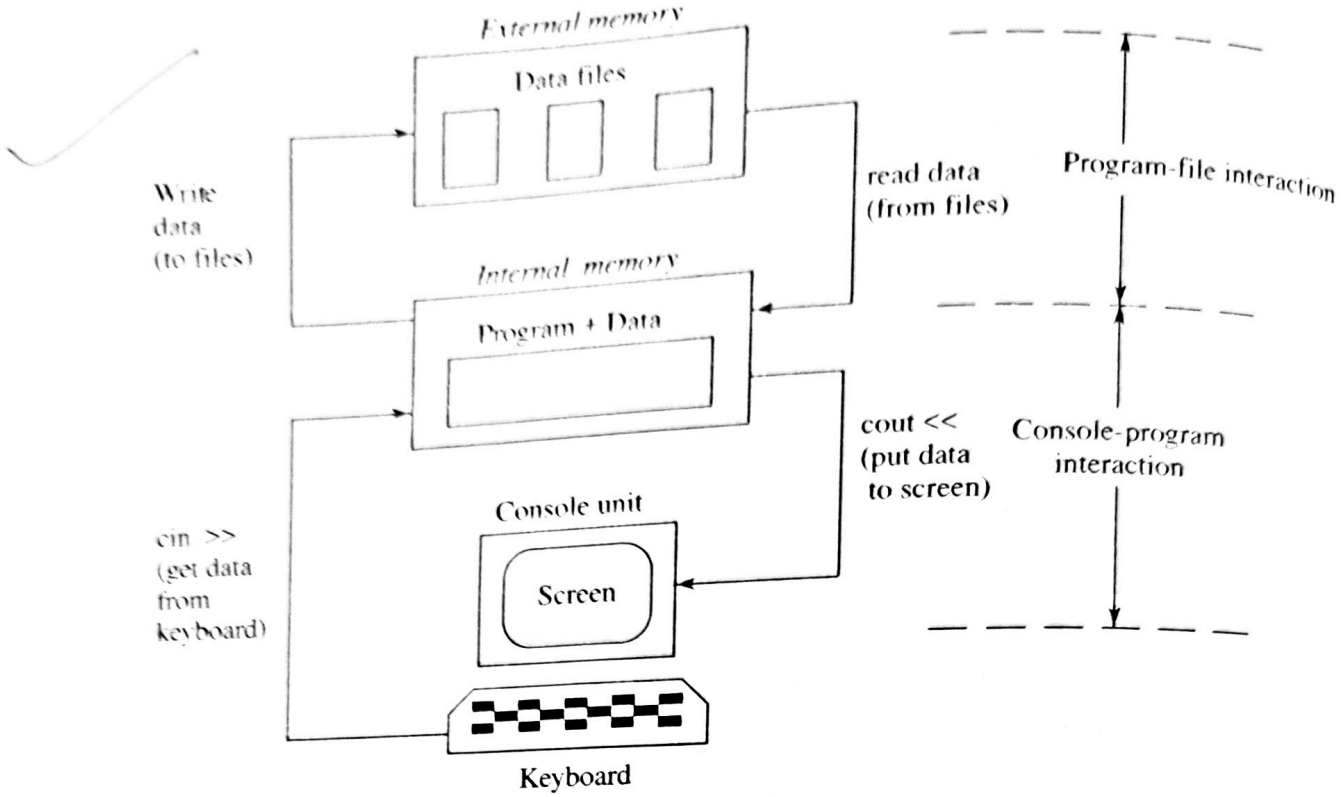

Fig. 11.1 *Consol-program-file interaction*

The input operation involves the creation of an input stream and linking it with the program and the input
Similarly, the output operation involves establishing an output stream with the necessary links with the program and
output file.

