Chapter 15: File Handling

In this chapter we will be taking a look at sequential file handling of text files. It is important to note that any data you set when running your programs will be lost once the program exits. There are many times that you need to retain information between runs of your program. An example would be a high score table. If the high scores were lost each time the application shut down it wouldn’t very interesting for the player. This information will be stored in a file and retrieved upon loading.

To be able to use any of the file handling functionality built into C++ you must include fstream and iostream at the top of your code files. Iostream is required as fstream inherits from this.

#include <iostream>

#include <fstream>

All the below examples have these included.

When opening a file in C++ you must state which operations you wish to perform on the file. This is determined by the iosMode that is set in the constructor. We will be looking at the constructor shortly, but first we need to understand what iosModes are available to us. Take a look at Table 14.1: iosMode Values.

Default values can be used when using the various fstreams available, butmore often than not you will wish to set the iosMode yourself. An example would be if you needed to append data at the end of a file instead of overwriting the data within.

|  |  |
| --- | --- |
| **iosMode** | **Description** |
| ios::in | Open file for input. |
| ios:out | Open file for output. |
| ios::binary | Open file in binary mode. |
| ios::ate | Set the initial position of the file pointer to the end of the file. |
| ios::app | All operations are done at the end of the file, appending any data to the current contents of the file. |
| ios::trunc | If the file is opened for output and it already exists, the previous version is deleted and replaced with this new one. |

Table 14.1: iosMode Values

At times you will need to set multiple iosModes. To do this you use the ‘**|’** symbol. An example of opening a file for output and to overwrite an existing file would look like this:

ofstream outFile;

outFile.open( “Example.txt”, ios::out | ios::trunk );

**Opening a File**

To open a file you first need to determine whether the file is going to be read from or written to. Depending on this choice there are three data type options. See Table 14.2: fstream Flags.

|  |  |
| --- | --- |
| Class | Default iosMode |
| ifstream | ios::in |
| ofstream | ios::out |
| fstream | ios::in | ios::out |

Table 14.2: fstream Flags

As can be seen from the table, default values are set depending on the data type. These can be changed in the constructor.

To open a file for input use one of the following approaches:

ifstream inFile( “Example.txt”);

Overriding the iosMode:

ifstream inFile( “Example.txt”, ios::in | ios::beg );

To open a file for output use one of the following approaches:

Using default iosMode:

ofstream outFile( “Example.txt”);

Overriding the iosMode:

ofstream outFile( “Example.txt”, ios::out | ios::trunk );

To open a file for input and output use on of the following approach which uses the default iosMode:

fstream inoutFile( “Example.txt”);

You can also use the open() function, which accepts the name of the file you want to open along with the iosMode flags.

ifstream inFile;

inFile.open( “Example.txt”, ios::in );

**Closing a File**

To close a file it is simply a matter of calling the close() function.

inoutFile.close();

**Checking a file was opened**

Before attempting to read or write to a file it is important to ensure that the file was successfully opened. To do this use one of the following approaches:

ifstream inFile;

inFile.open( “Example.txt”, ios::in );

if( !inFile )

{

cerr << “File could not be opened!”;

std::exit( 1 );

}

Alternatively use the is\_open() function.

ifstream inFile( “Example.txt”);

if( !inFile.is\_open() )

{

cerr << “File could not be opened!”;

std::exit( 1 );

}

NOTE: To use the exit() function you need to include <cstdlib>.

**Writing to a File**

To write to a file you simply use the **>>** operator in the same way as you have done in previous example with cout.

int myVariable = 10;

outFile >>”File will hold this string and the number “ << myVariable << endl;

**Program 39: Writing to a Text File**

This code listing will demonstrate how to open a file for output and to write text to it. When you run the program it will quickly close. To check that it was successful look in you folder and open the newly created example.txt file.

1. To begin, start Visual Studio.
2. Create a new project via File -> New -> Project or Ctrl+Shift+N Name it “Chapter14\_WritingToAFile”
3. Click **Next** and you should be greeted with the following screen. Make sure to have **Empty Project** ticked and click **Finish**.
4. Add a new source file and name it “WritingToAFileExample.cpp”
5. Replicate program listing 39.

#include <cstdlib>

#include <iostream>

#include <fstream>

using namespace std;

int main ()

{

ofstream outfile ("example.txt");

// Check if the file was opened.

if ( ! outfile.is\_open() )

{

cerr << “File could not be opened!”;

std::exit( 1 );

}

// To reach here the file must be open.

