Introduction to C++

External Data Files

Topic #2

Topic #2

- ***Working with External Files***
  - What is an external file?
- **Saving data and Writing it to an External File**
  - How do we save data in a file?
- ***More on External Files***
  - How do we Read from a File?
  - What is “end of file”?
- ***Examine a Program using Files***

External Files

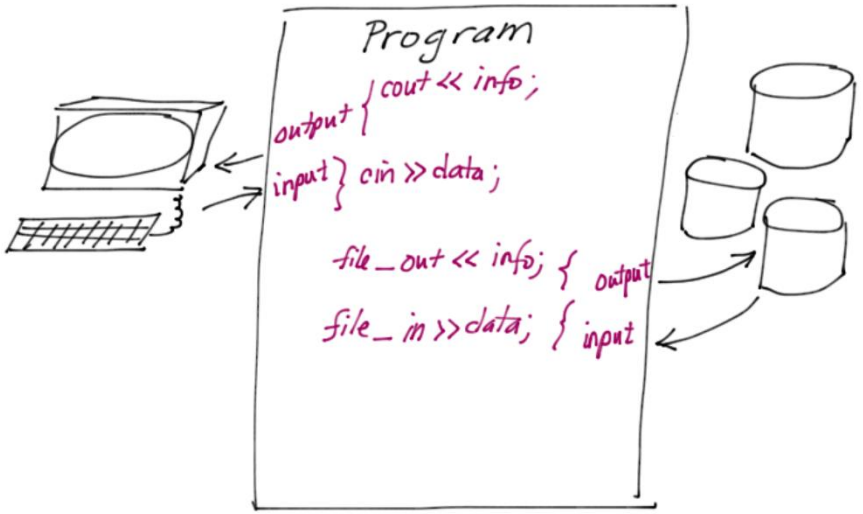
- So far, all of our programs have used main memory to temporarily store information.
- In addition, all input and output has been done with standard-in and standard-out devices
  - this includes input from the keyboard and output to our terminal's screen for prompts, echoing data, and displaying results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

External Files: 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.
- It is important to be clear with terminology
- OUTPUT is when we WRITE TO a file
  - Saving information in external storage
- INPUT is when we READ FROM a file
  - Obtaining saved information and storing it in temporary variables

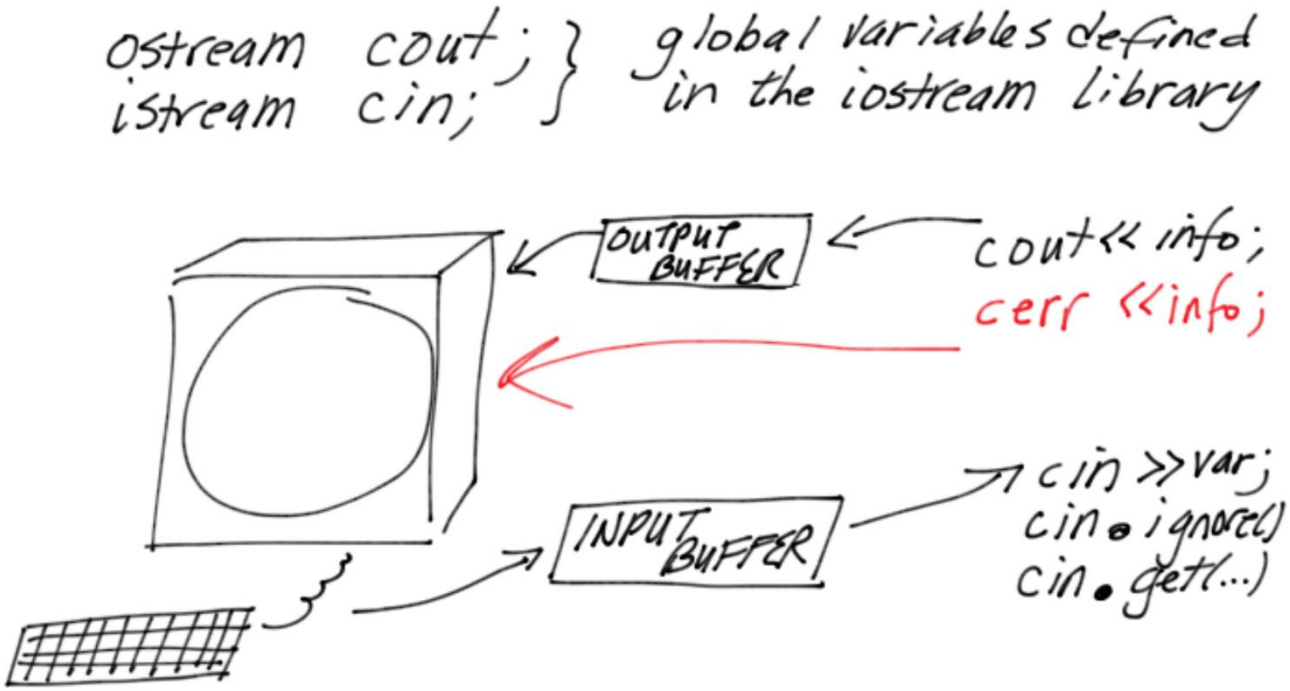
Based on What we already Know!

