

# File Handling in C++

Ankit Dalal<sup>1\*</sup> and Ankur Atri<sup>2</sup>

Department of Information Technology, Computer Science and Information Technology,  
Dronacharya College of Engineering, Gurgaon-122001, India

\*E-mail: [ankit.16898@ggnindia.dronacharya.info](mailto:ankit.16898@ggnindia.dronacharya.info), [ankur.16900@ggnindia.dronacharya.info](mailto:ankur.16900@ggnindia.dronacharya.info)

## Abstract:

*This research paper is a general overview of file handling in C++. In this paper we have studied about stream and stream classes, file and its types, functions of file handling, opening of files using constructors and using open() function. Concepts file modes. At*

*the end we have covered error handling and error handling functions.*

## Keywords:

Stream; File handling; constructors; error; class ; objects

## Introduction

A file represents a sequence of byte on disk where a group of related data is stored. File is created for permanent storage of data. It is a ready made structure. In C, we use a structure pointer of file type to declare a file.

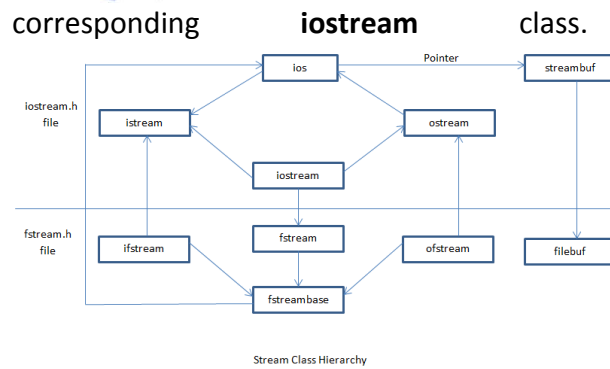
## Stream

A stream is a general name given to a flow of data at the lowest level (at the lowest level, data is just the binary data without any notation of data type). Different streams are used to represent different kind of data flow such as whether data is flowing into the memory or out of the memory. There may be flow of data between variety of devices such as terminals, disks, and tape drives. The I/O system supplies an interface to the programmer that is independent of the actual device being accessed. This interface is called stream.

A stream is a sequence of bytes. The stream that supplies data to the program is known as input stream. It reads the data from the file and hands it over to the program. The stream that receives data from the program is known the output stream. It writes data to the file[1].

## Stream Classes

The file I/O system contains a set of classes that define the file handing methods. These classes, designed to manage both the console and disk files. These classes are called stream classes. These classes are derived from **fstreambase** and from



**Figure 1**

The functions of these classes have been summarized in table below:

Class	Functions
<b>ios</b>	The class ios contains most of the actual input/output code. It keeps the track of error state of the stream and converts data for display.
<b>ostream</b>	It inherits the properties of ios and contains declaration of output functions <b>put()</b> and <b>write()</b> . It contains overloaded insertion operator <b>&gt;&gt;</b> .
<b>istream</b>	It inherits the properties of ios and contains declaration of input functions such as <b>get()</b> , <b>getline()</b> and <b>read()</b> . It contains overloaded extraction operator <b>&lt;&lt;</b> .
<b>iostream</b>	It inherits properties of <b>ios</b> , <b>istream</b> and <b>ostream</b> and hence contains all the input and output functions.
<b>filebuf*</b>	It is a set of buffers to read and write. It contains <b>close()</b> and <b>open()</b> member functions in it.
<b>fstreambase*</b>	This is a base class for <b>fstream</b> , <b>ifstream</b> and

**ofstream** classes. So, it provides operations common to these file streams. It also contains **open()** and **close()** functions.

#### streambuf

It acts as base for **filebuf** class and provides interface to physical devices through buffers.

#### ifstream\*

It provides input operations for file. It inherits the functions **get()**, **getline()**, **read()** and functions for supporting random access (**seekg()** and **tellg()**) from class defined inside **iostream.h** file.

#### ofstream\*

It provides output operations. It inherits **put()** and **write()** functions along with supporting random access (**seekp()** and **tellp()**) from class defined inside **iostream.h** file.

#### fstream\*

It is an input-output file stream class. It provides support for simultaneous input and output operations. It inherits all the functions from **istream** and **ostream** classes through **iostream** class defined inside **iostream.h** file.

