

## Writing and Reading Text Files in C++

### Introduction to File I/O

The objective for this guided laboratory assignment is to learn about the basics for writing and reading text files in C++. Up to this point in class, all of the programs you have written typically require the user of your program to enter data each time they use the program. Now, wouldn't it be nice if we could write programs that have data entered as input through a simple text file?

In this lab, we will look at how we can have our programs write/read data. When we want to use a file in a program, we follow three very fundamental steps. These three steps are:

- **Opening the file**: When we open a file, we are creating a connection between the file and our program. The process of opening a file for output typically creates the file on the disk and allows the program to write data to it (there are ways for us to open a file for output and append new data to the end of the file - more on this later). The process of opening a file for input allows the program to read data from the file (a nice idea so we do not have to keep entering data when testing our programs).
- **Processing the file**: The data is either written to the file (if the file is an output file) or read from the file (if the file is an input file).
- **Closing the file**: After our program is finished using the file, we must have the file be closed. The idea of closing a file disconnects the file from the program (this also allows the operating system to unlock our file).

There are two types of files we can write and read in C++, text files and binary files. A text file contains data that has been written as text, using a scheme such as ASCII or Unicode (this is the easiest form of file to read the contents of the file). Even if we have the file contain numbers (integers and floating point), those numbers have their digits stored in the file as a series of characters that we can easily read. As a result, the file may be opened and viewed in any text editor such as Notepad++.

Now, a binary file contains data that has not been converted to text (this is data in its natural format, depending on its data type). With binary files, you cannot view the contents of the binary file with a text editor (you can try, but more than likely, you'll see a lot of

strange looking characters). For now, we work only with text files as they are easier to learn. After completing Chapter 9 of our text, we will look at parts of Chapter 12 where you will learn to work with binary files.

Okay, so now that the basic fundamentals about files have been introduced, let's see what we need in a C++ program to work with files (the three steps above work for both text files and binary files).

### **A New Include Directive**

To work with files (both text and binary), we need to have the following in our program's list of #include directives:

```
#include <fstream>
```

The fstream library has several classes that we will want to use for doing file input/output. The first class we use is called: ofstream, which stands for output file stream. We use ofstream if we want to have our program output (write) data to the file. The second class we is called: ifstream, which stands for input file stream. We use ifstream if we want to have our program input (read) data from a file. There is a third class, which is a combination of both ofstream and ifstream, called fstream. It is a little bit different to use than ofstream and ifstream, because when we open our file, we have to specify to the program how we are opening our file (we have to tell it if the file is being opened for reading or writing, or both). To keep things simple, I recommend working with ofstream and ifstream until you have had some good practice with writing programs doing file I/O.

### **Program #1 - Integer Writer**

Let's write a program that will ask our user for the following:

1. The name of the file our user wants to create.
2. Create the output file with user supplied file name.
3. Ask the user to enter an integer number that is not -999999 (-999999 is a special sentinel [flag] value we will use to signal the termination of entering as many integers as they want).
4. If the input value is not -999999, write the number out to the output file.
5. Loop back to step #3 (if you notice, this is a do/while loop).
6. Close the file and exit program.

I would like for you to write this program (I have provided screenshots of the code – along with their line numbers for reference for certain things that should be noted).

```
1 // All necessary include directives here
2 #include <iostream>
3 #include <fstream>
4 #include <string>
5 #include <iomanip>
6 using namespace std;
7
```

In our program, let's have the following include directives and using the standard library namespace (lines 2-6).

```
8 // main function here
9 int main()
10 {
11     string filename;
12
13     // prompt user for filename
14     cout << "Enter name of file you want to write: ";
15     getline(cin, filename, '\n');
16
17     // declare an ofstream object for file output
18     ofstream outputFile;
19
20     // try to open the file
21     outputFile.open(filename.c_str());
```

Our main function starts at about line 9. Everything looks pretty basic except for some new items on lines 15, 18, and 21. In line #15, we use a function called `getline` (it is a part of `iostream`). The reason we use this `getline` function instead of `cin` (which we are used to using) is because if we want to have the user enter a text string that has whitespaces (spaces, tabs, carriage return), `cin` will treat the whitespaces as a delimiter of the text string. To use the `getline` function, we call it in the following manner:

```
getline(cin, your_string_variable, delimiter_character);
```

Where `cin` is our `cin` object we have been using in class, `your_string_variable` is the variable of the type string that you want to store the input in, and `delimiter_character` is the character literal that you want to use to signal the end of the input string (typically we will use `'\n'` for new line).

On line #18, we have our program create an output file stream object called `outputFile` (like any other variable or object, you can name it whatever you want – I encourage naming it something that you and other programmers should easily recognize as the variable's purpose).

On line #21, we try to open a file with the name the user typed in on line #15. Note: the string class has a special function called

c\_str(), what that does is converts the string object (in this case, named filename) to another data type called c-string (this is found in Chapter 10 of our text). The outputFile object has a function called open. We use the open function to try opening a file for output with the file name in the given string (one scenario in which this opening the file for output can fail is if the file already exists and is opened by another program, or another scenario is that the operating system has the file flagged as a read-only file).

```
22
23     if (outputFile)
24     {
25         int input;
26         do
27         {
28             // prompt user for integer to input, -999999 stops the loop and closes the file
29             cout << "Enter an integer (-999999 to stop): ";
30             cin >> input;
31
32             if ( input == -999999 ) break;
33
34             outputFile << setw(8) << input << "\r\n";
35             outputFile.flush();
36
37         } while(input != -999999);
38     }
39     else
40     {
41         cout << "File could not be opened for writing..." << endl;
42         return 1;
43     }
```

Line #23 is an interesting line with the if statement and the output file stream object named outputFile. If the open function on line #21 is successful in opening the file, the if statement in line #23 will branch as true. Lines #26 through #37 is our do/while loop that repeats until the user has enter the number:

-999999

To stop the loop.