// Write to the open file.

outfile << "This is a line.\n";

outfile << "This is another line.\n";

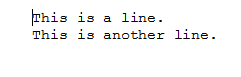
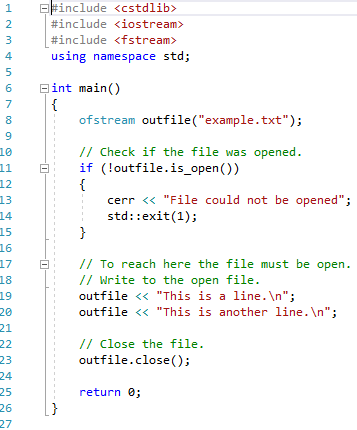
// Close the file.

outfile.close();

return 0;

}

Program Listing 39



**Reading from a File**

To read data from a file there are a couple of approaches. You can use the **<<** operators in a similar manner to the way you have used it with cin. Alternatively you can use the getline() function and pass in the file you wish to read from.

int myVariable;

inFile >> myVariable;

string lineOfText;

getline( inFile, lineOfText );

**Program 40: Reading from a Text File**

In this program we will be reading in the data that was saved in the previous program. If you haven’t yet completed Program 39: Writing to a Text File you need to go back and finish that before attempting this one. Copy across the example.txt file from the previous program. It needs to be placed in the project folder.

1. To begin, start Visual Studio.
2. Create a new project via File -> New -> Project or Ctrl+Shift+N Name it “Chapter14\_ReadingFromAFile”
3. Click **Next** and you should be greeted with the following screen. Make sure to have **Empty Project** ticked and click **Finish**.
4. Add a new source file and name it “ReadingFromAFileExample.cpp”
5. Replicate program listing 40.

#include <cstdlib>

#include <iostream>

#include <fstream>

#include <string>

using namespace std;

int main ()

{

string lineOfText;

ifstream infile ("example.txt");

// Check if the file was opened.

if ( infile.is\_open() )

{

// Loop through the entire file and output each line to the console.

while ( getline (infile, lineOfText) )

{

cout << lineOfText << endl;

}

// Close the file.

infile.close();

}

else

{

// File did not open. Inform the user.

cout << "Unable to open file";

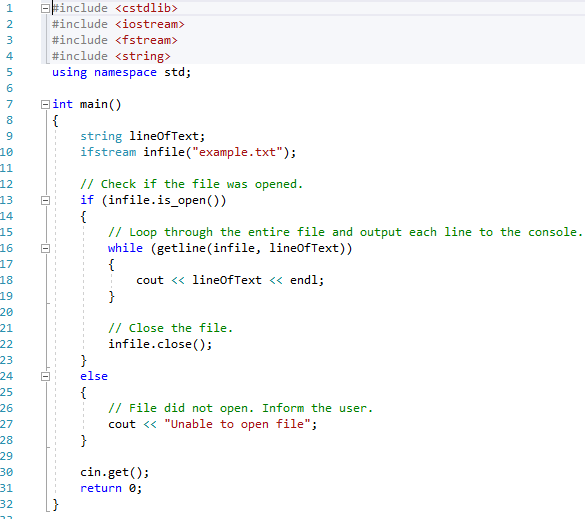
}

cin.get();

return 0;

}

Program Listing 40





**Moving the File Pointer**

When first opening a file the file pointer will be positioned at the beginning of the file. At times you will need to move this pointer. Functions have been provided which enable this. seekg() is used with ifstream and seekp() for ofstream. You need to pass in to the function the number of bytes you wish to offset from the beginning of the file.

To seek n bytes from the beginning of the file:

inFile.seekg( *n )*;

To seek n bytes from the current position of the file:

inFile.seekg( *n*, ios::cur *)*;

To seek back n bytes from the end of the file:

inFile.seekg( *n*, ios::end *)*;

To seek to the end of the file:

inFile.seekg( o, ios::end *)*;

When using the tellg() or tellp() functions you will notice that they return a new data type. This is the streampos type. This type can hold the internal position of the file pointer.

To seek to the end of a file and return the streampos:

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

streampos currentPosition = inFile.tellg();

**Program 41: File Size**

This code listing will demonstrate how you can use the tellg() function to determine the size of the file. Copy across the example.txt file from the previous program. It needs to be placed in the project folder.

1. To begin, start Visual Studio.
2. Create a new project via File -> New -> Project or Ctrl+Shift+N Name it “Chapter14\_FilePointer”
3. Click **Next** and you should be greeted with the following screen. Make sure to have **Empty Project** ticked and click **Finish**.
4. Add a new source file and name it “FilePointerExample.cpp”
5. Replicate program listing 41.

#include <iostream>

#include <fstream>

#include <cstdlib>

using namespace std;

int main ()

{

streampos begin,end;

// Open the file.

ifstream inFile ("example.txt", ios::binary);

if( !inFile )

{

cerr << “File could not be opened!”;

std::exit( 1 );

}

// Get the current position in the file.

begin = inFile.tellg();

// Reposition the file pointer to the end of the file.

