Simple File I/O

from Chapter 6

http://www.cplusplus.com/reference/fstream/ifstream/
http://www.cplusplus.com/reference/fstream/ofstream/

I/O Streams

- I/O refers to a program's input and output
  - Input is delivered to a program via a stream
    - Input can be from: the keyboard or a file
  - Output is delivered to an output device via a stream
    - Output can be: the screen, or a file

Streams and Basic File I/O

- A stream is a flow of data.
  - Input stream: data flows into the program
    - If input stream flows from keyboard, the program will accept data from the keyboard
    - If input stream flows from a file, the program will accept data from the file
  - Output stream: data flows out of the program
    - To the screen
    - To a file

cin And cout Streams

- `cin`

  input stream connected to the keyboard
- `cout`

  output stream connected to the screen
- `cin` and `cout` defined in the `iostream` library
  - Use include directive: `#include <iostream>`

We can declare our own streams to use with files.

Why Use Files?

- Files allow you to store data permanently!
- Data output to a file lasts after the program ends
- A file can be used for input over and over
  - No typing of data again and again for testing
- Create a data file or write into an output file at your convenience
- Files allow you to deal with larger data sets

File I/O

- Reading from a file
  - Taking input from a file
  - From beginning to the end (for now)
    - Just as done from the keyboard
- Writing to a file
  - Sending output to a file
  - From beginning to end (for now)
    - Just as done to the screen

Stream Variables or Objects

- We use a stream variable or object for file I/O
  - It must be declared before it can be used
  - It must be initialized before it contains valid data
    - **Initializing a stream means connecting it to a file**
    - The value of the stream variable can be thought of as the file it is connected to
  - Can have its value changed
    - Changing a stream value means disconnecting from one file and connecting to another

Declaring An Input-file Stream Variable

- Input-file streams are of type `ifstream`
  - Type `ifstream` is defined in the `fstream` library
- You must use the include and using directives

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

- Declare an input-file stream variable or object

```
ifstream    in_stream;
```

Declaring An Output-file Stream Variable

- Ouput-file streams of are type `ofstream`
- Type `ofstream` is defined in the `fstream` library
  - You must use these include and using directives

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

- Declare an output-file stream variable using

```
ofstream    out_stream;
```

Connecting To A File

- Once a stream variable or object is declared, we can connect it to a file
  - Connecting a stream to a file is opening the file
  - Use the open function of the stream object

```
ifstream  in_stream;
in_stream.open("infile.dat");
```

File name on the disk

External File Names

- An External File Name…
  - Is the "real", on-the-disk, name for a file
  - Is the name for a file that the operating system uses
    - `infile.dat` used in the previous example
  - Needs to match the naming conventions on your system
  - Usually only used in a stream's `open` statement
  - Once opened (i.e., connected to a stream object), it is referred to using the name of the stream connected to it.

Using The Input Stream

- Once connected to a file, the input-stream variable can be used to produce input just as you would use `cin` with the extraction operator
  - Example:

```
ifstream  in_stream;
in_stream.open("infile.dat");
int one_number, another_number;
in_stream >> one_number
          >> another_number;
```

- Intro to objects and classes

Objects

- An object is a variable that has member functions as well as the ability to hold data values
  - inStream and outStream each have a function named **open** associated with them
    - inStream and outStream use different versions of a function named **open**
    - One version of open is for input files
    - A different version of open is for output files
  - string str; str is an object
    - str.length() calling member function length() on str obj…
- **Class**: a data type whose variables are objects
  - so far, we have seen classes: string, ifstream, …
- Class vs. native data type

Member Functions

- A member function is a function associated with an object
  - The open function is a member function of inStream in the previous examples
  - A different open function is a member function of outStream in the previous examples

Objects and Member Function Names

- Objects of different types have different member functions
  - Some of these member functions might have the same name

- Different objects of the same type have the same member functions
  - string str1, str2;
  - …
  - http://www.cplusplus.com/reference/string/string/

Classes

- A type whose variables are objects, is a class
  - ifstream is the type of inStream variable (object)
  - ifstream is a class
  - The class (type) of an object determines its member functions
  - Example:

    ifstream inStream1, inStream2;

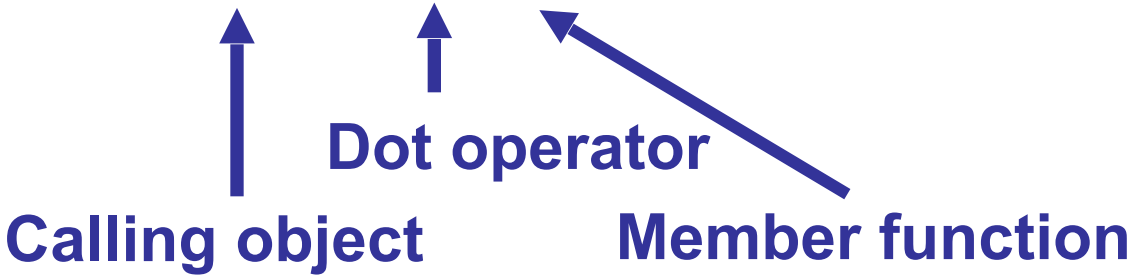
    - inStream1.open and inStream2.open are the same function but might have different arguments