Question: Where is writing the user entered input to the output file happening?

Answer: On line #34. Notice how the outputFile object uses << just like cout does? The << (insertion operator) takes data and inserts it into a stream (in this case the output file stream - cout does the same thing but the output stream is to the console screen [you could think of the console screen as a "file"]). Line #35 has an interesting function called flush(). The job for the flush() function is to take all of the data sitting in the stream and write it out to the file all in one chunk of data.

The else block (lines 39 through 43) is executed if for some reason the output file could not be created for writing data (one scenario in which this opening the file for output can fail is if the file already exists and is opened by another program, or another scenario is that the operating system has the file flagged as a read-only file). Note how I had the return code be 1. I decided that a return code of 1 in this program would signal that an error has occurred (you can have your error code be whatever you desire – just make sure to document it in case your user needs to know).

```
44
45     outputFile.close();
46     cout << "File named: " << filename << " written." << endl;
47
48     // your main program goes here
49     return 0; // this line is optional
50 }
```

Whew! We are almost done with this program. Line #45 closes our output file stream object. Report to our user on line #46 a nice message. And of course, line #49, exit the program with an error code of 0 (I like using 0 for “all went good – no errors”).

### **Program #2 – Average of Integers File Reader**

I would like for you to write this program (I have provided screenshots of the code – along with their line numbers for reference for certain things that should be noted).

```
1 // All necessary include directives here
2 #include <iostream>
3 #include <fstream>
4 #include <iomanip>
5 #include <string>
6 using namespace std;
7
```

In our program, let’s have the following include directives and using the standard library namespace (lines 2-6).

```
8 // main function here
9 int main()
10 {
11     string filename;
12
13     // prompt user for filename
14     cout << "Enter name of file you want to read: ";
15     //cin >> filename;
16     getline(cin,filename,'\n');
17
```

Our main function starts at about line 9. Everything looks pretty much starting off like the first program in this lab. In line #16, we use the function called getline (it is a part of iostream) again for

reading in a string for the file name which could possibly include spaces in the name. If you understand getline from the first program in this lab, you can skip the following paragraph until the next screenshot.

The reason we use this getline function instead of cin (which we are used to using) is because if we want to have the user enter a text string that has whitespaces (spaces, tabs, carriage return), cin will treat the whitespaces as a delimiter of the text string. To use the getline function, we call it in the following manner:

```
getline(cin, your_string_variable, delimiter_character);
```

Where cin is our cin object we have been using in class, your string variable is the variable of the type string that you want to store the input in, and delimiter character is the character literal that you want to use to signal the end of the input string (typically we will use '\n' for new line).

```
18 // declare an ifstream object for file input
19 ifstream inputFile;
20
21 // try to open the file
22 inputFile.open(filename.c_str());
23
```

On line #19, we have our program create an input file stream object called inputFile (like any other variable or object, you can name it whatever you want – I encourage naming it something that you and other programmers should easily recognize as the variable's purpose).

On line #22, we try to open a file with the name the user typed in on line #16. If you understand the open function from the previous program in this lab, you can skip the following paragraph.

The string class has a special function called c\_str(), what that does is converts the string object (in this case, named filename) to another data type called c-string (this is found in Chapter 10 of our text). The inputFile object has a function called open. We use the open function to try opening a file for input with the file name in the given string (one scenario in which this opening the file for input can fail is if the file does not exist, or another scenario is that the file name [or even the file path] could be incorrect).

```

24     if (inputFile)
25     {
26         int input, sum=0, count=0;
27         // read in all integer values in the file until -999999 encountered
28         do
29         {
30             if( (inputFile >> input) )
31             {
32                 cout << "Found the value: " << input << endl;
33
34                 if (input == -999999) break;
35                 sum=sum+input;
36                 count++;
37             }
38             else break;
39
40         } while( input != -999999 );
41
42         // close the file when we are done
43         inputFile.close();
44
45         // output the average
46         cout << "Average: " << setw(10) << fixed << setprecision(5);
47         cout << sum/(float)count << endl;
48
49     }

```

Line #24 is an interesting line with the if statement and the input file stream object named inputFile. If the open function on line #22 is successful in opening the file, the if statement in line #24 will branch as true. Lines #26 through #40 is our do/while loop that repeats until the program has either encountered the number:

-999999

To stop the loop or the end of file (no more data to read).

Question: Where is reading of data from the input file happening?

Answer: On line #30. Notice how the inputFile object uses >> just like cin does? The >> (extraction operator) takes data and extracts it from a stream (in this case the input file stream - cin does the same thing but the input stream is to the console keyboard [you could think of the console keyboard as a "file"]).

```

50     else
51     {
52         cout << "File could not be opened for reading..." << endl;
53         return 1;
54     }
55     return 0; // this line is optional
56 }

```

The else block (lines 50 through 54) is executed if for some reason the input file could not be opened for reading data (one scenario in which this opening the file for input can fail is if the file does not

exist, or another scenario is that the file name [or even the file path] could be incorrect).

Note how I had the return code be 1. I decided that a return code of 1 in this program would signal that an error has occurred (you can have your error code be whatever you desire – just make sure to document it in case your user needs to know).

### **Closing Thoughts**

Modify both programs to work with floating point numbers (floats or doubles).