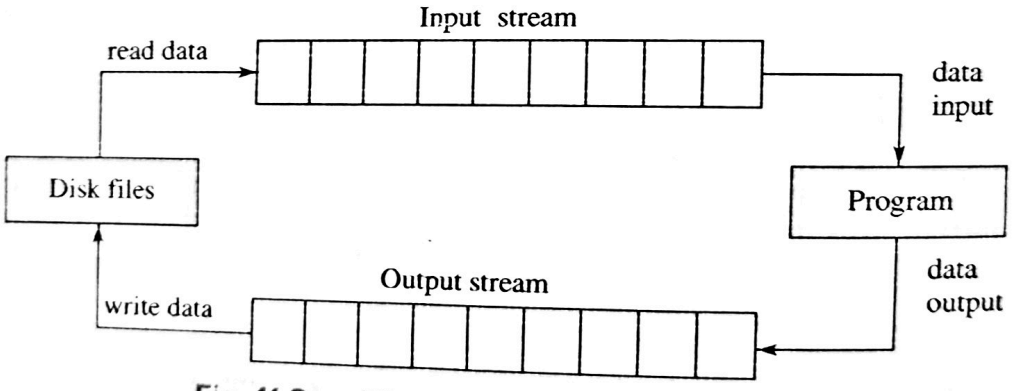

Fig. 11.2 *File input and output streams*

11.2 CLASSES FOR FILE STREAM OPERATIONS

The I/O system of C++ contains a set of classes that define the file handling methods. These include **ifstream, ofstre**
and **fstream**. These classes are derived from **fstreambase** and from the corresponding *iostream* class as shown
Fig. 11.3. These classes, designed to manage the disk files, are declared in *fstream* and therefore, we must include t
file in any program that uses files.

Table 11.1 shows the details of file stream classes. Note that these classes contain many more features. For mo
details, refer to the manual.

