

CS161: Introduction to Computer Science

Week 10

12/2019

What is in CS161 today?

- ☐ What is an external file?
- ☐ How do we save data in a file?
- ☐ How do we read data from a file?

External Files

- ❑ So far, all of our programs have used main memory to temporarily store information.
- ❑ In addition, all input and output has been done with standard-in and standard-out devices
 - this includes input from the keyboard and output to our terminal's screen for prompts, echoing data, and displaying results
- ❑ Now it is time to work with secondary storage!

External Files

- ❑ There is one big difference between main memory and secondary storage.
- ❑ Remember with main memory, each time we run our program the value of our variables is lost and we start from scratch (i.e., we don't remember what the values of the variables were when we last ran the program).
- ❑ Also, when we power down the computer all of the data stored in main memory goes away.

External Files: Secondary Storage

- ❑ With files, when you finish executing a program, the data no longer goes away!
- ❑ It stays around so you can access it the next time you run your program.
- ❑ Plus, when your computer is powered down, information stored in secondary storage is not lost (unless you have a disk crash or failure!).
- ❑ Therefore, secondary storage can be used to store data for as long as it is needed.

External Files: Secondary Storage

- ❑ We can write information to secondary storage by creating a file which consists of a collection of data.
- ❑ We then name this file, so that we can store different types of information all in one directory or on one disk.
- ❑ We will be using text files in this class (and in CS162)

External Files: Secondary Storage

- ❑ A text file contains the same kind of data that we are used to seeing on the screen of our terminals.
- ❑ What this means is that a text file is a stream of characters: line by line.
- ❑ Therefore, these are files of characters. Lines in our files can be separated by end-of-lines ('\n').
- ❑ This defines how many characters there are on a line and how many lines there are in the text file.

External Files: Secondary Storage

- To use a text file to store information,
 - we first need to include a new library of I/O functions,
 - declare input/output stream variables to be used instead of cin and cout, and
 - attach a C++ input or output stream to that file.

External Files: Secondary Storage

- ❑ Text files all have names, called filenames.
- ❑ If you do a directory
 - you will see the filenames for all of your programs.
- ❑ Text files are just one of these, commonly with the extension `.txt`.
- ❑ We must specify a filename when we want to attach our input or output stream so that we can access the correct file!

The Steps to Using External Files

- ❑ In your program when using C++ I/O stream files, you need to include the header file for a new library:

```
#include <fstream>
```

The Steps to Using External Files

- The next step is to define a set of variables that will be used to read or write to the stream for a particular file. Let's backup a moment.
- Remember we use `cin` to read from standard-in and `cout` to write to standard-out.
- Well, `cin` and `cout` are really just variables.
 - `cin` is a variable of type `istream` (input stream)
 - `cout` is a variable of type `ostream` (output stream).

The Steps to Using External Files

- Therefore, to work with files we need to create our own variables;
 - for input our variable will be of type ifstream (input file stream) and
 - for output our variable will be of type ofstream (output file stream).
- Once these variables are defined, we can attach these input or output streams to the corresponding file in our directory.

The Steps to Using External Files

- ❑ For input (i.e., reading data from a file), we can define a variable of type `ifstream`:

```
ifstream in;
```

- ❑ For output (i.e., writing data to a file), we can define a variable of type `ofstream`:

```
ofstream out;
```

The Steps to Using External Files

- ❑ The next step after this is to attach these streams to our files.
- ❑ We do this by opening files.
- ❑ To open a file to enable us to write into it, we must tie the out variable in our program/function with the filename in our directory.
- ❑ The filename is specified as either a constant, a literal string, or even as an array of characters read in from standard-input.

