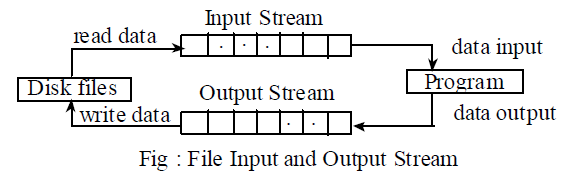
# File I/O

## Introduction

The I/O system of C++ handles file operations which are very much similar to the console input and output operations. A file stream is an interface between the programs and the files. The stream which supplies data to the program is called **input stream** and that which receives data from the program is called **output stream**. That is, the input stream reads or receives data from the file and supplies it to program while the output stream writes or inserts data to the file. This is illustrated in the figure.



## Classes for File Stream Operations

The I/O system of C++ contains several stream classes for handling file I/O operations:

* ***ifstream***(input file stream) class provides input operations on files. This class represents the input file stream and is used to read information from files. It contains open( ) with default input mode. Inherits the function get( ), getline( ), read( ), seekg( ) and tellg( ) function from istream class.
* ***ofstream****(output file stream)* class provides output operations on files. This class represents the output file stream and is used to create files and to write information to files. It Contains open( ) with default output mode. Inherits put( ), seekp( ), tellp( ) and write( ) function from ostream class.
* ***fstream****(file stream)* class supports for simultaneous input and output operations on files. This class represents the file stream generally, and has the capabilities of both ofstream and ifstream which means it can create files, write information to files, and read information from files. Inherits all the functions from istream and ostream classes.

### Summary:

* ofstream: Stream class to write on files
* ifstream: Stream class to read from files
* fstream: Stream class to both read and write from/to files.

## General File I/O Steps

1. Declare a file stream object using **ifstream, ofstream or fstream**. For reading only: **ifstream** is used, for writing only**: ofstream** is used, and for simultaneous reading-writing**: fstream** is used.
2. Connect the file stream object with the file name.
3. Open the file.
4. Use the file: **read/write or both**.
5. Close the file.

## File Operations

There are different types of operations that can be performed on a file, like opening, reading, writing, closing, etc. A file stream can be defined using the classes **ifstream**, **ofstream**, and **fstream** that are contained in the header file **fstream**.

To perform read-write operations in the file, first we need to open that file. A file can be opened in two ways:

* + **Using the constructor of the class**: suitable when we use only one file in the stream.
  + **Using the member function open( ):** suitable when we want to manage multiple files using one stream.

### Opening files Using Constructor

This method is used when we use only one file in a stream. While opening the file using constructor, we need to pass the desired filename as a parameter to the constructor. This involves following steps:

1. Create a file stream object for read or write operation.

2. Initialize the file object with desired filename in appropriate mode.

**Syntax:**

**f**ile\_stream\_class object\_name(“file\_name”, mode);

Here file\_stream\_class can be ifstream, ofstream, or fstream.

**For example:**

ofstream fout(“results.txt”) ; **//output only**

This statement creates an object fout (which can be any valid name like myfile, outfile, o\_file, etc) of ofstream class and attaches this object with a file “results.txt” which can be any valid name. In this results file, we can only write data because it is defined using **ofstream** class.

Similarly, a file can be opened for input or reading as,

ifstream fin(“test”) ; **//input only**

This creates an object fin(it may be any valid name like infile, myfile, etc) of ifstream and attaches a file “test” with it. In file “test” we can perform only input operation i.e. we can only read data because it is created using **ifstream** class.

**First Example Program**

**The first program, will create a file, and write some text into it.**

#include <fstream>

using namespace std;

int main()

{ **// ofstream constructor opens file.**

ofstream fout("cpp.txt"); **//creating output file stream object fout and connecting it to cpp.txt.**

fout << "Hello World, from Sagarmatha College!"; **//writing to the file, same as cout.**

fout.close(); **//closing file after writing.**

return 0;

}

**Program Description:**

This program will create the file cpp.txt in the directory from where you are executing it, and will put "Hello World, from Sagarmatha College!" into it.

