**Unit - V   
File Handling**

**Prof. Jayshri A. Kandekar**

**Department of Computer science and Design**

**K.K.Wagh Institute of Engineering Education & Research**

**Nashik**

**Syllabus**

Datahierarchy,Streamandfiles,StreamClasses,DiskFileI/Owith

Streams,FilePointers,FileI/OwithMemberFunctions.

SelfStudy:FormattedI/O,commandlinearguments

Files are used to store data in a storage device permanently. File handling provides a mechanism to store the output of a program in a file and to perform various operations on it. Using file handling we can store our data in secondary memory (Hard disk).  
How to achieve the File Handling  
For achieving file handling we need to follow the following steps:-  
 STEP 1-Creating a file  
 STEP 2-Opening a file  
 STEP 3-Writing data into the file  
 STEP 4-Reading data from the file  
 STEP 5-Closing a file.

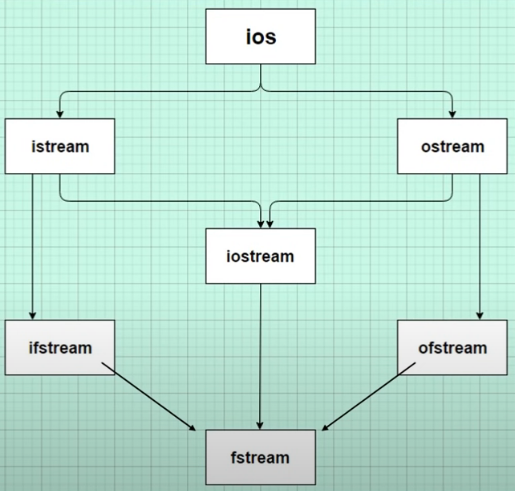
**Datahierarchy:**

**Streamandfiles:**

We give input to the executing program and the execution program gives back the output. The sequence of bytes given as input to the executing program and the sequence of bytes that comes as output from the executing program are called stream. In other words, streams are nothing but the flow of data in a sequence.

A stream is an abstraction that represents a device on which operations of input and output are performed. A stream can be represented as a source or destination of characters of indefinite length depending on its usage.

The input and output operation between the executing program and the devices like keyboard and monitor are known as “console I/O operation”. The input and output operation between the executing program and files are known as “disk I/O operation”.



**StreamClasses:**

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

## **fstream library**

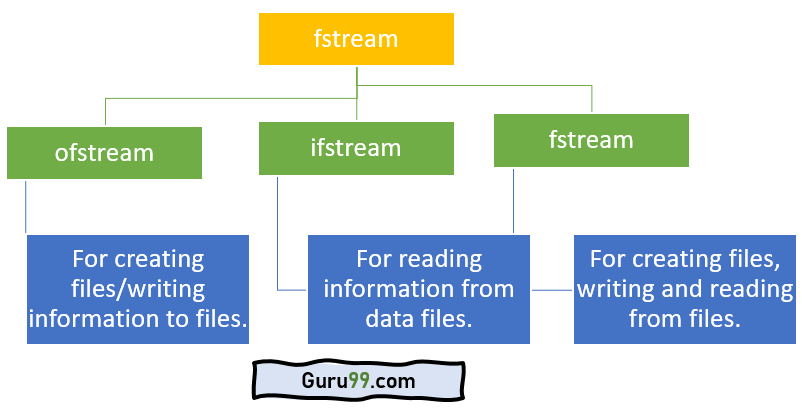
## The header file we will be using to gain access to the file handling method.

## In C++, fstream library is used to handle files, and it is dealt with the help of three classes known as ofstream, ifstream and fstreamavailable in fstreamheaderfile..

**ofstream:** Stream class to write on files

This class helps create and write the data to the file obtained from the program’s output. It is also known as the input stream.  
**ifstream:** Stream class to read from files

We use this class to read data from files and also known as the input stream.  
**fstream:** Stream class to both read and write from/to files.



This class is the combination of both ofstream and ifstream. It provides the capability of creating, writing and reading a file.

To access the following classes, you must include the fstream as a header file like how we declare iostream in the header

|  |  |
| --- | --- |
|  | #include<iostream>  #include<fstream> |

## **File Operations in C++**

C++ provides us with four different operations for file handling. They are:

1. **open()** – This is used to create a file.
2. **read()  –** This is used to read the data from the file.
3. **write() –** This is used to write new data to file.
4. **close() –** This is used to close the file.

C++ provides us with the following operations in File Handling:

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

**Opening a File**

Generally, the first operation performed on an object of one of these classes is to associate it to a real file. This procedure is known to open a file.

To read or enter data to a file, we need to open it first. This can be performed with the help of ‘ifstream’ for reading and ‘fstream’ or ‘ofstream’ for writing or appending to the file. All these three objects have open() function pre-built in them.

Now the first step to open the particular file for read or write operation. We can open a file using any one of the following methods:

1. First is bypassing the file name in constructor at the time of object creation.

2. Second is using the open() function.

**For e.g.**

***Open File by using constructor****ifstream (const char\* filename, ios\_base::openmode mode = ios\_base::in);   
ifstreamfin(filename, openmode) by default openmode = ios::in   
ifstream fin(“filename”);*

***Open File by using open method****Calling of default constructor   
ifstream fin;  
fin.open(filename, openmode)   
fin.open(“filename”);*

To open a file use

|  |  |
| --- | --- |
|  |  |

**Syntax**

Here, the first argument of the open function defines the name and format of the file with the address of the file.

**open() function**

|  |  |
| --- | --- |
|  | open( FileName , Mode ); |

Here:

FileName – It denotes the name of file which has to be opened.

Mode – There different mode to open a file and it explained in this article.

The second argument represents the mode in which the file has to be opened.

The following modes are used as per the requirements.

**void open(const char\* file\_name, ios:: openmode mode);**

**Modes :**

| **Member Constant** | **Stands For** | **Access** |
| --- | --- | --- |
| in \* | input | Opens the file to read(default for ifstream)  File open for reading: the internal stream buffer supports input operations. |
| out | output | Opens the file to write(default for ofstream)  File open for writing: the internal stream buffer supports output operations. |
| binary | binary | Opens the file in binary mode  Operations are performed in binary mode rather than text. |
| ate | at end | Opens the file and moves the control to the end of the file  The output position starts at the end of the file. |
| app | append | Opens the file and appends all the outputs at the end  All output operations happen at the end of the file, appending to its existing contents. |
| trunc | truncate | Removes the data in the existing file  Any contents that existed in the file before it is open are discarded. |
| Nocreate |  | Opens the file only if it already exists |
| noreplace |  | Opens the file only if it does not already exist |

**Default Open Modes :**

|  |  |
| --- | --- |
| ifstream | ios::in |
| ofstream | ios::out |
| fstream | ios::in | ios::out |

## **Writing to File**

Till now, we learned how to create the file using C++. Now, we will learn how to write data to file which we created before. We will use fstream or ofstream object to write data into the file and to do so; we will use stream insertion operator (<<) along with the text enclosed within the double-quotes.