Opening Files

- Let's look at all three approaches of opening files for writing (i.e., output):

- Using a constant filename:

```
#include <fstream>
const char* outfile = "outdata.txt";
...    //later in a function
ofstream out;
out.open(outfile);
```

Opening Files

- Using a literal string as the filename:

```
#include <fstream>
...    //later in a function
ofstream out;
out.open("outdata.txt");
```


Opening Files

- ❑ Using an array of characters as the filename, from standard in:

```
#include <fstream>
#include <iostream>
using namespace std;
... //later in a function
char filename[12];
ofstream out;
cout << "Please enter filename: ";
cin.get(filename, 12);
out.open(filename);
```

The Steps to Using External Files

- ❑ To make sure that the file was properly opened,
 - it is best to double check that the out variable is not zero.
 - If it is, the file was not appropriately opened.

```
out.open("mydata.txt");  
if (!out)  
    cout << "Error " << endl;  
else  
    //continue with the program...
```

Writing to an Open File

- ❑ After this statement is executed, we are ready to write to the file.
- ❑ Writing to a file follows all the same rules we use for writing to the screen, but instead of using `cout`...we use our output file variable:

```
out.open(filename);  
out << "hello world";  
out.put('!');  
out << endl;
```

Writing to an Open File

- ❑ Using `open` (eg., `out.open`) always opens the file so that we begin writing at the beginning of the file, as if we had a blank file, like a clean screen or a blank piece of paper.
- ❑ If in another program we had previously written information to this file, it is lost as soon as we invoke the open function.
- ❑ If we have never written information to this file before, then the open function will create a new file for us.

Appending to a File

- It is possible to **append** information to an existing file...by specifying `ios::app` when we open a file:

```
ofstream out;  
out.open(filename, ios::app);  
if (out) {  
    out << data << endl;  
    ...  
}
```

Closing a File when done....

- ❑ Once we have named the file, and opened it for writing, we can then write information to it.
- ❑ Just think of using the new stream variable (eg., out) instead of cout as directing your output from getting displayed on the terminal to being saved in the corresponding file.
- ❑ Once done writing to the file, close the currently open file. We do that with another function included in our fstream library:

```
out.close();    //parens are necessary!
```

Working with files

☐ ***More on External Files***

- How do we Read from a File?
- What is “end of file”?
- Examine a Program using Files

The steps to read from external files

- ❑ To read from an external file, we go through the same steps that we used to write to a file
- ❑ First, make sure you have included the `fstream.h` file

```
#include <fstream>
```

- ❑ Then, define a variable of type `ifstream`

```
ifstream in;
```

- ❑ Now, we are ready to open the file to read...

The steps to read from external files

- ❑ To open a file to read from, we call the open function through our file variable:

```
in.open("text.dat");
```

or,

```
char filename[20];  
cin.get(filename, 20);  
in.open(filename);
```

- ❑ Now the file should be open, with the file pointer positioned at the beginning of the file

The steps to read from external files

- ❑ To make sure that the file was properly opened,
 - it is best to double check that the in variable is not zero.
 - If it is, the file was not appropriately opened.

```
in.open(filename);  
if (!in)  
    cout <<"Error in opening test.dat"  
    <<endl;  
else  
    //continue with the program...
```

Reading from this Opened File

- ❑ After this statement is executed, we are ready to read from the file.
- ❑ Reading from a file follows all the same rules we use for reading from the keyboard, but instead of using `cin`...we use our input file variable:

```
in.open(filename);  
if (in) {  
    in >> some_variable;  
    ...  
}
```

A Reminder about Reading...

```
in >> some_variable;
```

- ❑ Works the same as reading from standard in
- ❑ If we are reading an integer, it skips leading white space, reads digits, and stops as soon as it encounters a non-digit
- ❑ If we are reading a single character, it skips leading white space, reads 1 character
- ❑ If we are reading an array of characters, it skips leading white space, reads characters, and stops as soon as it encounters a white space character

A Reminder about Reading...

- ❑ But, if we are interested in reading in whitespace characters from the file, then using the extraction operator (>>) isn't going to do the job for us
- ❑ We will need to use the get function!
- ❑ We precede the get function by our file input variable (rather than cin).
- ❑ So, to read the very next character in a file, we can use `in.get(ch)` or `ch = in.get();`

A Reminder about Reading...

- ❑ But, if we are interested in reading in an array of characters (whitespace included), we need to use the 2 or 3 argument version of the get function
- ❑ If we want to read a line of a file until the carriage return is encountered, we can say:

```
char line[81];
```

```
in.get(line, 81, '\n');
```

- ❑ But, what if we wanted to read in the next line?
Can we say `in.get(line, 81)????` (no!)

A Reminder about Reading...