* #include <fstream> - We need to include this file in order to use C++’s functions for File I/O. In this file, there are several classes, including ifstream, ofstream and fstream, which are all derived from istream and ostream.
* ofstream fout(“cpp.txt”); : This statement creates an object fout (*which can be any valid name like myfile, outfile,o\_file, etc*) of ofstream class and attaches this object with a file “cpp.txt” which can be any valid name. In this “cpp.txt” file, we can only write data because it is created by ofstream class. A filename is used to initialize the file stream object as fout.
* fout.close(); : As we have opened the stream, when we finish using it, we have to close it. fout is an object from class ofstream, and this class (ofstream) has a function that closes the stream. That is the close() function. So, we just write the name of the object, dot operator and close(), in order to close the file stream.

**File Modes:**

* File modes specifies how file will be opened. Different file opening modes in C++ are as follows:

M**odes Functions**

ios: :app Start reading or writing at end of file (Append)

ios: : in Open file for reading only (default for ifstream)

ios: : out open file for writing only (default for ofstream)

ios: : binary open file in binary (not text) mode

ios :: ate go to end of file on opening

ios :: trunc If file is present erases its content, otherwise creates a new file.

We can combine two or more modess by **bitwise OR ( | )**ing them together. For example if we want to open a file in write mode and want to truncate it in case it already exists, following will be the syntax:

ofstream outfile("file.dat", ios::out | ios::trunc );

Similarly, following statement will open a file student.dat in read, and binary mode.

ifstream file(("student.dat", ios::in | ios::binary);

**Reading from file:**

**Sample Program0:**

#include<fstream>

#include<iostream>

#include<string>

using namespace std;

int main()

{

ifstream fin("data.txt"); //file is opened using ifstream class so can perform read operation only.

if(!fin) //check for file.

{

cout << "File could not found" << endl;

}

string item;

while(!fin.eof()) //read from file until end of file.

{

fin>>item; //Remember it cannot read whitespaces.

cout<<item; //displays the word to the monitor.

}

fin.close();

return 0;

}

**To read one entire line with embedded blanks use following:**

while(!fin.eof()) //read from file until end of file.

{

getline(fin, item); //Can read one line of text with whitespaces.

cout<<item; //displays on the screen.

}

**To read multiple lines with embedded blanks use following:**

while(!fin.eof()) //reads the file until end of file.

{

getline(fin,item,'$');

cout<<item; //displays on the screen.

}

**Sample Program1:**

using namespace std;

int main()

{

ifstream fin("kk.txt"); //file is opened using ifstream object so can perform read operation only.

if(!fin) //check for file.

{

cout << "File could not found" << endl;

}

string item;

while(!fin.eof()) //read till end of file.

{

fin>>item; //read one word at a time from file and assigns it to item.

cout<<item<<endl; //displays the string on the screen.

}

fin.close();

return 0;

}

### Opening a file using open( ) function

The open( ) function can also be used to open files.

**General syntax:**

file\_stream\_class stream\_object\_name;

stream\_object\_name.open (“filename”, mode);

The second argument (called file mode parameter) specifies the purpose for which the file is opened. The mode can combine two or more parameters using the bitwise OR operator ( | ).

**Examples:**

1) fstream file ;

file.open(“Person.Dat”, ios: : app|ios: : out|ios: : in);

It creates a Person.Dat file for input and output in append mode.

2) ifstream infile ;

infile.open(“Test.txt”, ios: : in) ; **//This opens Test.txt file for reading only.**

3) ofstream outfile ;

outfile.open(“Test.txt”, ios: : out) ; **//This opens Test.txt for writing only.**

**Example:**

#include <fstream>

#include<iostream>

using namespace std;

int main()

{

string data, rdata;

cout<<"Enter a string: "<<endl;

getline(cin, data)

ofstream fout; //can perform write operation.

fout.open("String.txt", ios::out|ios::app);

fout<<data<<endl;

fout<<"Well done!! You entered some text";

fout.close();

ifstream fin;

fin.open("String.txt", ios::in)

while(!fin.eof())

{

getline(fin, rdata); //read a line from file and assigns it to rdata.

cout<<rdata; //displaying it on screen.

