# Software Induction:

## File Handling in C++

Version: 1.0

Prepared By:

Prashant Singh

**Vehant Technologies Private Limited**

B-24, Sec 59, Noida,

UP - 201301

INDIA

## Table Of Content

Overview 4
Scope 4
File 4
File Input/Output Classes in C++ 4
Files And Streams 5
What is File Handling in C++? 5
    ofstream 6
    ifstream 6
    fstream 6
Opening a File 7
    Opening Modes 7
        1. Open a File Using Constructor 8
        2. Open a File Using stream.open () Method 8
Reading from a File 9
    1. Using get () Method 10
        Output: 10
        Explanation: 10
    2. Using getline () Method 10
        Output: 11
        Explanation: 11
Writing to a File 12
    1. Writing in Normal Write Mode 12
        Output: 13
        Explanation: 13
    2. Writing in Append Mode 13
        Output: 13
        Explanation: 13
    3. Writing in Truncate Mode 14
        Output 14
        Explanation 14
Closing a File 14
File Position Pointers 15
    tellp () & tellg () 15
        Example: 15
        Output: 16
        Explanation: 16
    seekg () & seekp () 16
        Example: 17

Output: 17
Explanation: 17
Checking State Flags 18
Flushing a Stream 18
Conclusion 18
SUMMARY 19

# File Handling in C++

## Overview

A file is a basic entity that stores the user's relevant data. The concept of file I/O in programming refers to reading and writing files stored in a secondary storage device through a program. The C++ programming language provides several classes for file I/O operations, including ofstream, ifstream, and fstream

## Scope

- This article will discuss the file input-output classes in c++.
- Subsequently, we will do file handling in a C++ program, like opening a file, reading a file, and writing to a file with those classes.
- Before coming to an end, we will discuss related concepts like file position pointer, stream flush, etc.

## File

"A file is a logical collection of records where each record consists of a number of items known as fields".

The records in a file can be arranged in the following three ways:

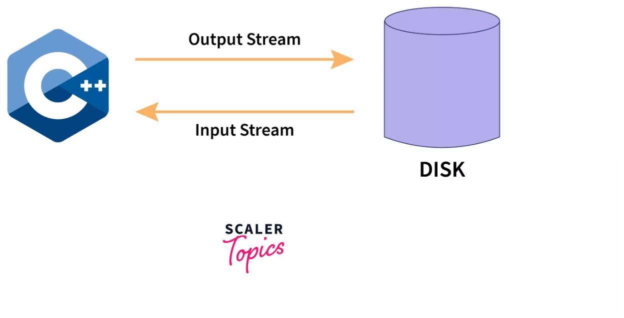
**Ascending/Descending order:** The records in the file can be arranged according to ascending or descending order of a key field.

**Alphabetical order: I**f the key field is of alphabetic type then the records are arranged in alphabetical order.

**Chronological order:** In this type of order, the records are stored in the order of their occurrence i.e. arranged according to dates or events. If the key-field is a date, i.e., date of birth, date of joining, etc. then this type of arrangement is used.

## File Input/Output Classes in C++

Before understanding file, I/O, let's discuss some basic concepts of programming, A program is a set of instructions that manipulates some data according to the specified algorithm. The data is then used for several purposes. Whenever we execute a program, it gets loaded into the main memory, and all data required for the program also exists in the main memory. But in real-world programming, manipulation exists on a large data set (typically in Gigabytes), so we will not store all data on Main Memory. The extensive data remains stored on disk, and through a stream, we bring only a particular set of data that the program needs now.



## Files And Streams

In C++, a stream is a data flow from a source to a sink. The sources and sinks can be any of the input/output devices or files.

For input and output, there are two different streams called input stream and output stream.

| Stream | Description |
|---|---|
| cin | standard input stream |
| cout | standard output stream |
| cerr | standard error stream |

The standard source and sink are keyboard and monitor screen respectively

## What is File Handling in C++?

The basic entity that stores the user's relevant data is called a file. Files can have a lot of types that are depicted by their extensions. For example .txt (text file), .cpp (c++ source file), .exe (executable file), .pdf (portable document file), and many more.

File Handling stands for the manipulation of files storing relevant data using a programming language, which is C++ in our case. This enables us to store the data in permanent storage even after the program performs file handling for the same ends of its execution. C++ offers the library fstream for file handling. Here we will discuss the classes through which we can perform I/O operations on files. For taking input from the keyboard and printing something on the console, you might have been using the cin(character input stream) and cout(character output stream) of the istream and ostream classes. File streams are similar to them. Only the console here is replaced by a file.