\*File stream classes

## Data Files

The data files are the files that stores data pertaining to a specific application, for later use. The data files can be stored in two ways:

- i. Text files.
- ii. Binary files.

A **text file** stores information in ASCII characters. In text file, each line of text is terminated (delimited) with a special character known as EOL (End of Line) character. In text files some internal translations occurs when EOL character is read or written.

A **binary file** is just a file that contains information in the same format in which the information is held in memory. In binary file, there is no delimiter for a line. Here no translations occur in binary files. So binary files are faster and easier for program to read and write than the text files[2].

## Functions of File Handling

Function Name	Operation
<pre>ifstream object;  stream_object.open("File_name",Mode);  OR  ifstream object("File_name",Mode);</pre>	
<b>open()</b>	If file already exists then open the file else create the file and open it.
<b>close()</b>	Close a file which has been opened.

<b>get()</b>	Read a character from a file.
<b>put()</b>	Write a character to a file.
<b>read()</b>	Read a set of data values from file.
<b>write()</b>	Write a set of data values to a file.
<b>seekg()</b>	Set the pointer (input) to a desired point in a file.
<b>seekp()</b>	Set the pointer (output) to a desired point in a file.
<b>tellg()</b>	Gives the current position of get pointer
<b>tellp()</b>	Gives the current position of put pointer.

## Opening and Closing Files

For using a disk file, we need to decide following about the file and its intended use:

- i. Suitable name for file
- ii. Data Structure
- iii. Purpose
- iv. Opening method

**File name** is a string of characters that make a valid file name for operating system. E.g. ABC.CPP, XYZ.txt etc.

**Data Structure** of a file is defined as FILE in the library of standard input-output function definition.

**Purpose** includes in which mode the file is to be opened i.e. read, write or append.

### Syntax for Defining a File

If a file is to be opened then a stream must be obtained first. There are three types of streams: *input*, *output*, and *input/output*.

To create an input stream, you must declare the stream to be of class **ifstream**. To create an output stream, you must declare it as class **ofstream**. Stream that will be performing both input and output operations must be declared as class **fstream**. Once a stream has been created, next step is to associate a file with it. And thereafter the file is opened for processing.

Opening of file can be achieved in two ways:

- i. Using the constructor function of the stream class.
- ii. Using the function **open()**.

The first method is preferred when a single file is used with a stream; however, for managing multiple files with the same stream, the second method is preferred.

### Opening File Using Constructor

A constructor of a class initializes an object of its class when it is being created. In the same way, the constructor of stream classes (**ifstream**, **ofstream**, or **fstream**) are used to initialize file stream objects with the *filenames* passed to them.

To open a file, **Datafile**, as an input file, we shall create a file stream object of input type i.e., **ifstream** type:

```
ifstream input_file("Datafile",ios::in);
```

This statement will create an object (**input\_file**) of input file stream. The object

name is a user defined name i.e. any valid identifier name. After creating the **ifstream** object **input\_file**, the file **Datafile** is opened and attached to the input stream **input\_file**.

Now to read from this file, this stream object will be used using the *getfrom operator* ("**>>**")

E.g.

```
charch;
```

```
input_file>>ch;
```

Similarly, when you want a program to write a file i.e., to open an *output* file (on which no operation can take place except writing). This can be accomplished by creating **ofstream** object to manage the output stream and associating that object with a particular file.

```
ofstream outpt_file("secret",ios::out);
```

This would create an output stream object named as **outpt\_file** and attach the file **secret** with it.

Now to write something to it, the stream object will be used using *put to operator* ("**<<**")

### Opening File Using Open() Function

**There** may be situations requiring a program to open more than one file. The strategy for opening multiple files depends upon they will be used. If the situation requires simultaneous processing of two files, then you need to create a separate stream for each file. If there is sequential processing of files then you can open a

single stream and associate it with each file in turn. To use this approach, declare a stream object without initializing it, then use a second statement to associate the stream with file.

E.g.

```
ifstreamabc;
//create an input stream

abc.open("Menu.txt",ios::in);
//associate abc stream with file Menu.txt

abc.close();
//terminate association with Menu.txt

abc.open("Misty.txt",ios::in);
//associate abc with file Misty.txt

abc.close();
//terminate association
```

This code will let you handle reading two files in succession. But the first file is closed before opening the second file. This is necessary because a stream can be connected to only one file at a time.

