ECE 2036 Lab 2: "Thanks for All the Fish!!!"

Due: June 12, 2017 @ 11:59
Hardcopy of Graph Due: June 13, 2017 at the beginning of class (Appendix B)
Reading: Deitel & Deitel Chapter 2-6

Please watch this video first...

www.sciencechannel.com/tv-shows/through-the-wormhole/videos/through-the-wormhole-information-theory/

or at

http://www.dailymotion.com/video/x2eye5t_information-theory-through-the-wormhole_school

In this lab you will develop a software system to collect data from a portion of the book ***War of The Worlds*** by H.G. Wells to verify the experiments performed by Dr. Lawrence. Ultimately, Dr. Lawrence is trying to apply information theory to see if the chatter of dolphins constitutes a complex language.

You will create a software system that will analyze English text by creating word frequency histograms. Some general program specifications are as follows:

1. You must develop a C++ class library to work with the `main.cc` file provided to you in Appendix A. Your libraries will be used to create a word frequency histogram from a text file that is provided to you on t-square called `ProcessedWOW.dat`. This file contains modified text for *War of the Worlds*; as you see in the file it contains only lower case letters and no punctuation marks.

2. Your program will create two output files that contain the histograms sorted in two different fashions -- one sort alphabetically and one sorted by frequency.

3. You must create at least TWO C++ classes, which are discussed in the next section.

4. We are limited to data structures that we have talked about so far; therefore, you can only use static C++ arrays to store your word frequency histograms. A discussion of a strategy to do this appears later in this lab.

5. You must have three specific files to manage your system. The header file `histogram.h` contains your class interfaces for all your classes, the source file `histogram.cc` will contain the implementations of your member functions of all your C++ classes, and the source file `main.cc` will contain the main function that is provided to you in Appendix A.

The objectives of this lab are to give you practice:

1. Using basic C++ classes;
2. Creating basic arrays of user-defined objects;
3. Creating constructors and overloading constructors;
4. Using and creating set and get member functions in C++ classes;
5. Using C++ string objects; and
6. Using basic text file I/O objects and operators.

This lab has been tested using the g++ compiler on deepthought19.cc.gatech.edu. Please see Appendix B to see your turn in options. To compile on deepthought you will need to add an option to use some of the features of C++.

```
g++ -std=c++0x histogram.cc main.cc -o make_histogram
```

C++ Class Requirements

I would like for you to create at least two separate classes for your program. The specifications are below.

WordUnit

This class will be called `WordUnit`. You will need to create a class with data members that correspond to a *string* that contains a single word of text and an *integer* that contains the number of times that the word appears in a given book. You can choose your own data member names.

WordHistogram

This class will be call `WordHistogram`. It will contain the entire word histogram for the book that you are analyzing. A primary data member will be an array of `WordUnits`. We have not discussed dynamic arrays at this point, so I would like for you to have a static fixed array with a large number of entries. For this exercise, assume that you have 10,000 elements in your array. In addition, you will need to have an *integer variable* that contains the number of elements that you are storing in this array. Furthermore, I would like for you to have a *string* that contains the file name of the text file that contains your book.

You may need other member functions, but as you see in Appendix A, you will definitely need to create the following member functions in the `WordHistogram` class.

void makeHistogram()

This member function belongs to the `WordHistogram` class and it creates the histogram from the name of the text file that is passed to the constructor. You will need to open the text file and populate the static array of WordUnits. Because we have not discussed dynamic arrays yet, please use an array "over-allocation strategy" as seen below. You will need to keep track of how much of the array is being used as you build the histogram.

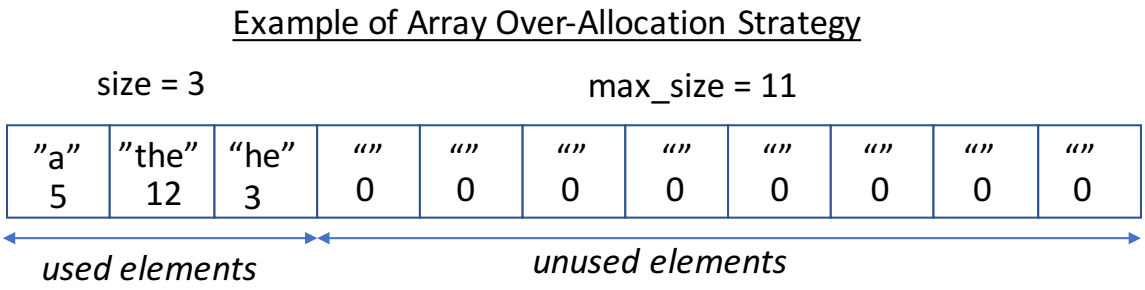

Figure 1: Illustration of using an array with a fixed size to contain a list with an initially unknown length.

void sortAlphaHistogram()

This member function belongs to the `WordHistogram` class and it will sort the `WordUnit` array in alphabetical order according to the word in each `WordUnit`. This arrangement of the array will be useful if you want to quickly find a frequency of a given word.

void sortFreqHistogram()
When this member function is called, the WordUnit array will be sorted by the frequency parameter. This will be useful when making a plot of the histogram to see if a slope of -1 in a log-log plot describes the data.

void exportHistogram(string filename)
This member function in WordHistogram will create an output file that contains the word histogram. The argument of this function is a *string* that contains the name of the output file. For the file that is alphabetized by each word, the format of the output should look something like the following:

```
a 937
abandoned 3
abandoning 1
abart 1
ability 1
ablaze 1
able 8
aboard 1
about 112
above 16
```

Please note that the word and its frequency are separated by a single space.