Below are three stream classes of the fstream library for file handling in C++ that are generally used for file handling in C++.

## ofstream

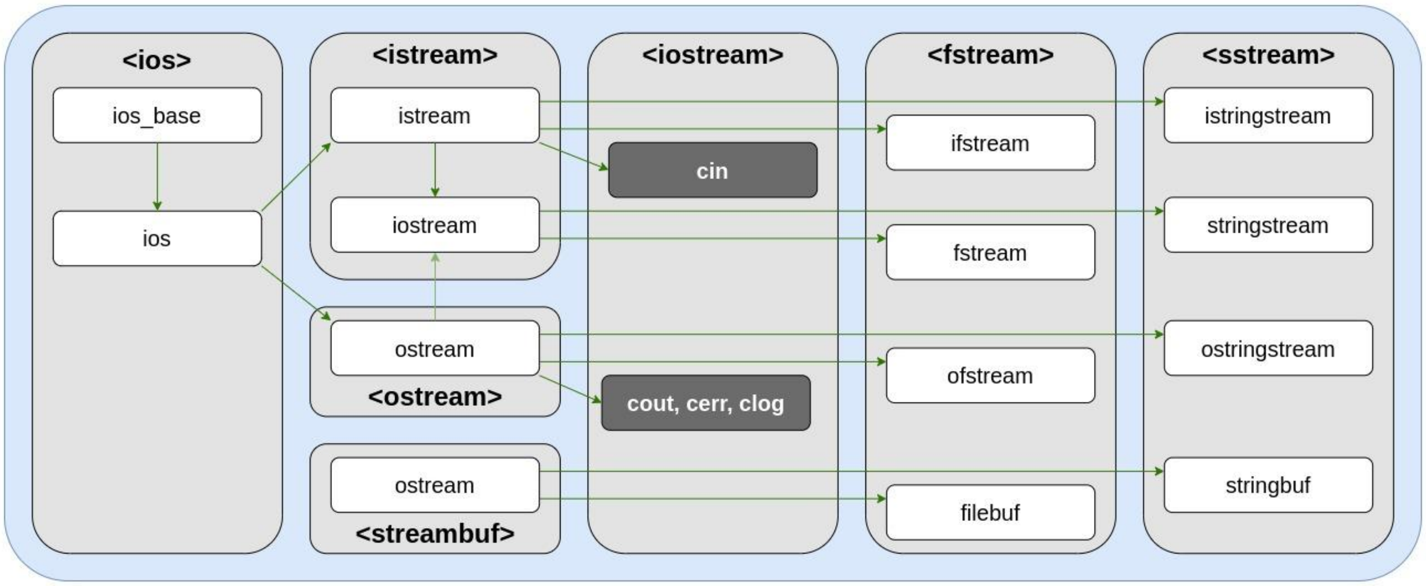
The ofstream is derived from the ostream class. It provides the output stream to operate on file. The output stream objects can be used to write the sequences of characters to a file. This class is declared in the fstream header file.

## ifstream

The ifstream is derived from the istream class. It provides the input stream to operate on file. We can use that input stream to read from the file. This class is declared in the fstream header file.

## fstream

The fstream is derived from the iostream class, and the iostream is further derived from the istream and ostream classes. It provides the input as well as output streams to operate on file. If our stream object belongs to the fstream class, we can perform read and write operations on the file with the same stream object. This class is declared in the fstream header file.



## Opening a File

To start working with the file, first, we need to open it in our program. We can either open our file with the constructor provided by the file I/O classes or call the open method on the stream object. Before discussing how the file can be opened, it is necessary to discuss several opening modes.

