File I/O

Lecture 13

Outline

1. I/O Streams
2. File I/O via `fstream`
3. File Names as Inputs
4. Streams as Arguments
5. EOF
6. Output formatting

I/O

- I/O stands for Input/Output
- So far, we've used **`cin`** for all input (from the user's keyboard) and **`cout`** for all output (to the user's screen)
- **`cin`** and **`cout`** are predefined I/O *streams* that are available if you include the iostream library

Streams

- A stream is sequence of data (numbers, characters, strings, …)
- Input Streams are used to get data into your program
  - **`cin`** is an input stream, getting data from the keyboard into your program
- Output Streams are used to get data out of your program
  - **`cout`** is an output stream, getting data from your program to the screen

File I/O

- Streams can also be used to get data from a file and put data into a file
- Files are used to store data that needs to be available after the program ends
- Files are also used to store large data sets that might be input for a program, to save the need to type all the data values individually

`fstream` Library

- When using file I/O, you must include the **`fstream`** library

- **`fstream`** defines two new variable types
  - **`ifstream`**: input file stream
  - **`ofstream`**: output file stream

- Use these variable types to declare variables that represent files
  - `ifstream input_file; // input_file is a variable of type ifstream`
  - `ofstream output_file; // output_file is a variable of type ofstream`

`fstream` Functions

- There are many functions in the `fstream` library that work with **`ifstream`** and **`ofstream`** variables
- Two of the most important are **`open()`** and **`close()`**
- The way these functions are used is a bit different from what we've seen before
  - **`ifstream`** and **`ofstream`** are actually C++ *classes* which are more complex than standard variable types like int or double (like string!)

fstream open()

- **open()** is used to connect a file to the stream
- A stream can only be connected to one file at a time, and it must be connected to a file before you can use the stream for input or output
- Example to open the file named **test.txt** for writing (output):

```
ofstream output_file;
output_file.open( "test.txt” );
```

`fstream close()`

- `close()` is used to disconnect the file from the stream
- You should always close files when you are finished, otherwise they could be left in a corrupted state after your program ends
- Example:

```
output_file.close();
```

Reading from and Writing to Streams

- Once the stream is connected to a file (the file is opened), you read or write to the file just like with **`cin`** or **`cout`**, with the **`<<`** and **`>>`** operators
- We normally use **`cout`** with **`<<`** to output values from our programs to the screen
- We can now use **`ofstream`** variables and **`<<`** to output values from our programs to a file on your hard drive
- Likewise, we can use **`ifstream`** variables with **`>>`** to get input from a file into our programs

Writing Example

```
#include <fstream>
using namespace std;

int main()
{
    ofstream output_file;

    output_file.open( "hello.txt” );

    output_file << "Hello world!" << endl;

    output_file.close();

    return 0;
}
```

File Location

- Any files you create are put by default in the project directory under the directory with the same name as the project
- That is, in the same directory as the C++ source file
- You can open them with any editor you like, or in Visual Studio directly via the File->Open->File menu

Reading Example

```
#include <fstream>
#include <iostream>
#include <string>
using namespace std;

int main()
{
    string input;
    ifstream input_file;

    input_file.open( "hello.txt” );

    input_file >> input;
    cout << "Read: " << input << endl;

    input_file.close();

    return 0;
}
```

Exercise

Write a program that writes the numbers from 1 to 100 to a file named "numbers.txt”

Answer

```
#include <fstream>
using namespace std;

int main()
{
    int i;
    ofstream output_file;

    output_file.open( "numbers.txt" );

    for ( i=1; i<=100; i++ )
    {
        output_file << i << endl;
    }

    output_file.close();

    return 0;
}
```

Reading and Writing

```
#include <iostream>
#include <fstream>
using namespace std;

int main()
{
	ifstream inf;
	ofstream outf;
	inf.open( "input.txt” );
	outf.open( "output.txt” );

	for ( int i=1; i<=10; i++ )
	{
		int value;
		inf >> value;
		cout << "Read " << value << endl;
		outf << value*value << endl;
	}

	inf.close();
	outf.close();
	return 0;
}
```

`fstream fail()`

- When you open a file, you have to check that it actually opened successfully
- For **`ifstream`**, if the file does not exist or if it is not readable then it will not open correctly
- For **`ofstream`**, if the file is not writeable then it will not open correctly (it will automatically create a file with the name if one does not exist)
- Use the **`fail()`** function to check if it was successful or not

Example

```
#include <iostream>
#include <fstream>
using namespace std;

int main()
{
	ifstream inf;
	ofstream outf;
	inf.open( "input.txt” );
	if ( inf.fail() )
	{
		cout << "Input file failed to open!\n";
		return 1;
	}
	outf.open( "output.txt” );
	if( outf.fail() )
	{
		cout << "Output file failed to open!\n";
		return 1;
	}
	for ( int i=1; i<=10; i++ )
	{
		int value;
		inf >> value;
		cout << "Read " << value << endl;
		outf << value*value << endl;
	}

	inf.close();
	outf.close();
	return 0;
}
```

`inf.fail()` will return `true` if the file did not open correctly, and `false` if it did

Basic File I/O Summary

- You can read from and write to files just like getting input and output with **`cin`** and **`cout`**
- Use the **`fstream`** library to get access to **`ifstream`** and **`ofstream`** variable types
- Use **`<<`** and **`>>`** as with **`cin`** and **`cout`**
- Use **`open()`** to open files, and **`close()`** to close files
- Use **`fail()`** to check if files opened correctly