With the help of open() function, we will create a new file named ‘FileName’ and then we will set the mode to ‘ios::out’ as we have to write the data to file.

#### **Syntax:**

|  |  |
| --- | --- |
|  | FileName<<"Insert the text here"; |

## **Reading from file** in C++

We can perform the reading of data from a file with the CIN to get data from the user, but then we use CIN to take inputs from the user’s standard console. Here we will use fstream or ifstream.

#### **Syntax:**

|  |  |
| --- | --- |
|  | FileName>>Variable; |

## **Read a File**

To read from a file, use either the ifstream or fstream class, and the name of the file.

Note that we also use a while loop together with the getline() function (which belongs to the ifstream class) to read the file line by line, and to print the content of the file:

### Example

// Create a text string, which is used to output the text file  
string myText;  
  
// Read from the text file  
ifstreamMyReadFile("filename.txt");  
  
// Use a while loop together with the getline() function to read the file line by line  
while (getline (MyReadFile, myText)) {  
  // Output the text from the file  
  cout<<myText;  
}  
  
// Close the file  
MyReadFile.close();

## 

## **Closing a file** in C++

Closing a file is a good practice, and it is must to close the file. Whenever the C++ program comes to an end, it clears the allocated memory, and it closes the file. We can perform the task with the help of close() function.

#### **Syntax:**

|  |  |
| --- | --- |
|  | FileName.close(); |

## Create and Write To a File

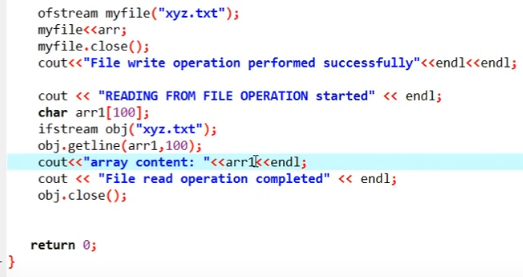
To create a file, use either the ofstream or fstream class, and specify the name of the file.

To write to the file, use the insertion operator (<<).

Create and Write to File:

### Open Write file.png

Read From File:



Another example:



### Example

#include <iostream>  
#include <fstream>  
using namespace std;  
  
int main() {  
  // Create and open a text file  
  ofstreamMyFile("filename.txt");  
  
  // Write to the file  
  MyFile<<"Files can be tricky, but it is fun enough!";  
  
  // Close the file  
  MyFile.close();  
}

## Read and Write Example

Following is the C++ program which opens a file in reading and writing mode. After writing information entered by the user to a file named afile.dat, the program reads information from the file and outputs it onto the screen −

#include <fstream>

#include <iostream>

using namespace std;

int main () {

char data[100];

// open a file in write mode.

Ofstreamoutfile;

outfile.open("afile.dat");

cout<<"Writing to the file"<<endl;

cout<<"Enter your name: ";

cin.getline(data, 100);

// write inputted data into the file.

outfile<< data <<endl;

cout<<"Enter your age: ";

cin>> data;

cin.ignore();

// again write inputted data into the file.

outfile<< data <<endl;

// close the opened file.

outfile.close();

// open a file in read mode.

ifstreaminfile;

infile.open("afile.dat");

cout<<"Reading from the file"<<endl;

infile>> data;

// write the data at the screen.

cout<< data <<endl;

// again read the data from the file and display it.

infile>> data;

cout<< data <<endl;

// close the opened file.

infile.close();

return 0;

}

When the above code is compiled and executed, it produces the following sample input and output −

$./a.out

Writing to the file

Enter your name: Zara

Enter your age: 9

Reading from the file

Zara

9

Above examples make use of additional functions from cin object, like getline() function to read the line from outside and ignore() function to ignore the extra characters left by previous read statement.

## C++ FileStream example: writing to a file

Let's see the simple example of writing to a text file **testout.txt** using C++ FileStream programming.

1. #include <iostream>
2. #include <fstream>
3. using namespace std;
4. int main () {
5. ofstream filestream("testout.txt");
6. if (filestream.is\_open())
7. {
8. filestream << "Welcome to javaTpoint.\n";
9. filestream << "C++ Tutorial.\n";
10. filestream.close();
11. }
12. else cout <<"File opening is fail.";
13. return 0;
14. }

**Output:**

The content of a text file testout.txt is set with the data:

Welcome to javaTpoint.

C++ Tutorial.

## C++ FileStream example: reading from a file

Let's see the simple example of reading from a text file **testout.txt** using C++ FileStream programming.

1. #include <iostream>
2. #include <fstream>
3. using namespace std;
4. int main () {
5. string srg;
6. ifstream filestream("testout.txt");
7. if (filestream.is\_open())
8. {
9. while ( getline (filestream,srg) )
10. {
11. cout << srg <<endl;
12. }
13. filestream.close();
14. }
15. else {
16. cout << "File opening is fail."<<endl;
17. }
18. return 0;
19. }

#### Note: Before running the code a text file named as **"testout.txt"** is need to be created and the content of a text file is given below: Welcome to javaTpoint. C++ Tutorial.

**Output:**

Welcome to javaTpoint.

C++ Tutorial.

## C++ Read and Write Example

Let's see the simple example of writing the data to a text file **testout.txt** and then reading the data from the file using C++ FileStream programming.

1. #include <fstream>
2. #include <iostream>
3. using namespace std;
4. int main () {
5. char input[75];
6. ofstream os;
7. os.open("testout.txt");
8. cout <<"Writing to a text file:" << endl;
9. cout << "Please Enter your name: ";
10. cin.getline(input, 100);
11. os << input << endl;
12. cout << "Please Enter your age: ";
13. cin >> input;
14. cin.ignore();
15. os << input << endl;
16. os.close();
17. ifstream is;
18. string line;
19. is.open("testout.txt");
20. cout << "Reading from a text file:" << endl;
21. while (getline (is,line))
22. {
23. cout << line << endl;
24. }
25. is.close();
26. return 0;
27. }

**Output:**

Writing to a text file:

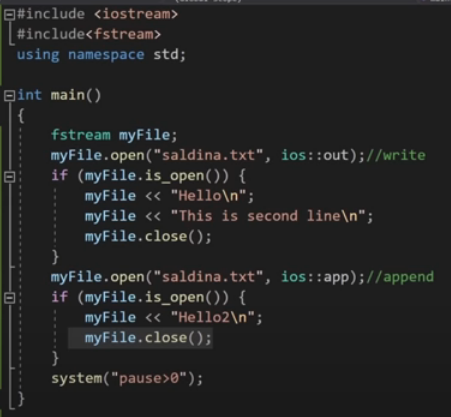
Please Enter your name: Nakul Jain

Please Enter your age: 22

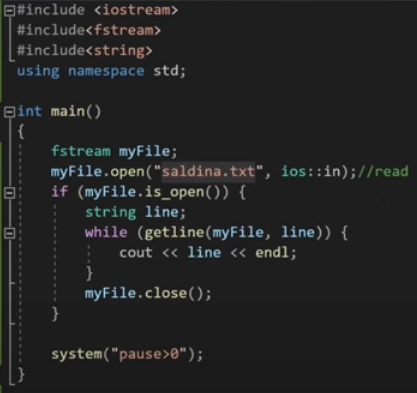
Reading from a text file: Nakul Jain

22

Append Operation in File:



Read from File:



Below is the implementation by using **ifstream&ofstream classes**.

