System Design and Programming II

CSCI – 1943

David L. Sylvester, Sr., Professor

Chapter 12

Advanced File Operation

File Operations

A file is a collection of data that is usually stored on a computer's disk.

File Stream	
Data Type	Description
ifstream	Input File Stream. This data type can be used only to read data from files into memory.
ofstream	Output File Stream. This data type can be used to create files and write data to them.
fstream	File Stream. This data type can be used to create files, write data to them, and read data from them.

Using the ***fstream*** Data Type

You define an fstream object just as you define objects of other data types.

Ex: fstream dataFile;

As with ifstream and ofstream objects, you use an fstream object's open function to open a file. An fstream object's open function requires two arguments. The first argument is a string containing the name of the file, and the second argument is a file access flag that indicates the mode in which you wish to open the file.

Ex: dataFile.open("info.txt", ios::out);

The first argument is the name of the file info.txt. The second argument is the file access flag ios::out. This tells C++ to open the file in output mode, which allows data to be written to the file.

Ex: dataFile.open("info.txt", ios::in);

This statement uses the ios::in access flag to open a file in input mode, which allows data to be read from the file.

Access Flags	
File Access Flags	Meaning
ios::app	Append mode. If the file already exists, its contents are preserved and all output is written to the end of the file. By default, this flag causes the file to be created if it does not exist.
ios::ate	If the file already exists, the program goes directly to the end of it. Output may be written anywhere in the file.
ios::binary	Binary mode. When a file is opened in binary mode, data are written to or read from it in pure binary format. (Default mode is text.)
ios::in	Input mode. Data will be read from the file. If the file does not exist, it will not be created and the open function will fail.
ios::out	Output mode. Data will be written to the file. By default, the file's content will be deleted if it already exists.
ios:: trunc	If the file already exists, its contents will be deleted (truncated). This is the default mode use by ios::out.

Several flags may be used together if they are connected with the | operator.

Ex: `dataFile.open("info.txt", ios::in |ios::out);`

This statement opens the file info.txt in both input and output mode. This means that data may be written to and read from the file.

Note: *When used by itself, the ios::out flag causes the file's contents to be deleted if the file already exists. When used with the ios::in flag, however, the file's existing contents are preserved. If the file does not already exist, it will be created.*

The following statement opens the file in such a way that data will only be written to its end.

```
dataFile.open("info.txt", ios::out | ios::app);
```

By using different combinations of access flags, you can open files in many possible modes.

Sample Program

```
•   // This program uses an fstream object to write data to a file.
•   #include <iostream>
•   #include <fstream>
•   using namespace std;

•   int main()
•   {
•      fstream dataFile;

•      cout << "Opening file...\n";
•      dataFile.open("demofile.txt", ios::out);   // Open for output
•      cout << "Now writing data to the file.\n";
•      dataFile << "Jones\n";                      // Write line 1
•      dataFile << "Smith\n";                      // Write line 2
•      dataFile << "Willis\n";                     // Write line 3
•      dataFile << "Davis\n";                      // Write line 4
•      dataFile.close();                           // Close the file
•      cout << "Done.\n";
•      return 0;
•   }
```

Causes each name to appear on a new line.

J	o	n	e	s	\n	S	m	i	t	h	\n	W	i	l
l	i	s	\n	D	a	v	i	s	\n	<EOF>				

The \n characters are written to the file along with all the other characters. The characters are added to the file sequentially, in the order they are written by the program. The very last character is an end-of-file marker. It is a character that marks the end of the file and is automatically written when the file is closed. The particular mark used depends on the operating system being used. Just like the new line character, the eof marker is a non-printable character also.

Sample Program

```
// This program writes data to a file, closes the file,
// then reopens the file and appends more data.
#include <iostream>
#include <fstream>
using namespace std;

int main()
{
   ofstream dataFile;

   cout << "Opening file...\n";
   // Open the file in output mode.
   dataFile.open("demofile.txt", ios::out);
   cout << "Now writing data to the file.\n";
   dataFile << "Jones\n";                    // Write line 1
   dataFile << "Smith\n";                    // Write line 2
   cout << "Now closing the file.\n";
   dataFile.close();                         // Close the file

   cout << "Opening the file again...\n";
// Open the file in append mode.
   dataFile.open("demofile.txt", ios::out | ios::app);
   cout << "Writing more data to the file.\n";
   dataFile << "Willis\n";                   // Write line 3
   dataFile << "Davis\n";                    // Write line 4
   cout << "Now closing the file.\n";
   dataFile.close();                         // Close the file

   cout << "Done.\n";
   return 0;
}
```

The first time file is opened, two names are written.

J	o	n	e	s	\n	S	m	i	t	h	\n	**<EOF>**

The file is closed and an end-of-file character is automatically written. When the file is reopened, the new output is appended to the end of the file.