File I/O Exercise

Write a program that gets 8 integers from the user and writes them to a file named "8ints.txt”

Answer

```
#include <iostream>
#include <fstream>
using namespace std;

int main()
{
      ofstream outf;
      outf.open( "8ints.txt” );
      if ( outf.fail() )
      {
            cout << "Output file failed to open!" << endl;
            return 1;
      }
      cout << "Please enter 8 integers:\n";
      for ( int i=1; i<=8; i++ )
      {
            int value;
            cin >> value;
            outf << value << endl;
      }
      outf.close();
      return 0;
}
```

Appending to a File

- By default, when you open a file for an ofstream, it overwrites any data that was in the file to begin with
- You can also append to the file, which means that anything you write to the file will start after whatever data was there previously
- Use a special second argument (**`ios::app`**) to the **`open()`** function:

```
ofstream outf;
outf.open( "out.txt”, ios::app );
```

Example

```
#include <iostream>
#include <fstream>
using namespace std;

int main()
{
      ofstream outf;
      outf.open( "8ints.txt", ios::app );
      if( outf.fail() )
      {
            cout << "Output file failed to open!" << endl;
            return 1;
      }
      cout << "Please enter 8 integers:\n";
      for( int i=1; i<=8; i++ )
      {
            int value;
            cin >> value;
            outf << value << endl;
      }
      outf.close();
      return 0;
}
```

Only difference is the additional argument to **open**

File Names as Input

- The **`fstream open()`** function takes a string argument, which is the file name to open
- The argument does not have to be hard coded (like **`"test.txt"`**)
- It can be a **`string`** variable, which can be read from the user via **`cin`** (or another file, or anywhere else)

Example

```
#include <iostream>
#include <fstream>
#include <string>
using namespace std;

int main()
{
      ofstream outf;
      string filename;

      cout << "Enter the file name to write to: ";
      cin >> filename;

      outf.open( filename );
      if( outf.fail() )
      {
            cout << "Output file " << filename << " failed to open!" << endl;
            return 1;
      }
      for ( int i=1; i<=1000; i++ )
      {
            outf << i*i*i << endl;
      }
      outf.close();
      return 0;
}
```

Streams as Function Arguments

- Stream variables can be used as function arguments, just like any other variable
- Stream arguments must always be call by reference arguments
  - If you use them as call by value arguments you will get very strange build errors pointing you in to the `fstream` library code

Example

```
#include <iostream>
#include <fstream>
using namespace std;

void bottles(ofstream& ofs);

int main()
{
	ofstream output;
	output.open( "test.txt” );
	if( output.fail() )
	{
		cout << "Output file test.txt failed to open!" << endl;
		return 1;
	}
	bottles( output );
	output.close();
	return 0;
}

void bottles(ofstream& ofs)
{
	for ( int i=99; i>0; i-- )
	{
		ofs << i << " bottles of beer on the wall..." << endl;
	}
}
```

Checking for EOF

- When reading data from a file, you often want to read all of the data
- In other words, keep reading data until you get to the end of the file (EOF)
- C++ provides a simple mechanism to do this easily, but it looks a bit odd at first
- Relies on the `>>` operator to tell the program when there is no more input

Example

```
#include <iostream>
#include <fstream>
using namespace std;

int main()
{
	ifstream input;
	input.open( "numbers.txt” );
	if( input.fail() )
	{
		cout << "Input file numbers.txt failed to open!" << endl;
		return 1;
	}

	int val;
	while ( input >> val )
	{
		cout << "Read " << val << endl;
	}

	input.close();
	return 0;
}
```

Additional Stream Functions

- **`precision()`** is used to set the number of significant digits displayed when a double value is output
- **`width()`** is used to set the number of minimum number of spaces to output when a value is output (only applies to the next item that is output)
- **`setf()`** is used to set flags for the stream that affect the output behavior

`setf` Flags

`ios::fixed`	Double values should NOT use scientific notation.
`ios::scientific`	Double values should use scientific notation.
`ios::showpoint`	The decimal point and trailing zeros should be shown for floating point numbers.
`ios::showpos`	The plus sign should be output before positive integers.
`ios::right`	Right-align output values (usually use **`width`** with this, which is the default).
`ios::left`	Left-align output values (usually use **`width`** with this).

Example

```
#include <iostream>
#include <fstream>
using namespace std;

int main()
{
	ofstream output;
	output.open( "numbers.txt” );
	if( output.fail() )
	{
		cout << "Output file numbers.txt failed to open!" << endl;
		return 1;
	}
	output.setf( ios::left );
	for( int i=1; i<=110; i++ )
	{
		output.width( 3 );
		output << i;
		if( i % 10 == 0 )
		{
			output << endl;
		}
	}
	output.close();
	return 0;
}
```

Wrap Up

- You can append to an **`ofstream`** file by opening it with a second argument of **`ios::app`**
- File names are just strings, and can be treated as such (for example, by reading the file name from the user into a **`string`**)
- Output can be formatting by using the **`precision()`**, **`width()`**, and **`setf()`** functions on a stream
- **`ifstream`** and **`ofstream`** variables can be used as function arguments, but they must call by reference
- You can build the **`>>`** operator into loop calls to continue reading values until the file ends