Class Member Functions

- Member functions of an object are the member functions of its class
- The class determines the member functions of the object
  - The class ifstream has an open function
  - Every variable (object) declared of type ifstream has that open function

Calling a Member Function

- Calling a member function requires specifying *on which object the function is being called*
- The calling object is separated from the member function by the dot operator
- Example: inStream.open("infile.dat");



- e.g, string str="Hello! World";
- to find the length of str: str.length();

Member Function Calling Syntax

- Syntax for calling a member function:

  Calling_object.Member_Function_Name(Argument_list);

Using The Output Stream

- An output-stream works similarly to the input-stream

```
ofstream out_stream;
out_stream.open("outfile.dat");

out_stream << "one number = "
           << one_number
           << "another number = "
           << another_number;
```

Closing a File

- After using a file, it should be closed
  - This disconnects the stream from the file
  - Close files to reduce the chance of a file being corrupted if the program terminates abnormally
- It is important to close an output file if your program later needs to read input from the output file
  - The system will automatically close files if you forget as long as your program ends normally

Good practice: always close a file immediately after your code has used it.

Display 6.1

Errors On Opening Files

- Opening a file could fail for several reasons
  - Common reasons
    - The file might not exist
    - The name might be typed incorrectly
- May be no error message if the call to open fails
  - Program execution continues!

Catching Stream Errors

- Member function `fail`, can be used to test the success of a stream operation
  - `fail` returns a boolean type (`true` or `false`)
  - `fail` returns `true` if the stream operation failed

Halting Execution

- When a stream `open` function fails, it is generally best to stop the program
- The function `exit`, halts a program
  - `exit` returns its argument to the operating system
  - `exit` causes program execution to stop
  - `exit` is NOT a member function
- Exit requires the include and using directives

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

Using fail and exit

- Immediately following the call to open, check that the operation was successful:

```
in_stream.open("stuff.dat");
if( in_stream.fail( ) )
{
  cout << "Input file opening failed.\n";
  exit(1) ;
}
```

Display 6.2

Techniques for File I/O

- When reading input from a file…
  - Do not include prompts or echo the input
    - The lines

```
cout << "Enter the number: ";
cin  >> the_number;
cout << "The number you entered is "
     << the_number;
```

become just one line

```
in_file >> the_number;
```

The input file must contain exactly the data expected

Appending Data

- Output examples so far create new files

```
ofstream out_stream;
out_stream.open("outfile.dat");
```

  - If the output file already contains data, that data is lost
- To append new output to the end an existing file
  - use the constant `ios::app` defined in the `iostream` library:

```
outStream.open("important.txt", ios::app);
```

  - If the file does not exist, a new file will be created

Display 6.3

Using File Name

```
char file_name[16];
cout << "Enter the file_name ";
cin >> file_name;
ifstream in_stream;
in_stream.open(file_name);
if (in_stream.fail( ) )
{
   cout << "Input file opening failed.\n";
   exit(1);
}
```

Display 6.4 (1)

Display 6.4 (2)

- A program can ask the user to enter the name of a file to use for input or for output
- Program can use a string variable for a file's name

Detecting The End of The File

- Input files used by a program may vary in length
  - Programs may not be able to assume the number of items in the file

- How to know the end of the file is reached?
  - The expression `in_stream >> next`
    - Reads a value from `in_stream` and stores it in `next`
    - Returns `true` if a value can be read and stored in `next`
    - Returns `false` if there is not a value to be read (the end of the file)

Detecting The End of File Example

- To calculate the average of the numbers in a file

```
double next, sum = 0;
int count = 0;
while(in_stream >> next)
 {
      sum = sum + next;
      count++;
}

double average = sum / count;
```

Detecting the End of a File

- Member function `eof` (of every input-file stream) detects the end of a file
  - `eof` stands for end of file
  - `eof` returns a boolean value
    - `true` when the end of the file has been reached
    - `false` when there is more data to read
  - Normally used to determine when we are NOT at the end of the file
    - Example: `if ( ! in_stream.eof( ) )`

Using eof

- This loop reads each character, and writes it to the screen

```
in_stream.get(next);
while (! in_stream.eof( ) )
{
      cout << next;
      in_stream.get(next);
}
```

- (! In_stream.eof( )) becomes false when the program reads past the last character in the file

The End Of File Character

- End of a file is indicated by a special character
- `in_stream.eof( )` is still `false` after the last character of data is read
- `in_stream.eof( )` becomes `true` when the special end of file character is read