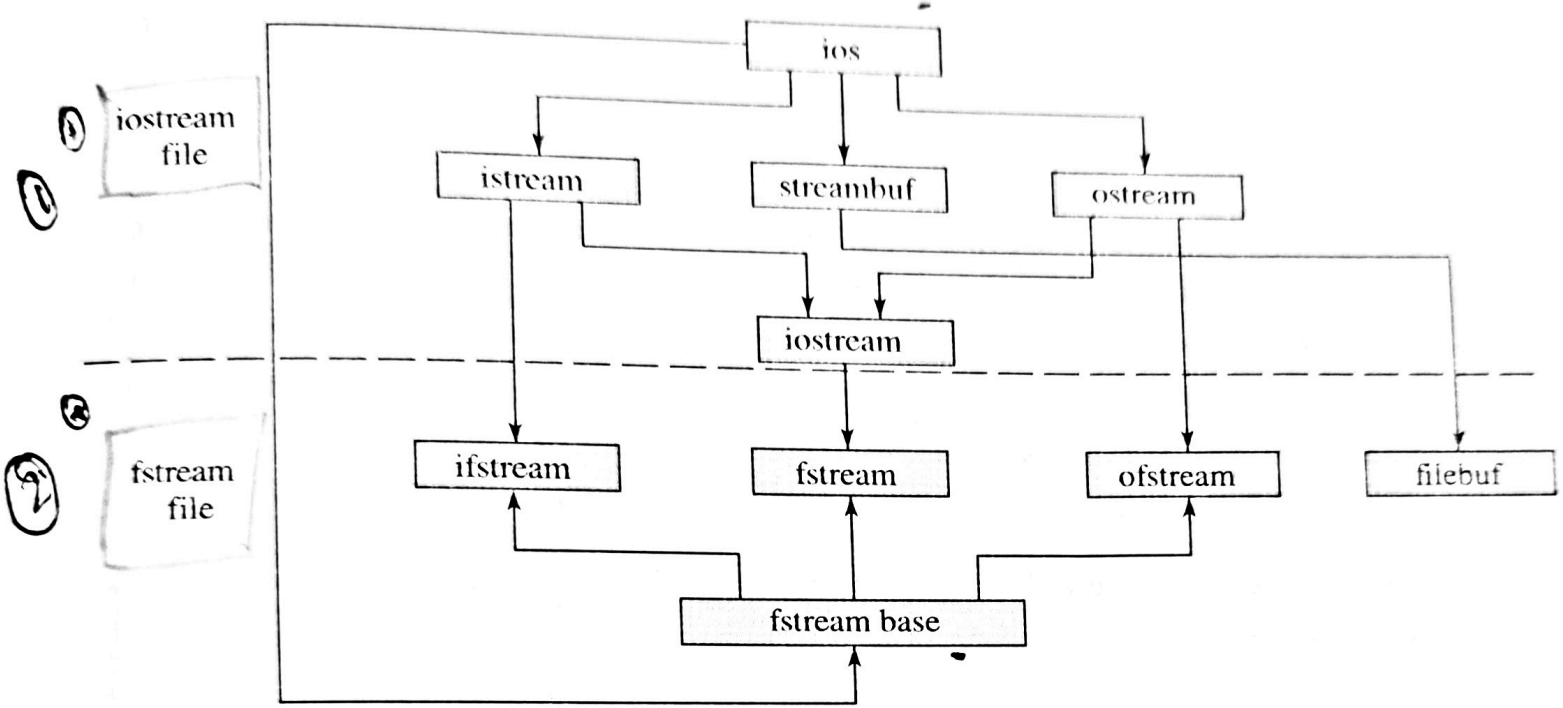

Fig. 11.3 *Stream classes for file operations (contained in fstream file)*

Table 11.1 *Details 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 the 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()** and **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. Inherits all the functions from **istream** and **ostream** classes through **iostream**.

```
INVENT.ORY
student
salary
OUTPUT
```

As stated earlier, for opening a file, we must first create a file stream and then link it to the filename. A file stream can be defined using the classes **ifstream, ofstream,** and **fstream** that are contained in the header file *fstream*. The class to be used depends upon the purpose, that is, whether we want to read data from the file or write data to it. A file can be opened in two ways:

1. Using the constructor function of the class.
2. Using the member function **open()** of the class.

The first method is useful when we use only one file in the stream. The second method is used when we manage multiple files using one stream.

Opening Files Using Constructor

We know that a constructor is used to initialize an object while it is being created. Here, a filename is used to initialize the file stream object. This involves the following steps:

1. Create a file stream object to manage the stream using the appropriate class. That is to say, the class **ofstream** is used to create the output stream and the class **ifstream** to create the input stream.
2. Initialize the file object with the desired filename.

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

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

This creates **outfile** as an **ofstream** object that manages the output stream. This object can be any valid object name such as **o_file, myfile** or **fout**. This statement also opens the file results and attaches it to the output stream outfile. This is illustrated in Fig. 11.4.