- Everything we already know about I/O works the same when working with external data file, except that instead of using cin and cout, we will be using file variables connected to the desired file.
- cin and cout are actually variables (called “objects”) of the istream and ostream data types (classes).
- By saying using namespace std globally, we are bringing these variables into the global namespace.



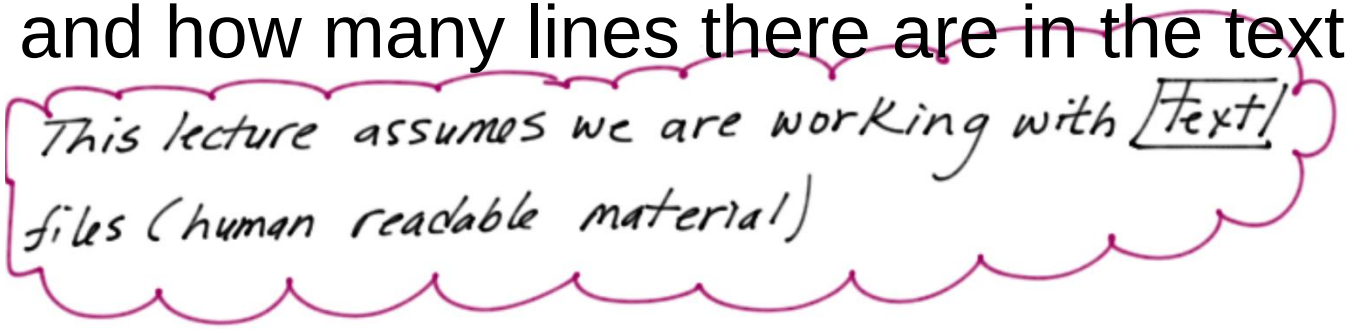
Using the IO stream library

- When we use cout, the output is buffered, which means it doesn't get displayed immediately. It waits for an event to take place such as an input request or an endl (end line flushes the output buffer)
- When we use cin, we are reading from the standard input device that is reading from an input buffer (called a stream).



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,

```
#include <fstream>   ← allows us to work with files
```

  - declare input/output stream variables to be used instead of cin and cout, and

```
ifstream file_in;   ← FOR INPUT FROM a File
ofstream file_out;  ← FOR OUTPUT TO a file
```

  - attach a C++ input or output stream to that file.

```
file_out.open("filename.extension");   //connects to a 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>
```

- You also need to include the iostream library.

```
#include <iostream>
```

- And of course bring the identifiers into the appropriate scope:

```
using namespace std;
```

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.
  - 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 current working directory.
- For this course, please do not provide a “path” for the files.
  - Data files should always be in the same directory as your program.

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

- File names are arrays of characters!!!
- Opening a file with a constant file name versus an array of characters

```
file_out.open("inv.dat");
        vs
char filename[31];
cin >>filename;
cin.ignore(100, '\n');
file_out.open(filename);
              an array of characters
```

The Steps to Using External Files

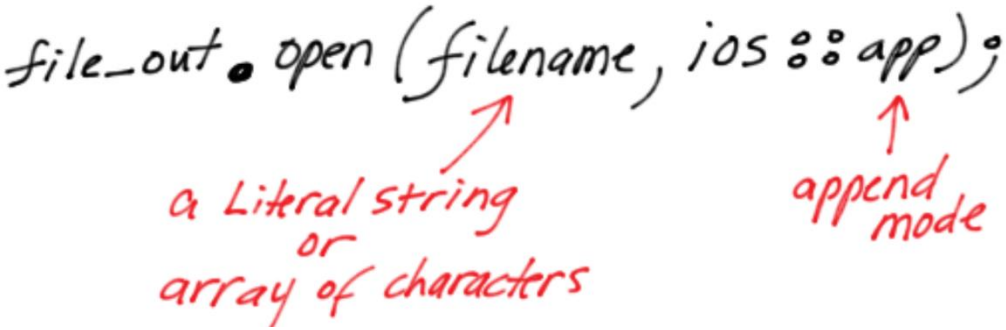
- To make sure that the file was properly opened,
  - it is best to double check that the out variable is not zero.
  - If it is, the file was not appropriately opened.