**C++**

|  |
| --- |
| /\* File Handling with C++ using ifstream&ofstream class object\*/  /\* To write the Content in File\*/  /\* Then to read the content of file\*/  #include <iostream>    /\* fstream header file for ifstream, ofstream,    fstream classes \*/  #include <fstream>    usingnamespacestd;    // Driver Code  intmain()  {      // Creation of ofstream class object      ofstreamfout;        string line;        // by default ios::out mode, automatically deletes      // the content of file. To append the content, open in ios:app      // fout.open("sample.txt", ios::app)      fout.open("sample.txt");        // Execute a loop If file successfully opened      while(fout) {            // Read a Line from standard input          getline(cin, line);            // Press -1 to exit          if(line == "-1")              break;            // Write line in file          fout<< line <<endl;      }        // Close the File      fout.close();        // Creation of ifstream class object to read the file      ifstream fin;        // by default open mode = ios::in mode      fin.open("sample.txt");        // Execute a loop until EOF (End of File)      while(getline(fin, line)) {            // Print line (read from file) in Console          cout<< line <<endl;      }        // Close the file      fin.close();        return0;  } |

Time Complexity: O(n)  
Auxiliary Space: O(1)

Below is the implementation by using **fstream class**.

**C++**

|  |
| --- |
| /\* File Handling with C++ using fstream class object \*/  /\* To write the Content in File \*/  /\* Then to read the content of file\*/  #include <iostream>    /\* fstream header file for ifstream, ofstream,     fstream classes \*/  #include <fstream>    usingnamespacestd;    // Driver Code  intmain()  {      // Creation of fstream class object      fstreamfio;        string line;        // by default openmode = ios::in|ios::out mode      // Automatically overwrites the content of file, To append      // the content, open in ios:app      // fio.open("sample.txt", ios::in|ios::out|ios::app)      // ios::trunc mode delete all content before open      fio.open("sample.txt", ios::trunc | ios::out | ios::in);        // Execute a loop If file successfully Opened      while(fio) {            // Read a Line from standard input          getline(cin, line);            // Press -1 to exit          if(line == "-1")              break;            // Write line in file          fio<< line <<endl;      }        // Execute a loop until EOF (End of File)      // point read pointer at beginning of file      fio.seekg(0, ios::beg);        while(fio) {            // Read a Line from File          getline(fio, line);            // Print line in Console          cout<< line <<endl;      }        // Close the file      fio.close();        return0;  } |

Time Complexity: O(n)  
Auxiliary Space: O(1)

**C++**

|  |
| --- |
| Q: write a single file handling program in c++ to reading and writing data on a file.    #include<iostream>  #include<fstream>    usingnamespacestd;  main()  {        intrno,fee;        charname[50];          cout<<"Enter the Roll Number:";        cin>>rno;          cout<<"\nEnter the Name:";        cin>>name;          cout<<"\nEnter the Fee:";        cin>>fee;          ofstreamfout("d:/student.doc");          fout<<rno<<"\t"<<name<<"\t"<<fee;   //write data to the file student          fout.close();          ifstream fin("d:/student.doc");          fin>>rno>>name>>fee;   //read data from the file student          fin.close();          cout<<endl<<rno<<"\t"<<name<<"\t"<<fee;        return0;  } |

Time Complexity: O(1)  
Auxiliary Space: O(1)

**C++**

|  |
| --- |
| // Q: WA C++ file handling program to read data from the file called student.doc    #include<iostream>  #include<fstream>    usingnamespacestd;    main()Create and Write To a File  To create a file, use either the ofstream or fstream class, and specify the name of the file.  To write to the file, use the insertion operator (<<). Example #include <iostream> #include <fstream> using namespace std;  int main() {   // Create and open a text file   ofstreamMyFile("filename.txt");    // Write to the file   MyFile<<"Files can be tricky, but it is fun enough!";    // Close the file   MyFile.close(); }  {        intrno,fee;        charname[50];          ifstream fin("d:/student.doc");          fin>>rno>>name>>fee;   //read data from the file student          fin.close();          cout<<endl<<rno<<"\t"<<name<<"\t"<<fee;          return0;  } |

Time Complexity: O(1)  
Auxiliary Space: O(50)

**DiskFileI/Owith Streams:**

**FilePointers:**

## File Position Pointers

A file position pointer points to a particular index in a file where read or write operations occur. There are two types of pointers get and put. We can find the position of these pointers by associated functions tellg() and tellp().We can reposition the file-position pointer in istream and ostream using its special member functions. These member functions are ‘seekg’ and ‘seekp’we can seek(change) the position of the pointer with these functions. ‘seekg’ or ‘seek get’ is used for istream and ‘seekp’ or ‘seek put’ is used for ostream. There, we can either read or write after seeking a particular position.

Both these member functions seekg() and seekp() takes parameters as long integers and seek directions.

The seek directions can be:

**ios::beg** (for positioning at the beginning of a stream)  
**ios::cur** (for positioning relative to the current position of a stream)  
**ios::end** (to position relative to the end of a stream)

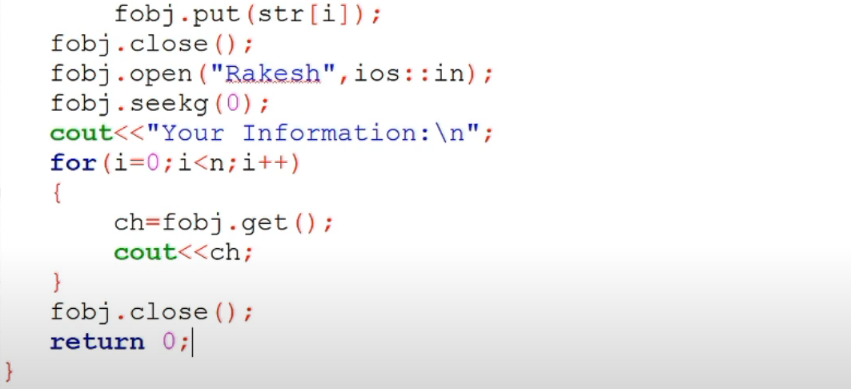
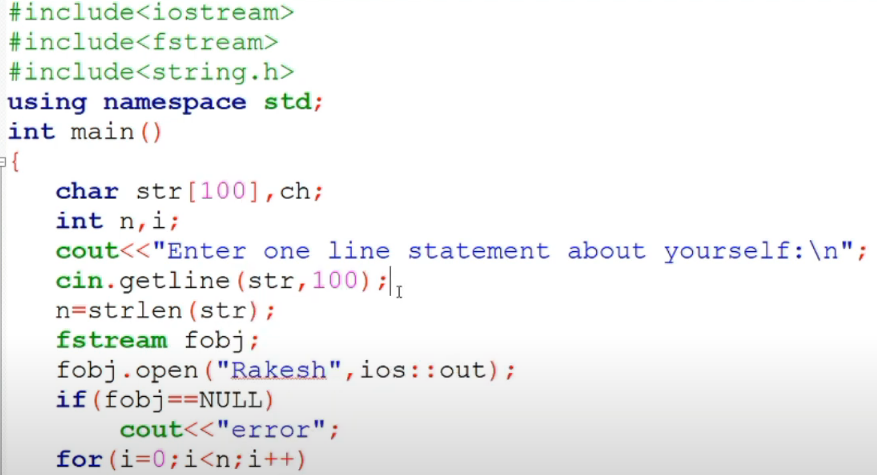
### 