}

fin.close();

return 0;

}

## File Pointers and Their Manipulations

Each file has two automatic pointers known as the file pointers. One of them is called the input pointer (or **get pointer**) and the other is called the output pointer (or **put pointer**). When input and output operation takes place, the appropriate pointer is automatically set according to mode. For example when we open a file in reading mode file pointer is automatically set to start of file. And when we open in append mode the file pointer is automatically set at the end of file.

In C++ there are some manipulators by which we can control the movement of pointer. The available manipulators in C++ are:

1. seekg( )
2. seekp( )
3. tellg( )
4. tellp( )

Following section shows their details:

1. **seekg(n):** Moves get pointer(input) to a specified location from the start.
   1. **fileObject.seekg(0)**: Goes to the start of the file for reading.
   2. **fileObject.seekg(n)**: Goes to nth byte from beginning for reading.
2. **seekp(n):** Moves put pointer(output) to a specified location from the start.
   1. **fileObject.seekp(0)**: Goes to the start of the file for writing.
   2. **fileObject.seekp(n)**: Goes to nth byte from beginning for writing.
3. **tellg():** Gives the current position of the get pointer. It returns an integer value that specifies the current location of the get pointer.
   1. int p = fileObject.tellg(); **//returned value is assigned to p.**
4. **tellp():** Gives the current position of the put pointer. It also returns an integer value that specifies the current location of put pointer.
   1. int p = fileObject.tellp(); **//returned value is assigned to p.**

**Consider following Example:**

int main()

{

ifstream fin;

fin.open("cpp.txt"); //open already created file for reading.

fin.seekg(4); //moving get pointer to the 4th position in the file

string data;

while(!fin.eof()) //it will read all the data after 4th position in the file.

{

fin>> data;

cout<<data<<endl;

}

fin.close();

}

**Consider the following statements:**

ifstream fin;

fin.open("cpp.txt"); **//while opening, get pointer is at the start of the file automatically.**

int p=fin.tellg(); **//determines the current position of the get pointer, i.e., 0.**

cout<<" current position of the get pointer is: "<<p<<endl; **//displays 0.**

fin.seekg(4); **//now get pointer is moved to 4th position.**

p=fin.tellg(); **//determines the current position of the get pointer, i.e., 4.**

cout<<" current position of pointer is: "<<p<<endl<<endl; **//displays 4.**

**Try this, analyze the output:**

int main()

{

ofstream file;

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

int p=file.tellp();

cout<<" current position of put pointer is: "<<p<<endl; //displays 0.

file<<"C++ is better than C "; //writing some text to file.

cout<<"current position of put pointer is: "<<file.tellp()<<endl;

file.seekp(4); //moving put pointer to 4th position.

file<<"hello world "; //writing some text again.

cout<<" current position of pointer is: "<<file.tellp()<<endl;

file.close();

string text;

ifstream fin;

fin.open("test.txt", ios::in);

while(!fin.eof()) **//reading file until, end of file.**

{

fin>>text;

cout<<text;

}

fin.close();

return 0;

}

### Specifying the Reading and Writing Direction

The other form for seekg(), and seekp()functions:

seekg ( offset, direction );

seekp ( offset, direction );

**Direction can be:**

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

For Example:

FileObject.seekg(10,ios::beg): Go 10 characters forward from the start(for reading).

FileObject.seekg(-5,ios::end): Go 5 characters backward from the end(for reading).

FileObject.seekg(12,ios::cur): Go 12 characters forward from the current position(for reading).