### Opening Modes

There are several modes we can specify when opening a file. These modes correspond to various controls given to stream objects during file handling in c++. The opening mode's description and syntax are in the tabular form below.

| Mode | Syntax | Description |
|---|---|---|
| Read | ios::in | Opens file for reading purpose. |
| Write | ios::out | Opens a file for writing purposes. |
| Binary | ios::binary | All operations will be performed in binary mode. |

| | | |
|---|---|---|
| Truncate before Open | ios::trunc | If the file already exists, all content will be removed immediately. |
| Append | ios::app | All provided data will be appended to the associated file. |
| At End | ios::ate | It opens the file and moves the read/write control at the End of the File. The basic difference between the ios::app and this one is that the former will always start writing from the end, but we can seek any specific position with this one. |
| No Create | ios::nocreate | Opens the file only if it already exists |
| No Replace | ios::noreplace | Opens the file only if it does not already exist |

## 1. Open a File Using Constructor

Each class has two types of constructors: **default** and those that specify the opening mode and the associated file for that stream.

```
ifstream Stream_Object(const char* filename, ios_base::openmode = ios_base::in);
```

```
ofstream Stream_Object(const char* filename, ios_base::openmode = ios_base::out);
```

```
fstream Stream_Object(const char* filename, ios_base::openmode mode = ios_base::in |
ios_base::out);
```

## 2. Open a File Using stream.open () Method

The open() is a public member function of all these classes. Its syntax is shown below.

```
void open (const char* filename, ios_base::openmode mode);
```

The open() method takes two arguments: the file name and the mode in which the file will open.

The is_open() method is used to check whether the stream is associated with a file. It returns true if the stream is associated with some file; otherwise returns false.

```
bool is_open();
```

## Reading from a File

We read the data of a file stored on the disk through a stream. The following steps must be followed before reading a file,

- Create a file stream object capable of reading a file, such as an object of ifstream or fstream class.

```
ifstream streamObject;
// Or
fstream streamObject;
```

- Open a file through the constructor while creating a stream object or calling the open method with a stream object.

```
ifstream streamObject("myFile.txt");
// Or
streamObject.open("myFile.txt");
// Note:- If a stream is already associated with some file, then the call to open method
will fail.
```

- Check whether the file has been successfully opened using is_open(). If yes, then start reading.

```
if(streamObject.is_open()){
  // File Opened successfully.
}
```

## 1. Using get () Method

```
#include <fstream>
#include <iostream>

int main () {
   std::ifstream myfile("sample.txt");

   if (myfile.is_open()) {
      char mychar;
      while (myfile.good()) {
         mychar = myfile.get();
         std::cout << mychar;
      }
   }
   return 0;
}
```

Output:

```
Hi, this file contains some content.
This is the second line.
This is the last line.
```

Explanation:

- First, we have created a stream object of the ifstream class and are providing the file name to open it in reading mode(default).
- Next, we check whether the file is being opened successfully. If yes, we read one character at a time until the file is good.
- The good() function returns true if the end of the file is not reached and there is no failure.

## 2. Using getline () Method

```
#include <fstream>
#include <iostream>
```

```
#include <string>

int main () {
   std::ifstream myfile("sample.txt");

   if (myfile.is_open()) {
      std::string myline;
      while (myfile.good()) {
         std::getline (myfile, myline);
         std::cout << myline << std::endl;
      }
   }
   return 0;
}
```

Output:

```
Hi, this file contains some content.

This is the second line.

This is the last line.
```

Explanation:

- At the beginning of the program, we opened the file with the constructor of the ifstream class.
- If the file is successfully opened, the 'open' function will return true, and the if block will be executed.
- In the while loop, we check whether the file stream is good for operations. When the end of the file is reached, the good function will return false.
- We have declared a string to store each line of file through the getline function, and later we are printing that string.

## Writing to a File

In writing, we access a file on disk through the output stream and then provide some sequence of characters to be written in the file. The steps listed below need to be followed in writing a file,

- Create a file stream object capable of writing a file, such as an object of ofstream or fstream class.

```
ofstream streamObject;
// Or
fstream streamObject;
```

- Open a file through the constructor while creating a stream object or calling the open method with a stream object.

```
ofstream streamObject("myFile.txt");
// Or
streamObject.open("myFile.txt");
```

- Check whether the file has been successfully opened. If yes, then start writing.

```
if(streamObject.is_open()){
   // File Opened successfully.
}
```

### 1. Writing in Normal Write Mode

```
#include <fstream>
#include <iostream>
#include <string>

int main () {
   // By default, it will be opened in normal write mode, ios::out.
   std::ofstream myfile("sample.txt");

   myfile << "Hello Everyone \n";
   myfile << "This content was being written from a C++ Program";
```

```
    return 0;
}
```

Output:

```
Hello Everyone

This content was written from a C++ Program.
```

Explanation:

- << operator is used for writing on the file.
- The text given above will be shown in our sample.txt after running the program.