### 

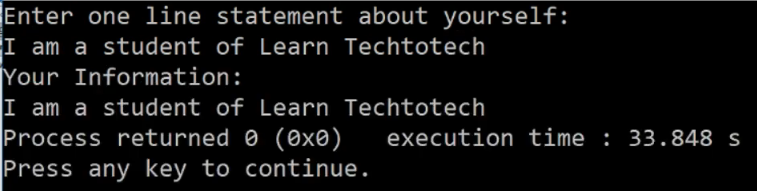
**Output:**

### 

# Use of get(), put() and getline() in File Handlings



**OUTPUT:**



### get and put stream positioning

All i/o streams objects keep internally -at least- one internal position:  
  
ifstream, like istream, keeps an internal *get position* with the location of the element to be read in the next input operation.  
  
ofstream, like ostream, keeps an internal *put position* with the location where the next element has to be written.  
  
Finally, fstream, keeps both, the *get* and the *put position*, like iostream.  
  
These internal stream positions point to the locations within the stream where the next reading or writing operation is performed. These positions can be observed and modified using the following member functions:

#### tellg() and tellp()

These two member functions with no parameters return a value of the member type streampos, which is a type representing the current *get position* (in the case of tellg) or the *put position* (in the case of tellp).

#### seekg() and seekp()

These functions allow to change the location of the *get* and *put positions*. Both functions are overloaded with two different prototypes. The first form is:  
  
seekg ( position );  
seekp ( position );  
  
Using this prototype, the stream pointer is changed to the absolute position position (counting from the beginning of the file). The type for this parameter is streampos, which is the same type as returned by functions tellg and tellp.  
  
The other form for these functions is:  
  
seekg ( offset, direction );  
seekp ( offset, direction );  
  
Using this prototype, the *get* or *put position* is set to an offset value relative to some specific point determined by the parameter direction. offset is of type streamoff. And direction is of type seekdir, which is an *enumerated type* that determines the point from where offset is counted from, and that can take any of the following values:

|  |  |
| --- | --- |
| ios::beg | offset counted from the beginning of the stream |
| ios::cur | offset counted from the current position |
| ios::end | offset counted from the end of the stream |

The following example uses the member functions we have just seen to obtain the size of a file:

|  |  |  |
| --- | --- | --- |
|  | // obtaining file size  #include <iostream>  #include <fstream>  usingnamespace std;  int main () {  streamposbegin,end;  ifstreammyfile ("example.bin", ios::binary);  begin = myfile.tellg();  myfile.seekg (0, ios::end);  end = myfile.tellg();  myfile.close();  cout<<"size is: "<< (end-begin) <<" bytes.\n";  return 0;  } | size is: 40 bytes. |

Notice the type we have used for variables begin and end:

|  |  |
| --- | --- |
| 1 | streampos size; |

streampos is a specific type used for buffer and file positioning and is the type returned by file.tellg(). Values of this type can safely be subtracted from other values of the same type, and can also be converted to an integer type large enough to contain the size of the file.  
  
These stream positioning functions use two particular types: streampos and streamoff. These types are also defined as member types of the stream class:

|  |  |  |
| --- | --- | --- |
| **Type** | **Member type** | **Description** |
| [streampos](https://cplusplus.com/streampos) | [ios::pos\_type](https://cplusplus.com/ios" \l "types) | Defined as [fpos<mbstate\_t>](https://cplusplus.com/fpos). It can be converted to/from [streamoff](https://cplusplus.com/streamoff) and can be added or subtracted values of these types. |
| [streamoff](https://cplusplus.com/streamoff) | [ios::off\_type](https://cplusplus.com/ios" \l "types) | It is an alias of one of the fundamental integral types (such as int or long long). |

Each of the member types above is an alias of its non-member equivalent (they are the exact same type). It does not matter which one is used. The member types are more generic, because they are the same on all stream objects (even on streams using exotic types of characters), but the non-member types are widely used in existing code for historical reasons.

### tellp() &tellg()

tellp() returns the current position of **put pointer**, which is used with output streams while writing the data to the file.

tellg() returns the current position of **get pointer**, which is used with input streams while receiving the data from the file.

**Example:**

#include <iostream>

#include <fstream>

Usingnamespacestd;

int main() {

ofstream file;

*// Open file in write mode.*

file.open("myfile.txt", ios::out);

cout<<"Position of put pointer before writing:"<<file.tellp() <<endl;

file<<"Hello Everyone"; *// Write on file.*

cout<<"Position of put pointer after writing:"<<file.tellp() <<endl;

file.close();

ifstream file1;

file1.open("myfile.txt", ios::in); *// Open file in read mode.*

cout<<"Position of get pointer before reading:"<< file1.tellg() <<endl;

intiter = 5;

while (iter--) {

charch;

file1 >>ch; *// Read from file.*

cout<<ch;

}

cout<<endl<<"Position of get pointer after reading:"<< file1.tellg();

file1.close();

}

**Output:**

Position of put pointer before writing:0

Position of put pointer after writing:14

Position of get pointer before reading:0

Hello

Position of get pointer after reading:5

**Explanation:**

* Before writing anything to the file, it was opened in the out mode; hence the put pointer was at 0.
* After writing the string Hello Everyone, the put pointer will reach to end of the file, which is 14.
* For reading, the get pointer is used, and the initial position of the get pointer is 0.
* After reading five characters from the file, the get pointer reaches 5.

### seekg() &seekp()

* istream&seekg (streampospos), this function returns the istream object by changing the position of **get pointer** to pos.
* istream&seekp (streampospos), this function returns the ostream object by changing the position of **put pointer**.
* We could also overload the seekg() &seekp() by providing an offset. Pointers will move with respect to offsets, i.e., ios\_base::beg to start from the beginning of the file, ios\_base::end to start from the ending of the file, ios\_base::curr to start from the current positions of the pointer.
* The default value of offset is the beginning of the file.