FileObject.seekp(10,ios::beg): Go 10 characters forward from the start of the file(for writing).

FileObject.seekp(-5,ios::end): Go 5 characters backward from the end(for writing).

FileObject.seekp(-6,ios::cur): Go 6 characters backward from the current position (for writing).

## 6. Reading and Writing a Class Object (Object I/O)

The binary input and output functions read() and write() can be used for reading and writing a class object into the file. These functions handle the entire structure of an object as a single unit. But only data members can be written to the disk file, and member functions are not.

### Syntax of read(), and write() functions:

* *stream\_object\_name.read((char\*) &V, sizeof(V));* ***//to read from file***
* *stream\_object\_name.write(char\*) &V, sizeof(V));* ***//to write into file***

These functions take two arguments. The first is the address of the variable or object V, and the second is the length of that variable in bytes. The address of the variable must be cast to type char\*(i.e., pointer to character type.).

Following programs shows, how class objects can be written to/and read from the file. The length of the object is obtained by using the **sizeof()** operator.

### Example Program for Writing an Object to File:

#include <iostream>

#include <fstream>

using namespace std;

class Student

{

char name[20];

int roll ;

public:

void getdata( )

{

cout<< " Enter name and roll";

cin>>name>>roll ;

}

void display()

{

cout<< "Name\t"<< "Roll"<<endl;

cout<<" "<<name<<"\t"<<" "<<roll;

}

} ;

int main( )

{

Student X;

X.getdata( ) ;

fstream file;

file.open("data.txt",ios::binary|ios::out); **//opened in out mode so we can perform write operation only.**

file.write ((char\*) &X, sizeof (X));

file.close( ) ;

return 0 ;

}

**Example Program for Reading an Object from File:**

#include <iostream>

#include <fstream>

using namespace std;

class Student

{

char name[20];

int roll ;

public:

void getdata( )

{

cout<< " Enter name and roll";

cin>>name>>roll ;

}

void display( )

{

cout<<" Name"<<"\t"<<" Roll" <<endl;

cout<<" "<<name<<"\t"<<" "<<roll;

}

};

int main( )

{

Student Y;

fstream file;

file.open(" data.txt", ios::binary|ios::in|ios::trunc); **//remember fstream is used.**

file.read((char\*)&Y, sizeof (Y));

Y.display( ) ;

file.close( ) ;

return 0 ;

}

**Testing errors during file operations**

Sometimes during file operations, errors may also creep in. For example, a file being opened for reading might not exits or a file name used for a new file may already exist or any invalid operation may be performed. There might not be enough space in the disk for storing data. There are several error handling functions supported by class ios that help you read and process the status recorded in a file stream.

-eof(): Returns non zero if end of fie is encountered while reading , otherwise returns zero.

-fail(): Returns non zero when an input or output operation has failed.

-good(): Returns non zero if no error has occurred.

-bad(): Returns a non zero value if an invalid operation is attempted or any unrecoverable error occurred.

### More example programs

### Example of Simultaneous Reading and Writing:

#include <iostream>

#include <fstream>

using namespace std;

class Student

{

char fname[100];

char lname[100];

int roll ;

public:

void getdata( )

{

cout<< " Enter first\_name, last\_name and roll";

cin>>fname>>lname;

cin>>roll ;

}

void show( )

{

cout<<" "<<fname<<"\t"<<" "<<lname<<"\t"<<" "<<roll;

}

} ;

int main( )

{

Student X, Y;

X.getdata( ) ;

fstream file; //declaring fstream object for both reading and writing.

file.open("sample.txt", ios::binary|ios::in|ios::out|ios::trunc); //multiple modes.

file.write ((char\*) &X, sizeof (X)); //writing to the file.

file.seekg(0); //Reset to the start, for reading.

cout<<" FName "<<"\t"<<" Lname "<<"\t"<<" Roll " <<endl;

cout<<"----------------------------------------------------"<<endl;

while (file.read((char\*)&Y, sizeof(Y)) ) //reading one object at a time from file.