### Closing a file

A file is closed by disconnecting it with the stream it is associated with. The close() function accomplishes this[3].

Syntax: stream\_object.close();

## Concept of File Modes

The *filemode* describes how a file is to be used that is to read from it, to write to it, to append it and so on. When you associate a stream with a file, either by initializing a file stream object with a file name or using

**open()** method, you can provide a second argument specifying the file mode.

syntax:

stream\_object.open("filename",(filemode));

The second argument of **open()**, *filemode*. Is of type **int**, and you can choose several constants defined in the **iosclass**. [4]

File mode Constants:

Constant	Meaning
ios::in	It opens file for reading, i.e., input mode.
ios::out	It opens file for writing. This also opens the file in <b>ios::truncmode</b> , by default. This means an existing file is truncated when opened i.e., previous contents are discarded.
ios::ate	This seeks to end-of-file upon opening of the file. I/O operation can still occur anywhere within the file
ios::app	This causes all output to that file to be appended to the end. This value can only be used with files capable of output.
ios::trunc	This value causes the contents of a pre-existing file by the same name to be destroyed and truncates the file to zero length.
ios::noncreate	This causes the <b>open()</b> function to fail if the file already exists. It will not create a new file with that name.
ios::noreplace	This cause the <b>open()</b> function to fail if the file already exists. This is used when you want to create a new file and at the same

	time.
ios::binary	This cause the file to be opened in binary mode. By default, files are opened in text mode. When a file is opened in text mode, various characters translations may take place, such as the conversion of carriage-return into newlines. However, no such character translations occur in files opened i binary mode.

## Detecting End of File

Detection of end-of-file condition is necessary for any further attempt to read data from the file. End-of-file condition can be detected by using **eof()** function whose return type is **int**. It returns a **non-zero (true value)** if end-of-file is encountered while reading else returns **zero(false value)**[5].

syntax:

```
stream_object.eof()==0;
```

//condition for occurrence of end-of-file condition

## Error Handling

It is possible that an error may occur during input-output operation on a file. Typical error situations include

1. Trying to read beyond EOF (End of File).
2. Device overflow.
3. Trying to use a file that has not been opened.

4. Trying to perform an operation on file, when a file is opened for another type of operation.
5. Opening a file with an invalid name.
6. Attempting to write to a write protected file.

C++ language provides some error handling functions to handle these kinds of errors.

1. **eof()** : Returns *true* (non-zero value) if end-of-file is encountered while reading, else, returns *false*(zero).
2. **fail()** : Returns true when input or output operation is failed.
3. **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 continue the operation.'

**good()** : Returns true when no error has occurred. This means, all the above functions are false. For example, if **file.good()** is true, all is well with the stream **file** and we can proceed to perform I/O operations[6]. When it returns false, no further operations can be carried out.

## Conclusion

A text file stores information in ASCII characters. Files help storing information permanently. File handling helps easy

implementation of various operations on file such as creating and updating. It also help in maintaining records. It deals with data in binary form at lower level(Computer level) and have associated data type such as integer and character etc.Manages interface with data at lower level and user level.

## REFERENCES

1. File handling, Study Tonight, <http://www.studytonight.com/c/file-input-output.php>
2. Sumita Arora, Computer Science with C++ ( Volume 1<sup>st</sup> 9<sup>th</sup> Edition)
3. Stanley B. Lippman, JoseeLajoie (1999). *C++ Primer* (third ed.). Massachusetts: Addison-Wesley. pp. 1109–1112. ISBN 0-201-82470-1.
4. Opening and Closing Files, Wikipedia, [http://en.wikibooks.org/wiki/C\\_Programming/File\\_IO#Opening\\_and\\_Closing\\_Files](http://en.wikibooks.org/wiki/C_Programming/File_IO#Opening_and_Closing_Files)
5. Holzner, Steven (2001). *C++ : Black Book*. Scottsdale, Ariz.: Coriolis Group. p. 584. ISBN 1-57610-777-9.
6. BjarneStroustrup (1997 3rd Printing). *The C++ programming language*. Addison-Wesley. pp. 637–640. ISBN 0-201-88954-4.