J	o	n	e	s	\n	S	m	I	t	h	\n	W	i	l
l	i	s	\n	D	a	v	i	S	\n	<EOF>				

Note: if the ios::out flag had been alone, without ios::app the second time the file was opened, the file's contents would have been deleted and only Willis and Davis would be the contents of the file.

File Open Mode with *ifstream* and *ofstream* Objects

The ifstream and ofstream data types each have a default mode in which they open files. This mode determines the operations that may be performed on the file, and what happens if the file that is being open already exists.

File Type	Default Open Mode
ofstream	The file is opened for output only. Data may be written to the file, but not read from the file. If the file does not exist, it is created. If the file already exists, its contents are deleted.
ifstream	The file is opened for input only. Data may be read from the file but not written to it. The file's contents will be read from its beginning. If the file does not exist, the open function fails.

You cannot change the fact that ifstream may only read from and ofstream files may only be written to. But, you can vary the way operations are carried out on these files by providing a file access flag as an optional second argument to the open function.

Ex:

```
ofstream outputFile;
outputFile.open(“Values.txt”, ios::out | ios::app);
```

The ios::app flag specifies that data written to the Values.txt file should be appended to its existing contents.

Checking for a File’s Existence Before Opening It

Sometimes you may want to determine whether a file already exists before opening it for output. You can do this by first attempting to open the file for input. If the file does not exist, the open operation will fail. In that case, you can create the file by opening it for output.

Ex:

```
fstream dataFile;
dataFile.open(“Values.txt”, ios::in);
if (dataFile.fail())
{
     // The file does not exist, so create it.
     dataFile.open(“Values.txt”, ios::out);
     //
     // Continue to process the file
    //
}
else           // The file already exists.
{
     dataFile.close();
     cout << “The file Values.txt already exists.\n”;
}
```

Opening a File with the File Stream Object Definition Statement

An alternative to using the open member function is to use the file stream object definition statement to open the file.

Ex:
```
fstream dataFile(“names.txt”, ios::in | ios::out);
```

This statement defines an fstream object named dataFile and uses it to open the file names.txt. The file is opened in both input and output modes. This technique eliminates the need to call the open function when you know the name and access mode of the file at the time the object is defined.

Another example:
```
ifstream inputFile(“info.txt”);
ofstream outputFile(“addresses.txt”);
ofstream dataFile(“customers.txt’,
            ios::out  | ios::app);
```

To test for errors after you have opened a file with this technique, you can use the following:

```
ifstream inputFile(“SalesData.txt”)
if (!inputFile)
     cout << “Error opening SalesDate.txt”;
```

File Names and Extensions

Name & Extension	File Content	Name & Extension	File Content
myprog.bas	BASIC program	vacation.jpg	JPEG image file
menu.bat	Windows batch file	invent.obj	Object file
install.doc	Microsoft Word document	instructions.pdf	Adobe Portable Document Format File
crunch.exe	Executable file	prog1.prj	Borland C++ project file
bob.html	Hypertext Markup Language	ansi.sys	System device driver
3dmodel.java	Java program or applet	readme.txt	Text file

File Output Format

The setprecision and fixed manipulators may be called to establish the number of digits of precisions that floating point values are rounded to.

```
// This program uses the setprecision and fixed
// manipulators to format file output.
#include <iostream>
#include <iomanip>
#include <fstream>
using namespace std;

int main()
{
   fstream dataFile;
   double num = 17.816392;

   dataFile.open("numfile.txt", ios::out);
// Open in output mode

   dataFile << fixed;
 // Format for fixed-point notation

   dataFile << num << endl;
// Write the number

 dataFile << setprecision(4);
// Format for 4 decimal places
dataFile << num << endl;
// Write the number

   dataFile << setprecision(3);
// Format for 3 decimal places
   dataFile << num << endl;
// Write the number

   dataFile << setprecision(2);
// Format for 2 decimal places
   dataFile << num << endl;
// Write the number

   dataFile << setprecision(1);
// Format for 1 decimal place
   dataFile << num << endl;
// Write the number

   cout << "Done.\n";
   dataFile.close();
// Close the file
   return 0;
}
```

Using setw to Format Output

```
// This program writes three rows
// of numbers to a file.
#include <iostream>
#include <fstream>
#include <iomanip>
using namespace std;

int main()
{
   const int ROWS = 3;   // Rows to write
   const int COLS = 3;   // Columns to write
   int nums[ROWS][COLS] = { 2897, 5, 837,
                            34, 7, 1623,
                            390, 3456, 12 };
   fstream outFile("table.txt", ios::out);

   // Write the three rows of numbers with each
   // number in a field of 8 character spaces.
   for (int row = 0; row < ROWS; row++)
   {
      for (int col = 0; col < COLS; col++)
      {
         outFile << setw(8) << nums[row][col];
      }
      outFile << endl;
   }
   outFile.close();
   cout << "Done.\n";
   return 0;
}
```

Representation of Output

				2	8	9	7								5						8	3	7	\n
						3	4								7					1	6	2	3	\n
					3	9	0					3	4	5	6							1	2	\n