{

Y.show(); //displaying on the screen.

cout<<endl;

}

file.close();

return 0;

}

**Reading and Writing Multiple Objects:**

#include <iostream>

int main( )

{

Item i;

char ch;

fstream file ;

file.open (" item.dat", ios:: in|ios:: out|ios::app);

do

{

i.readdata( ) ;

file.write((char\*) &i, sizeof (i)) ;

cout<< " Enter another Item (y/n)?\n" ;

cin>>ch;

} while(ch=='y');

file.seekg(0) ;  **// reset to start.**

cout<< " \n Output: \n\n" ;

while(file.read((char\*) &i , sizeof (i)))

{

i.writedata( );

}

file.close( ) ;

return 0 ;

}

#include <fstream>

#include <iomanip>

using namespace std;

class Item

{

int code ; // item code

char name[100]; //item name

float cost ; // cost of each item.

public:

void readdata(void) ;

void writedata(void) ;

} ;

void Item:: readdata(void)  **// read from keyboard.**

{

cout<< " Enter code: "; cin>>code ;

cout<< " Enter name: "; cin>>name ;

cout<< " Enter cost: "; cin>>cost ;

}

void Item:: writedata(void) **// for display.**

{

cout<<setw(10)<<code<<setw(10)<<name

<<setw(10)<<cost

<<endl ;

}

#### 1. SEARCHING IN THE FILE

int main( )

{

Item i;

char ch;

fstream file ;

file.open (" item.dat", ios:: in|ios:: out|ios::app);

do

{

i.readdata( ) ;

file.write((char\*) &i, sizeof (i)) ;

cout<< " Enter another Item (y/n)?\n" ;

cin>>ch;

}while(ch=='y');

file.seekg(0) ; **// reset to start**

cout<< " \n Output: \n\n" ;

while(file.read((char\*) &i , sizeof (i)))

{

i.search();

}

file.close( ) ;

return 0 ;

}

**//Displaying Items having price less than 90.**

#include <iostream>

#include <fstream>

#include <iomanip>

using namespace std;

class Item

{

int code ; **// item code**

char name[100]; **//item name**

float cost ; **// cost of each item.**

public:

void readdata(void) ;

void writedata(void) ;

void search();

} ;

void Item:: readdata(void) **// read from keyboard**

{

cout<< " Enter code: "; cin>>code ;

cout<< " Enter name: "; cin>>name ;

cout<< " Enter cost: "; cin>>cost ;

}

void Item:: writedata(void) **// for display.**

{

cout<<setw(10)<<code<<setw(10)<<name

<<setw(10)<<cost

<<endl ;

}

void search() **//searching for items, having price less than 90.**

{

if(cost<90)

{

writedata();

}

}

#### 2. Displaying nth item

For this, first find the location of the first byte of nth object in the file.

**General formula for obtaining the desired object;**

*int location= (n-1)\*sizeof(object);*

Then set the file pointer to reach this byte with the help of seekg() or seekp().

**Example:**

using namespace std;

class Item

{

int code ; // item code

char name[10] ; //item name

float cost ; // cost of each item.

public:

void writedata(void) ;

};

void Item:: writedata(void) // formatted display on screen.

{

cout<<setw(10)<<code<<setw(10)<<name <<setw(10)<<cost <<endl ;

}

int main( )

{

Item i;

int n;

fstream file ;

file.open (" item.dat", ios:: in|ios:: out);

cout<<" Enter Item number to be displayed: ";

cin>>n;

int a=sizeof(i); **//finding the size of individual object.**

int pos=a\*(n-1**); //finding position of nth object.**

file.seekg(pos); **//moving file pointer to that position.**

file.read((char\*)&i, sizeof(i)); **//reading object which is at the position** pos**.**

i.writedata( ); **//displaying that item.**

file.close( ) ;

return 0 ;

}

### COMPLETE MENU PROGRAM:

**//Use this concept in your project, but do not copy the code.**

#include <iostream>

#include <fstream>

#include <iomanip>

#include <cstdlib>

using namespace std;

class Item

{

int code ; // item code

char name[10] ; //item name

float cost ; // cost of each item.

public:

void readdata(void) ;

void writedata(void) ;

int retCode()

{

return code;

}

} ;

void Item:: readdata(void) // read from keyboard.