- ❑ If you want to read in more than a single line, using the get functions, one line at a time, then you need to remember to eat the carriage return!

```
char first_line[81], second_line[81];  
in.get(first_line, 81, '\n');  
in.get();    //eat the carriage return  
in.get(second_line, 81, '\n');
```

- ❑ Just as we learned with for reading from the input buffer!!

When do we Stop Reading?

- ❑ When reading from the keyboard, we can ask the user when they are finished
- ❑ But, what about a file? How can we determine when to stop reading?
- ❑ By sensing when an end of file has been encountered
- ❑ “**end of file**” is not something that is written to a file. Nor, it is something that we actually “read”

When do we Stop Reading?

- ❑ Instead, when a read operation fails because we have reached the end of file, an end of file flag gets set
- ❑ The good news is that we can check this flag, by using the following eof function:

`in.eof()` true if the previous read failed
due to end of file

`!in.eof()` true if the previous read did not
fail due to end of file

When do we Stop Reading?

- ❑ It is important to realize that end of file is not sensed after you have read the last valid thing in a file.
- ❑ Instead, you have to attempt to read at least once beyond the last valid thing in the file to have end of file be detected.
- ❑ In addition, if you perform too many reads prior to checking for whether or not an end of file flag has been set -- there is a high probability that the end of file flag will be reset!

When do we Stop Reading?

- ☐ This means, when reading from a file we need to use the following steps (order is important!)
 - Read the first thing from the file
 - While the End of File Flag is not Set
 - ☐ Process what was Read (e.g., display)
 - ☐ Read the next item from the file

Examine a Complete Example

- Let's use what we have learned to read and echo the entire contents of a file to the screen:

```
#include <fstream>
#include <iostream>
using namespace std;
ifstream in;
in.open("test.dat");
if (in) {
    char ch = in.get();
    while (!in.eof()){
        cout <<ch;
        ch = in.get();
    }
}
```

Examine a Complete Example

- In the previous program, understand the answers to the following questions:
 - What would have happened if I had read each character using `cin >> ch`
 - What would have happened if I hadn't read the first time outside the loop
 - How could I have changed this program to read one complete line of the file at a time?

Closing a File when done....

- ❑ Once we have named the file, and opened it for reading, we can then read information from it, starting at the beginning.
- ❑ Just think of using the new stream variable (eg., in) instead of cin as receiving your input from the corresponding file instead of the keyboard.
- ❑ Once done reading from the file, close the currently open file. We do that with another function included in our fstream library:

```
in.close();    //parens are necessary!
```

CS161



***Review for
the Final***

Material Covered:

- ❑ Using **functions**, writing your own functions, function prototypes, actual and formal arguments (call by value versus call by reference), and return values
- ❑ The **while** statement, the **for** statement, and the **do-while**
- ❑ The **increment** and **decrement** operators, in both their prefix and postfix forms
- ❑ **Arrays of characters** and **structures**
- ❑ **Text files**

Sample Questions:

- Show how you would read in an array of characters from the keyboard. Make sure that you don't read more characters than there is room for in the array:

```
char name[20];
```

```
Answer:    cin.get(name, 20);  
           cin.get();
```

Sample Questions:

- ☐ When should we use call by value instead of call by reference?

Answer: when you need a complete and duplicate copy of the data

Sample Questions:

- What are the benefits of call by reference?
 - List 2

Answer: 1) Can change original values of the actual arguments
 2) Doesn't require extra memory to hold a duplicate copy

Sample Questions:

- Show how you would copy one array of characters (representing a “string”) into another, given:

```
char original_name[20];
```

Answer:

```
strcpy(copied_name, original_name);
```

Sample Questions:

- Show how you would copy one array of characters (representing a “string”) into another, using a loop:

Answer:

```
int len = strlen(original_name);  
for (int i=0; i<len; ++i)  
    copied_name[i] = original_name[i];
```

Think about:

- In the previous answer, understand why the following were used:
 - prefix increment versus postfix
 - strlen was used outside the loop instead of as the test condition
 - how to turn the for loop into a while loop or a do while loop
 - how to turn this into a function (called string_copy)...what would the args look like?

Think about:

- What is the difference between the following two statements:

given:

```
char ch;
```

```
cin.get(ch);
```

versus

```
cin >> ch;
```

CS161: Introduction to CS



Good luck!