**Example:**

#include <fstream>

#include <iostream>

usingnamespacestd;

int main() {

fstreammyFile("myfile.txt", ios::out);

myFile<<"123456789";

myFile.seekp(5);

myFile<<"\*";

myFile.close();

myFile.open("myfile.txt", ios::in);

myFile.seekg(3);

std::stringmyline;

while (myFile.good()) {

std::getline (myFile, myline);

std::cout<<myline<<std::endl;

}

myFile.close();

}

**Output:**

45\*789

**Explanation:**

* Initially, we have written a string into a file named myfile.txt.
* Later, we have to change the position of the put pointer to the 5th index using seekp() and then write "\*" to the file, which will overwrite to file.
* Then, for the reading operation, we change the get pointer position to the 3rd index, which means reading will start from that position.
* As we can see from the output, the string started from the 3rdindex, and the 5th index is changed to '\*'.

othistream and ostream provide member functions for repositioning the file-position pointer. These member functions are seekg ("seek get") for istream and seekp ("seek put") for ostream.

The argument to seekg and seekp normally is a long integer. A second argument can be specified to indicate the seek direction. The seek direction can be ios::beg (the default) for positioning relative to the beginning of a stream, ios::cur for positioning relative to the current position in a stream or ios::end for positioning relative to the end of a stream.

The file-position pointer is an integer value that specifies the location in the file as a number of bytes from the file's starting location. Some examples of positioning the "get" file-position pointer are −

// position to the nth byte of fileObject (assumes ios::beg)

fileObject.seekg( n );

// position n bytes forward in fileObject

fileObject.seekg( n, ios::cur );

// position n bytes back from end of fileObject

fileObject.seekg( n, ios::end );

// position at end of fileObject

fileObject.seekg( 0, ios::end );

**FileI/OwithMemberFunctions:**

**SelfStudy:FormattedI/O,commandlinearguments**

1. ios:-

* ios stands for input output stream.
* This class is the base class for other classes in this class hierarchy.
* This class contains the necessary facilities that are used by all the other derived classes for input and output operations.

2. istream:-

* istream stands for input stream.
* This class is derived from the class ‘ios’.
* This class handle input stream.
* The extraction operator(>>) is overloaded in this class to handle input streams from files to the program execution.
* This class declares input functions such as get(), getline() and read().

3. ostream:-

* ostream stands for output stream.
* This class is derived from the class ‘ios’.
* This class handle output stream.
* The insertion operator(<<) is overloaded in this class to handle output streams to files from the program execution.
* This class declares output functions such as put() and write().

4. streambuf:-

* This class contains a pointer which points to the buffer which is used to manage the input and output streams.

5. fstreambase:-

* This class provides operations common to the file streams. Serves as a base for fstream, ifstream and ofstream class.
* This class contains open() and close() function.

6. ifstream:-

* This class provides input operations.
* It contains open() function with default input mode.
* Inherits the functions get(), getline(), read(), seekg() and tellg() functions from the istream.

7. ofstream:-

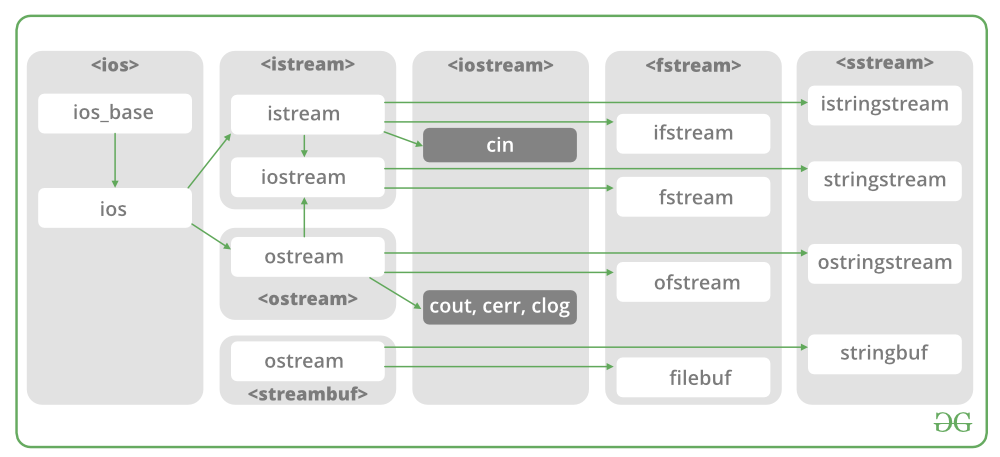
* This class provides output operations.
* It contains open() function with default output mode.
* Inherits the functions put(),  write(), seekp() and tellp() functions from the ostream.

8. fstream:-

* This class provides support for simultaneous input and output operations.
* Inherits all the functions from istream and ostream classes through iostream.

9. filebuf:-

* Its purpose is to set the file buffers to read and write.
* We can also use file buffer member function to determine the length of the file.



# File Handling in C++

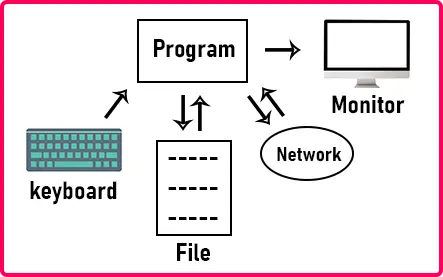
Back to: [C++ Tutorials For Beginners and Professionals](https://dotnettutorials.net/course/cpp-tutorials-for-beginners-and-professionals/)

## **File Handling in C++ with Examples**

In this article, I am going to discuss **File Handling in C++** with Examples. Please read our previous article where we discussed [**Virtual Destructors in C++**](https://dotnettutorials.net/lesson/virtual-destructors-in-cpp/) with Examples. Files are used to store data in a storage device permanently.

##### **What are Streams in C++?**

Before learning File Handling in C++, let us first learn about Streams in C++. A stream is a flow of data or a flow of characters. Streams are used for accessing the data from outside the program that is from external sources or destinations. So, data can be transferred from an external source to a program or from a program to a destination that is an external device. From accessing the data from outside the program we use streams. We can visualize it by the following diagram.



If we have a program, then the program may be getting the data from the keyboard or program may be sending the data to the monitor or the program may be accessing the data from an external file. The program can read or write data in the file. The program can be accessing the data from the network. So, there are various sources where the program can send and receive the data.

For accessing these sources or getting or sending the data from the program, we use the mechanism of the Streams in C++. So, there are I/O streams that are Input-output streams. For reading the data from the keyboard we use the input stream and for sending the data to the monitor we use the output stream.

And for the file, we need an input-output stream. There are built-in classes available in C++ for accessing the input-output stream. Class iostream is the input-output stream. From ios, the classes are coming out that are **istream** class for input stream and **ostream** class for the output stream. For a better understanding, please have a look at the below image.



Now similarly, for file access also, there are classes available that are **ifstream** for input stream and **ofstream** for the output stream.