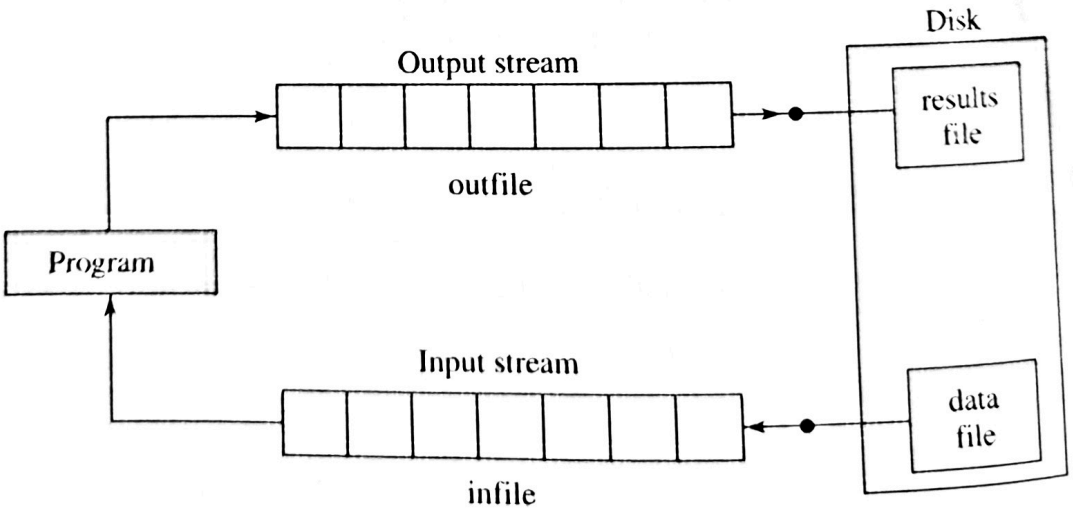

Fig. 11.4 *Two file streams working on separate files*

Similarly, the following statement declares **infile** as an **ifstream** object and attaches it to the file data (input).

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

The program may contain statements like:

```
outfile << "TOTAL";
outfile << sum;
infile >> number;
infile >> string;
```

We can also use the same file for both reading and writing data as shown in Fig. 11.5. The programs would contain the following statements:

```
program1
     .....
     .....
     ofstream outfile("salary");          // creates outfile and
                                             connects
                                          // "salary" to it
     .....
     .....
program2
     .....
     .....
     ifstream infile("salary");       // creates infile and connects
                                      // "salary" to it
     .....
     .....
```

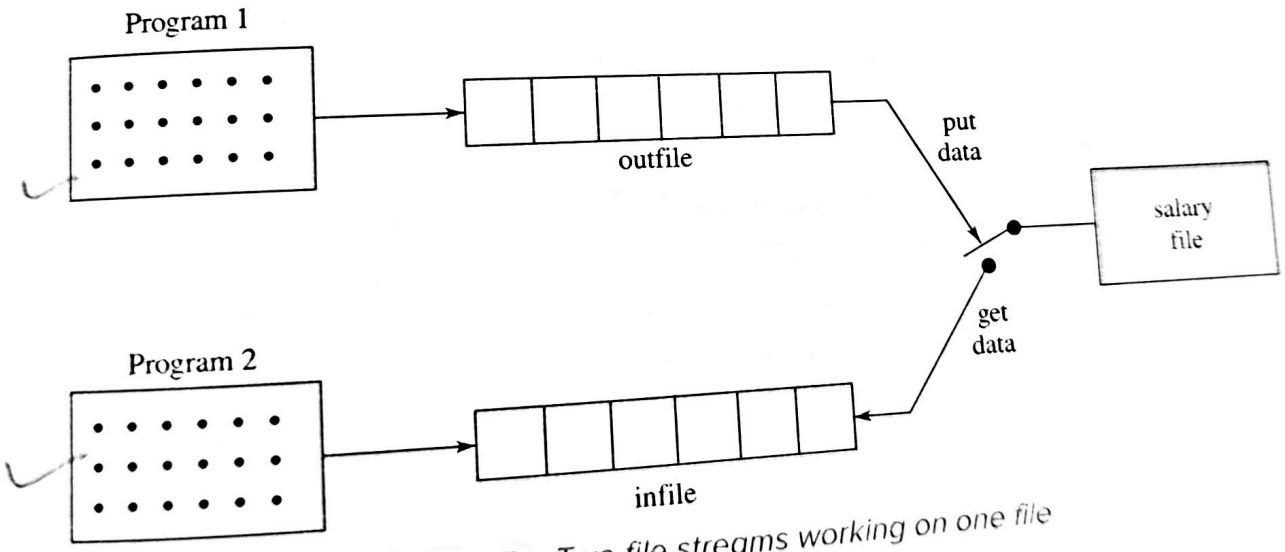

Fig. 11.5 *Two file streams working on one file*

The connection with a file is closed automatically when the stream object expires (when the program terminates). In the above statement, when the *program1* is terminated, the **salary** file is disconnected from the **outfile** stream. Similar action takes place when the *program 2* terminates.

Instead of using two programs, one for writing data (output) and another for reading data (input), we can use a single program to do both the operations on a file. Example.

```
.....
.....
     outfile.close();                 // Disconnect salary from outfile
     ifstream infile("salary");       // and connect to infile
```