## 2. Writing in Append Mode

```
#include <fstream>
#include <iostream>
#include <string>

int main () {
    std::ofstream myfile("sample.txt", std::ios_base::app);

    myfile << "\nThis content was appended in the File.";
    return 0;
}
```

Output:

```
Hello Everyone
This content was written from a C++ Program
This content was appended in the File.
```

Explanation:

- The same sample.txt, which was used in the last example, has more content appended to it now.

## 3. Writing in Truncate Mode

```
#include <fstream>
#include <iostream>
#include <string>

int main () {
   std::ofstream myfile("sample.txt", std::ios_base::trunc);

   myfile << "Only this line will appear in the file.";
   return 0;
}
```

Output

```
Only this line will appear in the file.
```

Explanation

- Again, we use the same sample.txt file from the previous Example. Now all older content is being removed.

## Closing a File

The concept of closing a file during file handling in c++ refers to the process of detaching a stream with the associated file on disk. The file must be closed after performing the required operations on it. Here are some reasons why it is necessary to close the file,

- The data might be in the buffer after the write operation, so closing a file will cause the data to be written in the file immediately.
- When you need to use the same stream with another file, it is a good practice to close the previous file.
- To free the resources held by the file.
- When the object passes out of scope or is deleted, the stream destructor closes the file implicitly.

## 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(). Also, we can seek(change) the position of the pointer with the function seekg() and seekp(). There, we can either read or write after seeking a particular position. Methods like seekg(), seekp() takes parameters as long integers and seek directions.
*A few examples are :*
***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)*

### 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>

using namespace std;

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;

    int iter = 5;
    while (iter--) {
       char ch;
       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 (streampos pos), this function returns the istream object by changing the position of get pointer to pos.
- istream& seekp (streampos pos), 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>
using namespace std;

int main() {
   fstream myFile("myfile.txt", ios::out);
   myFile << "123456789";

   myFile.seekp(5);
   myFile<<"*";
   myFile.close();
   myFile.open("myfile.txt", ios::in);
   myFile.seekg(3);
   std::string myline;

   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 3rd index, and the 5th index is changed to '*'.

## Checking State Flags

The state flags of the file tell about the current state of the file, and there are several functions to retrieve the current state.

- eof(), This function returns true if the end of the file is reached while reading the file.
- fail() returns true when the read/write operation fails or a format error occurs.
- bad() returns true if reading from or writing to a file fails.
- good() checks the state of the current stream and returns true if the stream is good for working and hasn't raised any error. good() returns false if any of the above state flags return true; otherwise, it returns true.

## Flushing a Stream

In C++, the streams get buffered by default for performance reasons, so we might not immediately get the expected change in the file during a write operation. To force all buffered writes to be pushed into the file, we can either use the flush() function or std::flush manipulator.

## Conclusion

- The file I/O in programming means interacting with the file on the disk to receive and provide the data.
- File Handling is the manipulation of files storing relevant data using a programming language( i.e., C++ in our case).
- We have three different classes: ifstream, ofstream, and fstream. These all are declared in the fstream header and provide us stream through which we can access the file and subsequently perform file handling in C++.

- To start working with a file, first, we have to open that file. We can do this either during the construction of the stream object or by calling the open() method on the stream object.
- There could be several modes of file opening we can choose according to our requirements.
- After working on the file, it is a good practice to close it with the help of the close() method of stream object.

## Summary

- Any thing stored on a permanent storage is called a file.
- A set of related data items is known as a record. The smallest unit of a record is called a field.
- A key field is used to uniquely identify a record.
- A file is a logical collection of records.
- In a serial file, the records are stored in the order of their arrival without regards to the key field.
- On the other hand, in a sequential file, the records are written in a particular order of the key field.
- The key field is also known as a primary key. 'ifstream' and 'ofstream' are input and output streams respectively.
- The objects that remember their data and information are called persistent objects.
- The function eof() returns 0 when it detects the end of file. A opened file must be closed after its usage.

## Assignment

**Q1)** Create a Student Record in a file and Provide Option of addition / deletion / Modification / Display of Data.
The Data Consist of:

-Student Name

-Roll Number

Data is sorted sorted by student Roll number. (Use File binary mode.)

**Q2)** Write a program to generate 10,00,000 random numbers and store them in a file. Limited Ram-Size capable of loading 1000 number in memory at a time.