{

cout<< " Enter code: "; cin>>code ;

cout<< " Enter name: "; cin>>name ;

cout<< " Enter cost: "; cin>>cost ;

}

void Item:: writedata(void) // for display.

{

cout<<code<<"\t"<<name<<"\t\t"<<cost<<endl ;

}

void Insert() **// function to write in a binary file.**

{

ofstream file;

file.open("Item.dat ", ios::binary | ios::app);

Item obj;

obj.readdata();

file.write((char\*)&obj, sizeof(obj));

file.close();

}

void Display() **// function to display records of file on screen, formatted output.**

{

ifstream file;

file.open("Item.dat ", ios::binary);

Item obj;

cout<<"Code\t"<<"Item Name\t"<<"Price"<<endl;

cout<<"..............................."<<endl;

while(file.read((char\*)&obj, sizeof(obj)))

{

obj.writedata();

}

file.close();

cout<<"..............................."<<endl;

cout<<"..............................."<<endl;

}

void Delete(int n) **// function to delete a record.**

{

inFile.close();

outFile.close();

remove("Item.dat "); **//deletes a file from memory.**

rename("temp.dat", "Item.dat");

**// renames the file temp.dat as Item.dat.**

}

Item obj;

ifstream inFile;

inFile.open("Item.dat ", ios::binary);

ofstream outFile;

outFile.open("temp.dat", ios::out | ios::binary);

while(inFile.read((char\*)&obj, sizeof(obj)))

{

if(obj.retCode()!= n)

{

outFile.write((char\*)&obj, sizeof(obj));

}

}

void Modify(int n) **// function to modify a record.**

{

fstream file;

file.open("Item.dat",ios::in | ios::out);

Item obj;

while(file.read((char\*)&obj, sizeof(obj)))

{

if(obj.retCode() == n)

{

cout << "\nEnter the new details of item";

obj.readdata();

int pos = sizeof(obj);

file.seekp(-pos, ios::cur); **//moving put pointer to the *pos* position for writing new detail.**

file.write((char\*)&obj, sizeof(obj));

}

}

file.close();

}

void Search(int n) **//function to search and display from binary file.**

{

ifstream inFile;

inFile.open("Item.dat ", ios::binary);

Item obj;

while(inFile.read((char\*)&obj, sizeof(obj)))

{

if(obj.retCode() == n)

{

obj.writedata();

}

}

inFile.close();

}

int main( )

{

int r,choice;

char ch;

do

{

cout<<"\*\*\*\*\*\*MENU\*\*\*\*\*\*\*\*\*\*\*"<<endl;

cout<<"1.INSERT"<<endl;

cout<<"2.DISPLAY"<<endl;

cout<<"3.MODIFY"<<endl;

cout<<"4.DELETE"<<endl;

cout<<"5.SEARCH"<<endl;

cout<<"6.EXIT"<<endl;

cout<<"Enter your choice: "<<endl;

cin>>choice;

switch(choice)

{

case 1:

Insert();

break;

case 2:

Display();

break;

case 3:

int c;

cout<<"Enter item code to modify(1-100): "<<endl;

cin>>c;

Modify(c);

break;

case 4:

int m;

cout<<"Enter item code to delete(1-100): "<<endl;

cin>>m;

Delete(m);

break;

case 5:

int s;

cout<<"Enter item code to search(1-100): "<<endl;

cin>>s;

Search(s);

break;

case 6:

exit(0);

}

} while(choice<7);

}