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

// Get the current position in the file.

end = inFile.tellg();

// Close the file.

inFile.close();

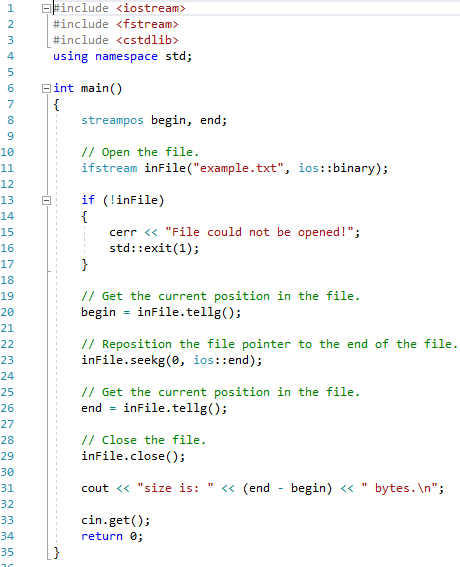
cout << "size is: " << (end-begin) << " bytes.\n";

cin.get();

return 0;

}

Program Listing 41





**Program 42: Counting Characters**

This code listing will read and write files using the different streams available. It will count the number of integers in the file and the number of non-blank characters. Ensure that you include a screenshot of the contents of the text file.

1. To begin, start Visual Studio.
2. Create a new project via File -> New -> Project or Ctrl+Shift+N Name it “Chapter14\_CountingCharacters”
3. Click **Next** and you should be greeted with the following screen. Make sure to have **Empty Project** ticked and click **Finish**.
4. Add a new source file and name it “CountingCharactersExample.cpp”
5. Replicate program listing 42.

#include <iostream>

#include <fstream>

using namespace std;

int main()

{

char character;

int number = 51;

int count = 0;

ofstream out\_stream;

ifstream in\_stream1;

ifstream in\_stream2;

// Create the file

out\_stream.open("Integers");

for (count = 1 ; count <= 5 ; count++)

{

out\_stream << number++ << “ “;

}

out\_stream.close();

// Count the integers in the file

in\_stream1.open("Integers");

count = 0;

in\_stream1 >> number;

while (!in\_stream1.eof())

{

count++;

in\_stream1 >> number;

}

in\_stream1.close();

cout << "There are " << count << " integers in the file,\n";

// Count the non-blank characters

in\_stream2.open("Integers");

count = 0;

in\_stream2 >> character;

while ( !in\_stream2.eof() )

{

count++;

in\_stream2 >> character;

}

in\_stream2.close();

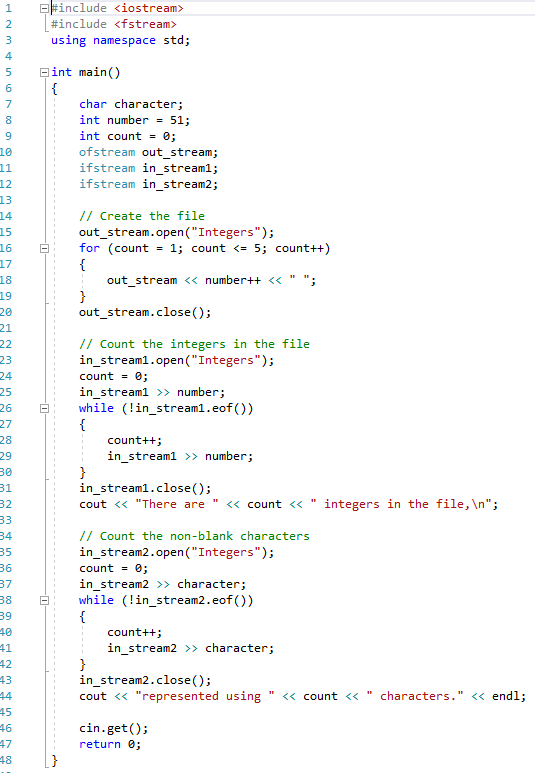
cout << "represented using " << count << " characters." << endl;

cin.get();

return 0;

}

Program Listing 42





**Program 43: Top Ten Scores**

This program is to test not only your file handling ability but also how you choose to handle the data internally. Only 10 scores and names should ever be stored. A screenshot of the scores text file should also be included below. It MUST match the output expected from the screenshot of the program running.

Write a program that presents the user with the following option screen:

1. Enter a score
2. Display scores
3. Exit

If the user enters an invalid option the program should inform the user and then re-present the menu screen.

Upon entering ‘1’ the user should be asked for a score and a name. If the entered score is greater than the lowest score already in the file, or there are less than 10 scores stored this new score should be incorporated in to the top ten. This should then be saved in a text file called ‘scores.txt’. The user is then returned to the menus screen.

Upon entering ‘2’ the program should output a well-formatted list of scores. This should stay on screen until the user presses the return key. If there are no scores the user should be informed and then returned to the menu screen.

Upon entering ‘3’ the program should close. All file streams should be closed correctly.

NOTE: This program must not crash. It is up to you as programmer to ensure invalid options are dealt with and that if the file doesn’t exist and the user tries to access it that they are informed and the program returns to the menu.