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

if(!file_out) or if(!file_out.is_open())

Writing to an Open File: IMPORTANT!

- When you open a file for output, the contents of the file is LOST!
  - If you want to preserve the information that was already in the file, then open the file to append!
- It is possible to append information to an existing file...by specifying `ios::app` when we open a file:



- Now, new information is written after the last item in the file.

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.

Writing to an Open File

- Make sure to write a newline or other delimiter after each item written to a file, otherwise we won't be able to distinguish one data item from another!

```
//Display a movie to the user AND write it
//out to a file called movie.dat
void display(/*const*/ movie & to_display)
{
   //First output to the user
   cout << to_display.title <<'\n'
        << to_display.notes <<'\t' <<to_display.length
        << endl <<endl;

   //Now let's work with files
   ofstream file_out;
   //file_out.open("movie.dat", ios::app);
   file_out.open("movie.dat");
   if (file_out) //AM I CONNECTED???? yes!
   {
      file_out << to_display.title <<':'
               << to_display.notes <<':' <<to_display.length
               << endl;
      file_out.close();
    }
}
```

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:

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

The Steps to Reading 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 file stream library:

```
#include <fstream>
```

- Then, define a variable of type ifstream

```
ifstream in_file;
```

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

When reading from a file

Inside a function:

```
ifstream file_in;  // Be careful of the name used

file_in.open(filename);
```

1. Always begins reading at the beginning of the file
2. Make sure there are delimiters in the file between fields
   ** there needs to be a way to read information back from the file

The Steps to Reading From External Files

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

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

or,

```
char filename[21];
cin >>filename;
cin.ignore(100,'\n');
in_file.open(filename);
```

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

The Steps to Reading 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_file.open(filename);
if (!in_file)
   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_file.open(filename);
if (in_file) {
    in_file >> some_variable;
    ...
```

A Reminder about Reading...

```
in_file >> 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_file.get(ch) or
ch = in_file.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 newline is encountered, we can say:

```
char line[81];
in_file.get(line, 81, '\n');
```

← you may want to replace this with a different delimiter depending on what was used!

- But, what if we wanted to read in the next line? Can we say in_file.get(line, 81)????? (no!)
- We still need to ignore the newline or whatever delimiter is used

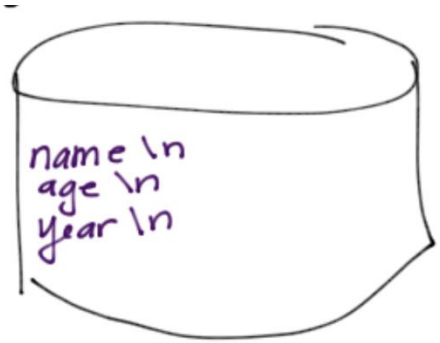
A Reminder about Reading…

- 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 get passed the newline!

```
char first_line[81], second_line[81];
in_file.get(first_line, 81, '\n');
in_file.ignore(100,'\n');                    //progress passed the delimiter
in_file.get(second_line, 81, '\n');
```

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

Everything we know applies!




```
file_in.get(namearray, size, '\n');
file_in.ignore(100, '\n');

file_in >> age;
file_in.ignore();

file_in >> year;
file_in.ignore();
```

But, when does input end?

— we can't prompt the "file" !!

∴ When we try to read from a file and there is nothing there, end of file (a "state" variable in the fstream Library) gets set.

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

- The bad news is that you need to be very careful how this is used!

Detecting End of File

1) We must attempt to read from the file to find out if there is anything in the file to read

2) file_in.eof()

function call

- Returns true if the previous read/input operation failed
- Returns false if the previous read/input was successful
- Therefore, BEFORE you check end of file, make sure to attempt to read FIRST. "Prime the Pump"

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

Can this Apply to Standard In?

- All that we have learned about end of file can also be used from standard in.
- While there is no such thing as a “file” when typing from a keyboard, we can redirect files through standard in:

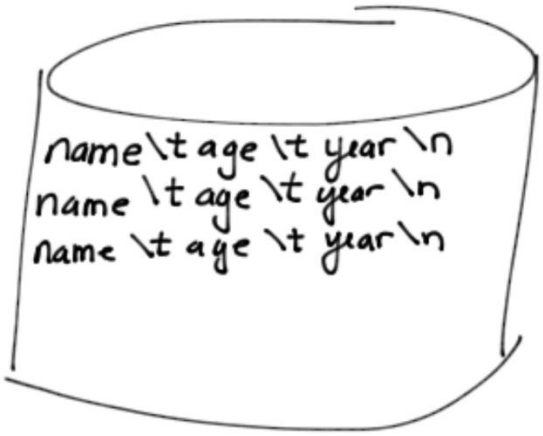
  a.out <filename

- And, this will behave has if we had typed in all of the information that was in the file (which is great except that there can be no interactions between the program and the user....