For input stream from the keyboard, we have already a built-in object present in iostream header file that is **cin** and also an extraction operator that is “>>”. So directly we can use cin with extraction operator. Cin is an object of istream class. And we have another object that is cout that we commonly used for displaying the text on the screen. **cout** is an object of ostream. cout is used with the insertion operator “<<”. If you want to print something on the screen then you have to insert your text in cout. So that’s why we use the insertion operator. Similar operators such as insertion and extraction operators are used for files also. So, this is a very easy mechanism provided by C++. And also, we can overload the insertion and extraction operator.

##### **File Writing in C++:**

Let us write a program for writing data into a file. For that, we need a header file that is fstream. Please have a look at the below code.



Here we have the main function. Suppose we have a file “**my.txt**” on our computer and we want to write something in the my.txt file. For that, we have written **ofstream outfile(“my.txt”)**. As a constructor, we have mentioned the filename to the object outfile. So, this object outfile will associate with the file my.txt. Now whatever you will write into this outfile object it will get dropped into that file. It’s like you have connected a pipe from your program to that file on the disk. So, whatever you drop in the outfile it will get stored in the file.

Next thing, when we were opening this file through outfile object, the file should be already existing. If it is existing then outfile will open the file and if it is not existing then it will create a new file with the name **my.txt**. This is the important thing about the output file or output stream.

Next point, if the already file is existing and it is having some content then what will happen to that content? The outfile will truncate that content or remove the content. Suppose my.txt has some content, **my.txt:**

**100**  
**200**  
**300**

So, we have these numbers in the my.txt file. These values will be removed and fresh content will be written. But we don’t want to lose that content so what should we do? If you want to append the new content after the old content then there is a mode available that is **ios::app**. ‘app’ stands for append. So, we can write like,

**ofstream outfile(“my.txt”, ios::app);**

In this way, we can add new content to the file without losing the existing content. So like this mode, there are two modes available that are **ios::app**and**ios::trunc**. ‘trunc’ stands for truncate.

So if you want to remove the existing content, you can use ios::trunc. By default, the truncate mode is taken otherwise you can mention appending. Here in our example, we don’t want to append, so we don’t write append mode in our program. So outfile is the object associated with my.txt file. We will write the following statement in our main function,

**outfile << “Hello” << endl;**

This command will insert “Hello” in the my.txt file. Then we will write,

**outfile << 25 << endl;**

Here 25 will be written in a new line after “Hello”. So, if you want more then you can add more content to the file. Now my.txt file contains,

**Hello**  
**23**

##### **How to Close the Stream in C++?**

After writing the content, you must close the file by writing the following statement.

**outfile.close();**

This statement will close the stream and the file will be free from the program. It is important to close the file. See sometimes, you have connected a pen drive or memory card to your laptop and you have kept one of the files open and you try to eject the card. Then you will get a message that “some program is using your memory card or pen drive”. So, you cannot eject it. It means the operating system knows that some program is using some file so it will not allow you to eject. It means that files are being occupied or that resource is in use. So, when the program is not used then it should be released. If the program end, then automatically the resource will be released. But if you say file close, then the file should be closed. So, it is a good practice to close the resource when you have finished.

##### **Example to Understand File Handling in C++:**

The following example will create a File with the name my.txt (as it is not already existing) and then write three lines of content in it using the C++ File Handling Mechanism. Now, you may have one questions about which location the file is going to be created. It is created in the same location where the project is created. I have created the following project with the name FileHandlingDemo.cpp in **D:\Projects\Cpp\FileHandlingDemo** location. So, the my.txt file is also going to be generated in the same **D:\Projects\Cpp\FileHandlingDemo** location.

#include<iostream>

#include<fstream>

using namespace std;

int main()

{

ofstream outfile ("my.txt", ios::trunc);

outfile << "Joy" << endl;

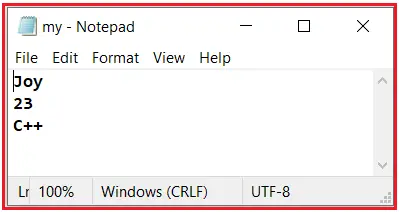
outfile << 23 << endl;

outfile << "C++" << endl;

outfile.close();

}

So, after the execution of this program, we will have a file that is my.txt which has the following content.



In the above example, as we are using **ios::trunc** option, so every time we execute the program, the old content is replaced by the new content. Here, trunc means truncating the old data. Let us see this. Please modify the content as follows:

#include<iostream>

#include<fstream>

using namespace std;

int main()

{

ofstream outfile ("my.txt", ios::trunc);

outfile << "Hellp" << endl;

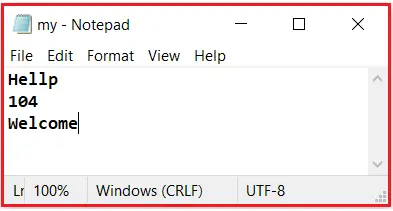
outfile << 104 << endl;

outfile << "Welcome" << endl;

outfile.close();

}

First, close the opened my.txt file and then run the above code. This time it will not create the my.txt file as this file already exists in the project folder. If you look at the content of the my.txt file, it will remove the old content and add the new content as shown in the below image.



Now, if you don’t want to remove the old data instead you want to append the new data below the old data, then you need to use the **ios::app** option as shown in the below example.

#include<iostream>

#include<fstream>

using namespace std;

int main()

{

ofstream outfile ("my.txt", ios::trunc);

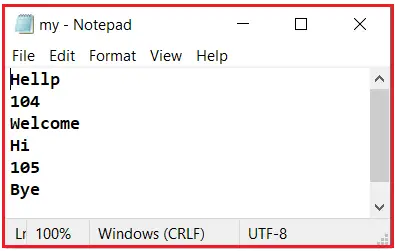
outfile << "Hi" << endl;

outfile << 105 << endl;

outfile << "Bye" << endl;

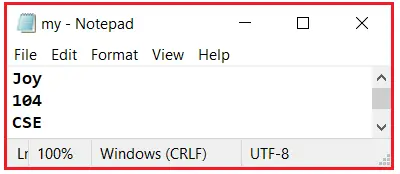
outfile.close();

}

Now, close the opened my.txt file and then run the above code and you will see that the new data is appended below the old data as shown in the below image.

##### **File Reading in C++:**

Now, we will write a program for reading the data from the files using C++ Language. Let’s say we have a file called my.txt with the following content in it in the same location where we created the project i.e. FileHandlingDemo.cpp.



We want to read the content from the above my.txt file. So let us see how to read the data from a file in C++ Language.

##### **How to Open a File in C++?**

Before reading the content of a file, we must first open that file. Please have a look at the following code.



Here we have written some code. Let us understand this step by step. Inside the main function, we have taken the ifstream object infile. Here we have just declared the object not mentioned in the file. Now we will see the other methods of opening a file.

Next, we use the open function to open the file my.txt. And we are using input file input stream so we are making it for reading purposes only. If you want to open the file for writing or reading purposes, there are some modes that are **ios::in** for reading the file and **ios::out**for writing in the file. These modes are also known as **flags**. By default, the open function will use read mode. So, we don’t want to mention any flag or mode there.

So, **infile.open(“my.txt”)**statement will open the file. Now one important thing, when you are reading from a file then **the file must be existing**. It will not create a new file so the file must be existing at the location where the program is saved. So, we have to check whether the file has opened or not. Next, we have checked for the file status. **if(!infile)**statement will check for whether the file is open or not. This condition will be true only if the file is not open. So, inside the if condition we can write the print statement “file cannot be opened”.

There is another method for checking the file status. **infile.is\_open()**,this method will return true if the file is opened otherwise it will return false. So, this is how you can check whether the file is open or not. Usually when you try to open some file then we get a prompt that the file is not existing or some other messages we get. So, like this only we have to check whether the file is open or not. For a better understanding, please have a look at the below example.

##### **How to Read the content of a File in C++?**

Now we want to read the content from the files. So, for that, the code is as follows:

#include <iostream>

#include <fstream>

using namespace std;

int main()

{

ifstream infile;

infile.open("my.txt");

string str;

int x;

infile >> str;

infile >> x;

cout << str << " " << x;

if (infile.eof())

cout << "end of file reached";

infile.close ();

}

Let us understand this step by step.

So, for reading the values from the files, we have taken two variables: **str** of type string and **x** of type int. We want to read from the files so we have used the extraction operator. **infile >> str**, this will read the first value that is ‘Joy’. On the next line, we have written, **infile >> x.**This will read the second value which is 104. So, the first value is a string and the second value is an integer. On the next line, we have printed these values.

So, we have extracted the values from the file in the program and printed those values on the screen. With the help of infile object, we have read the values from the file. That’s all.

##### **What is eof function in C++?**

One important thing, once we have finished reading the content, we have reached the end of the file (eof). So sometimes we need to check whether we have reached the end or not. There is a function available that will return true if we are at the end otherwise false. **infile.eof(),**this statement will return true if we are at the end of the file. So, we have written the print statement “end of file reached”.

Then at the last, we closed the file by writing **infile.close()**. So that’s all.

Note: We have learned how to write the data in the file and how to read the data from the file. Now one important thing, when we are reading the data from the file, we should already know in which order that data is present in the file. Then only we can read the data in the same form. So, there is some predefined order of writing the data available like a jpg file. So, there is some known order. So that’s why we say **File Format**. So, if data is written in the known format then only the programs can read the data in that particular format and show you the contents. Like jpg file can be opened anywhere. If you know the format then you can read the data. If you want to read the jpg file and display the image in your program then you should know the format of jpg files. Likewise, for pdf, you should know the format of pdf.

##### **Example to Understand File Reading in C++:**

#include<iostream>

#include<fstream>

using namespace std;

int main()

{

ifstream ifs;

ifs.open ("my.txt");

if (ifs.is\_open())

cout << "File is Opened" << endl;

string name;

int roll;

string branch;

ifs >> name >> roll >> branch;

ifs.close();

cout << "Name: " << name << endl;

cout << "Roll: " << roll << endl;

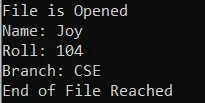
cout << "Branch: " << branch << endl;

if(ifs.eof())

cout << "End of File Reached";

}

###### **Output:**



# File Input Output Operations In C++

**A Study On File Input Output Operations & File Pointer Functions In C++.**

In real-time programming, we deal with large chunks of data that cannot be accommodated from standard Input-Output devices. Hence we need to make use of secondary storage for storing data. Using secondary storage we usually store data in the form of files.

We can read data from files or write data into files by using a sequence of data called streams either in the text or binary format. There are various input /output and other operation related to files in C++. This tutorial explains these operations related to files using various classes.

**=>** [**Visit Here For The Exclusive C++ Training Tutorial Series.**](https://www.softwaretestinghelp.com/cpp-tutorials/)

What You Will Learn: [[hide](https://www.softwaretestinghelp.com/file-input-output-in-cpp/)]

* [File Input/Output Classes In C++](https://www.softwaretestinghelp.com/file-input-output-in-cpp/" \l "File_InputOutput_Classes_In_C)
  + [Open A File](https://www.softwaretestinghelp.com/file-input-output-in-cpp/" \l "Open_A_File)
  + [Closing A File](https://www.softwaretestinghelp.com/file-input-output-in-cpp/" \l "Closing_A_File)
  + [Reading From A File](https://www.softwaretestinghelp.com/file-input-output-in-cpp/" \l "Reading_From_A_File)
  + [Writing To A File](https://www.softwaretestinghelp.com/file-input-output-in-cpp/" \l "Writing_To_A_File)
  + [File State Slags](https://www.softwaretestinghelp.com/file-input-output-in-cpp/" \l "File_State_Slags)
  + [Get/Put And Other Special Operations](https://www.softwaretestinghelp.com/file-input-output-in-cpp/" \l "GetPut_And_Other_Special_Operations)
  + [Conclusion](https://www.softwaretestinghelp.com/file-input-output-in-cpp/" \l "Conclusion)
  + [Recommended Reading](https://www.softwaretestinghelp.com/file-input-output-in-cpp/" \l "Recommended_Reading)

## File Input/Output Classes In C++

We have seen an iostream class in C++ which defines the standard input and output functionality including cin and cout. This class is limited to the standard input and output devices like keyboard and monitor respectively.

When it comes to file operations, C++ has a different set of classes that can be used.

**These classes are described as below:**

* **Ofstream:** File handling class that signifies the output file stream and is used for writing data to files.
* **Ifstream:** File handling class that signifies the input file stream and is used for reading data from the file.
* **Fstream:** File handling class that has the ability to handle both ifstream and ofstream. It can be used to read from and write to a file.

**The following operations are supported, in C++ File Handling:**

* Open a file
* Close a file
* Read from a file
* Write to a file

**Let us see each of these operations in detail!!**

### Open A File

Associating object of one of the stream classes to a file either for reading or writing or both is called opening a file. An open file is represented in code by using this stream object. Thus any reading/writing operation performed on this stream object will be applied to the physical file as well.

**The general syntax to open a file with the stream is:**

void open(const char\* filename, ios::open mode mode)

Here,

filename => The string containing path and name of the file to be opened.

mode => Optional parameter indicating the mode in which the file is to be opened.

C++ supports various modes in which the file can be opened. We can also specify a combination of these modes using the OR operator.

| File mode | Description |
| --- | --- |
| ios::in | Opens the file in input mode for reading. |
| ios::out | Opens the file in output mode for writing data to file. |
| ios::ate | Set initial position at the end of the file. If the end of file flag is not set, the initial position is set to the beginning of the file. |
| ios::trunc | If the file is opened for writing and already has contents, the contents are truncated. |
| ios::app | Opens the file in append mode such that all contents are appended at the end of the file. |
| ios::binary | Opens file in binary mode. |

**For Example, if we want to open a file “myfile.dat” for appending data in binary mode, then we can write the following code.**

ofstream myfile;

myfile.open(“myfile.dat”, ios::out|ios::app|ios::binary);

As already mentioned, the mode parameter is optional. When we open a file without specifying the second parameter, an open member function of ofstream, ifstream or fstream has a default mode to open the file with.