Can this Apply to Standard In?

- Using end of file from keyboard is also possible, by typing in a control-d on UNIX and a control-z on a PC.
- Once a control-d on ODIN is entered, no future input should be read from the input buffer...which means it is used to terminate a session, while still allowing the program to exit the program normally
- By..sensing end of file through !cin.eof()

Let's Read from a file!




```
ifstream fin; //file variable
fin.open(filename);
if (fin) //we are connected
{
   fin.get(name, size, '\t');
   fin.ignore(100, '\t');
   while (!fin.eof())
   { //we are not yet at end
      fin >>age; fin.ignore();
      fin >>year;
      fin.ignore(100, '\n');

      //Now prime the pump.....
      // is there another?
      fin.get(anothername, size, '\t');
      fin.ignore(100, '\t');
   }
```

notice

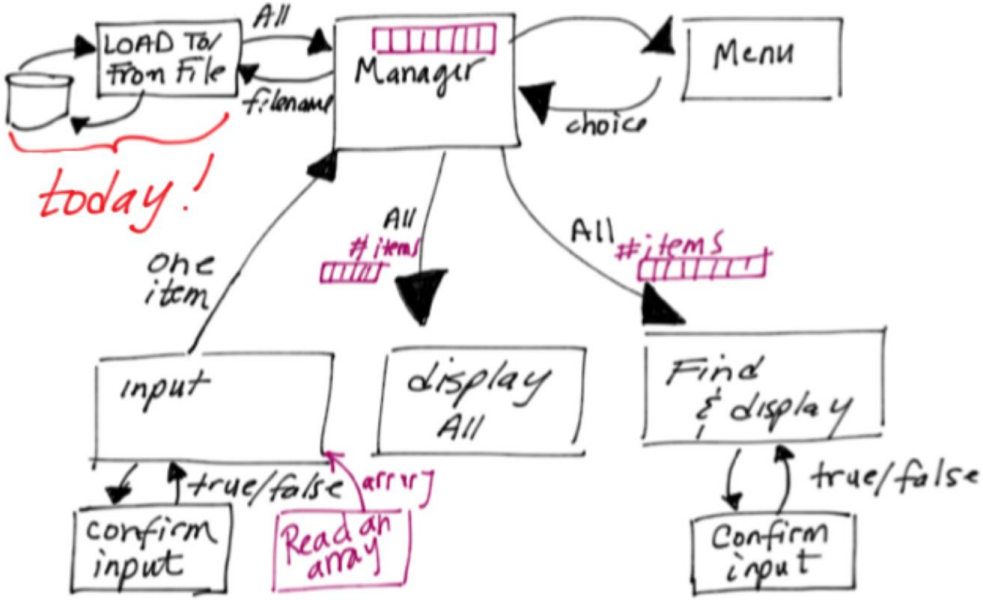
we have more data

handle the data.... display.....

is there another or are we done?

Adding External Files to our Design

- With the movie program from before, we can use structs to group different types of data together
- However, there are no operations built-in with structs to directly work with files
- We must work with each member individually



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>
ifstream in_file;

//Now in a function:
in_file.open("test.dat");
if (in_file)
{
      char ch = in_file.get();
      while (in_file && !in_file.eof())   ←Examine this!
      {
            cout << ch;
            ch = in_file.get();
      }
```

Structures and External Data Files

- Let's set this up to read a movie from a file
- First, let's get the file name and open the file:

```
//Read in the information about a movie from a file
//filling up the array of movies passed in as an argument
//and returning the number of movies read in
int load_from_file(movie array[])
{
    char file_name[21];
    ifstream in_file;
    int i = 0;

    cout << "Please enter the name of the file: ";
    cin.width(21);
    cin >> file_name;
    cin.ignore(100,'\n');

    //Connect up my file variable to the file
    in_file.open(file_name);
```

Reading with Arrays of Structures

- Next, read in the data, storing each item into a member of the structure.

```
if (in_file) //are we connected
{
    //prime the pump
    in_file.get(array[i].title, TITLE, ':');
    in_file.ignore(100,':');

   //Was the read operation successful?
    while (in_file && !in_file.eof() && i < 10)
    {       //so now read the rest!
            in_file.get(array[i].notes, NOTES, ':');
            in_file.ignore(100,':');
            in_file >> array[i].length;
            in_file.ignore(100,'\n');
            ++i; //on to the next!

            //prime the pump
            in_file.get(array[i].title,TITLE,':');
            in_file.ignore(100,':');
     }
     in_file.close();
}
return i;
```

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_file.clear();   //to reset eof!
in_file.close();   //No argument!!
```