**These are given as follows:**

| Class | Default mode |
| --- | --- |
| Ifstream | ios::in |
| ofstream | ios::out |
| Fstream | ios::in|ios::out |

So, if we do not specify the second parameter in the open function, depending on the stream class used, the file is opened with the default mode.

### Closing A File

We can use the close function to close a file and release the resources held by the file when we are done with the input and output operations on a file.

**Function to close a file is:**

void close()

So, when we are done with the operations on the above file myfile, we can close the file as follows:

myfile.close();

Once the file is closed using the close function, the file object associated can be re-used to open another file.

### Reading From A File

We can read the information from a file line by line using the stream extraction operator (>>). This is similar to reading input from the standard input using cin. The only difference being in case of files, we use ifstream or fstream object instead of cin.

**Sample code for reading from a file is given below:**

|  |
| --- |
| ifstream myfile;  myfile.open(“samp\_file.txt”);  cout<<”Reading from a file”<<endl; myfile>>data;  cout<<data<<endl;  myfile.close(); |

In the above code, we open a file and using the stream extraction operator (>>), we read the contents of the file. Once done with reading, we can close the file.

### Writing To A File

We can also write data to a file using the file operations. The operator we use to write data to a file is a stream insertion operator (<<). Once again this is the same operator that we use to print data to a standard output device using cout. Difference between the two is that for file related writing we use ofstream or fstream object.

**Let us consider the following Example code:**

|  |
| --- |
| char data[100];  ofstream myfile;  myfile.open(“samp\_file.txt”);  cout<<”Enter the string to be written to file”<<endl;  cin.getline(data, 100);  myfile<<data<<endl;  myfile.close(); |

Here, we read a line from the input and write it to a file that was opened with the ofstream object.

**In the code example below, we provide a demonstration of all the file handling operations.**

|  |
| --- |
| #include <fstream>  #include <iostream>  using namespace std;    int main ()  {      char data[100];        // opening a file in write mode.      ofstream myfile;      myfile.open("E:\\message.txt");        cout << "Writing to the file" << endl;      cout << "Enter your name: ";      cin.getline(data, 100);        myfile << data << endl;        cout << "Enter your age: "; cin >> data;      cin.ignore();        myfile << data << endl;        // close the opened file.      myfile.close();       // opening a file in read mode.     ifstream infile;     infile.open("E:\\message.txt");       cout << "Reading from a file" << endl; infile >> data;       cout << data << endl; infile >> data;     cout << data << endl;       infile.close();       return 0;  } |

**Output:**

Writing to the file  
Enter your name: Ved  
Enter your age: 7  
Reading from a file  
Ved  
7

In the above program first, we open a file in the write mode. Then we read data i.e. name and age and write it to a file. We then close this file. Next, we open the same file in the read mode and read the data line by line from the file and output it to the screen.

Thus this program covers all the file I/O operations.

### File State Slags

There are some member functions that are used to check the state of the file. All these functions return a Boolean value.

**We have tabularized these functions as follows:**

| Function | Description |
| --- | --- |
| eof() | Returns true if the end of file is reached while reading the file. |
| fail() | Returns true when read/write operation fails or format error occurs |
| bad() | Returns true if reading from or writing to a file fail. |
| good() | Returns  false  in the same cases in which calling any of the above functions would return  true. |

### Get/Put And Other Special Operations

The file I/O streams that we have seen so far have an internal get and put positions similar to the other I/O streams like iostream.

The class ifstream has an internal get position that contains the location of the element/character to be read in the file in the next input operation. The class ofstream has an internal put position that contains the location of the element/character to be written in the next output operation.

Incidentally, fstream has both get and put positions.

To facilitate reading and writing using these positions, we have a few member functions that are used to observe and modify these positions.

**These functions are listed below:**

| Functions | Description |
| --- | --- |
| tellg() | Returns current position of get pointer |
| tellp() | Returns current position of put pointer |
| seekg(position) | Moves get a pointer to specified location counting from the beginning of the file |
| seekg(offset,direction) | Moves get a pointer to offset value relative to the point given by parameter direction. |
| seekp(position) | Moves put a pointer to specified location counting from the beginning of the file |
| seekp(offset, direction) | Moves put a pointer to offset value relative to the point given by parameter direction. |

The parameter direction given in the above function prototypes is an enumerated type of type seekdir and it determines the point from which the offset is counted.

**It can have the following values.**

| ios::beg | Offset from beginning of the stream |
| --- | --- |
| ios::cur | Offset from current position |
| ios::end | Offset from the end of the stream |

**Let us see a complete Example that demonstrates the usage of these functions.**

|  |
| --- |
| #include <iostream>  #include <fstream>  using namespace std;    int main()  {      fstream myfile;      myfile.open("E:\\myfile.txt",ios::out);  if(!myfile)     {           cout<<"Cannot create File...";      }    else    {          cout<<"New file created"<<endl;          myfile<<"This is file input output tutorial";    cout<<"Initial File Pointer Position at: "<<myfile.tellp()<<endl;           myfile.seekp(-1, ios::cur);    cout<<"After seekp(-1, ios::cur), File Pointer Position at: "<<myfile.tellp()<<endl;            myfile.close();    }    myfile.open("E:\\myfile.txt",ios::in);    if(!myfile)    {         cout<<"Cannot open File...No such file";    }  else    {       char ch;       myfile.seekg(5, ios::beg);  cout<<"After seekg(5, ios::beg), File Pointer at: "<<myfile.tellg()<<endl;          cout<<endl;          myfile.seekg(1, ios::cur);  cout<<"After seekg(1, ios::cur), File Pointer at: "<<myfile.tellg()<<endl;          myfile.close();    }    return 0;    } |

**Output:**

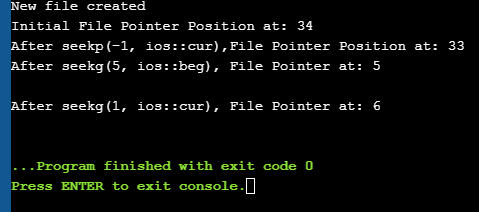
New file created

Initial File Pointer Position at: 34

After seekp(-1, ios::cur),File Pointer Position at: 33

After seekg(5, ios::beg), File Pointer at: 5

After seekg(1, ios::cur), File Pointer at: 6

[](https://www.softwaretestinghelp.com/wp-content/qa/uploads/2019/05/fileIO.png)

As shown in the above program, we have a file created in which we write a line of text. Then using the various functions described above, we display various positions of the File Pointer.