C++ String Objects

In this lab, you will need manipulate strings in a very basic way. For completeness, I have included in Appendix E a list of a variety of member functions for C++ string objects. You can use any of these you like, but I believe that you may find the following operators that can be used with C++ string objects more useful.

string1 == string2
The equality operator can be used to compare two strings. If ALL characters are the same, then this operation returns a true value; otherwise, it will return false.

string1 = string1 + string2;
Both the assignment (=) operator and the addition (+) operator can be used with strings. When used with strings the + operators will concatenate the two string operands.

string1 > string2
The greater than (and less than) can both be used with C++ strings objects. One string is greater than the other when it appears later in an alphabetize list. For example, "cat" is greater than "apple" because "cat" is listed after "apple" in an alphabetized list.

string1[number]
You can also use the indexing operator [] to access *each character* in the string. This is similar to accessing an array of characters with zero indexing.

Input and Output Text Files

For this lab, you will do some basic manipulation of text files. In this section, I will show you how to instantiate an input or output text file object. In addition, you can use the insertion stream (<<) and extraction stream (>>) operators to send data to an output file or receive data from an input file, respectively.

Preprocessor Directive
You will need to have the following include statement in your header file that allows you to use the C++ standard library for I/O files.

```
#include <fstream>
```

Instantiating Output File Objects
To instantiate an output file object that you can use to manipulate your output text file, you will need something like the following.

```
std::ofstream YourOutputFileObject("outputfile.dat", std::ios::out);
```

The "outputfile.dat" name is an arbitrary identifier, and this constructor will create a file in the local directory that you execute your program in. The std::ios::out is a designation that you are creating an output file; any existing file with the same name will be overwritten.

Instantiating Input File Objects
To instantiate an input file object from which you can read data, you can do the following:

```
std::ifstream YourInputFileObject("inputfile.dat", std::ios::in);
```

Like the output file example, the "inputfile.dat" name is arbitrary and specifies the name of the file in the local directory from which you would like to read. As you see in the main file, you can use this file object with the ! operator to check to see if the file is valid.

Insertion Stream Operator
Just like with the cout object, you can use the << operator to write data to an output file. You use the object name in place of cout in the following way.

```
YourOutputFileObject << "Hello File! " << std::endl;
```

Extraction Stream Operator
Furthermore, just like the cin object, you can use the >> operator to read data from an input file. You can use the object name in the following way.

```
YourInputFileObject >> string1;
```

This command will read a *single* string from the input file that is delineated by white space. For example, if the input file has the following text:

```
hello from professor Snape
```

The above line would have "hello" stored in `string1` with *no spaces*. Furthermore, the operation itself will return a true value if a string is successfully read in from the input file. This can enable you to embed this statement in a while loop condition to access all the strings in a file sequentially. For example, the following while loop will continue until all the strings have been read into the program one at a time.

```
while (YourInputFileObject >> string1)
{
   //manipulate value in string1
}
```

APPENDIX A: main.cc

```
#include "histogram.h"

using namespace std;

int main()
{
	string filename;

	cout << "Please input the file name" <<endl;
	cin >> filename;
	//I will try to open the file to see if it exists
	//You will need to open this in your member functions
	//later to read in the data
	ifstream InputFileObject(filename,ios::in);

	if (!InputFileObject) //i.e. if not valid
	{
	  cout << "The file you specified does not exist!!" << endl;
	}
	else // if successful opened the file
	{
	  InputFileObject.close(); //close the file so other M.F. can open

	  WordHistogram bookHistogram(filename);

	  bookHistogram.makeHistogram();

	  bookHistogram.sortAlphaHistogram();

	  bookHistogram.exportHistogram("histo_alpha_"+filename);

	  bookHistogram.sortFreqHistogram();

	  bookHistogram.exportHistogram("histo_freq_"+filename);

	}
}
```

APPENDIX B: Turnin Procedures

1. You may upload your source code to the deepthought cluster and submit it using the turnin script, as you did for Lab 0. You MUST ensure that your code compiles and runs properly on deepthought if you choose this method.

   From your *home* directory enter one of the following at the command prompt.

   `turnin-ece2036a Lab2`

   This automatically copies everything in your Lab2 directory to a place that we can access (and grade) it.

2. Please turn in a hardcopy of a plot of the word frequency histogram for the text that I have given you (i.e. `ProcessedWOW.dat`). This needs to be a log-log plot in order to see the slope of 45 degrees that Dr. Lawrence discusses in the video. You can make this plot with your data in any program like Excel, Numbers, or MATLAB. You will turn this plot in with your name on it at the beginning of class on June 13th to Dr. Davis.

3. (5 points extra credit) In addition to the plot for the ProcessedWOW.dat, you can earn 5 points extra credit on this lab if you also process the file given to you called moby_dick.txt. You will need to do more processing on this file to build your histogram. Also, please turn in a hardcopy plot of this data on June 13th as well.

APPENDIX C: ECE 2036 Lab Grading Rubric

If a student's program runs correctly and produces the desired output, the student has the potential to get a 100 on his or her lab; however, TA's will **randomly** look through this set of "perfect-output" programs to look for other elements of meeting the lab requirements. The table below shows typical deductions that could occur.

In addition, if a student's code does not compile, then he or she will have an automatic 30% deduction on the lab. Code that compiles, but does not match the sample output can incur a deduction from 10% to 30% depending on how poorly the output matches the output specified by the lab. This is in addition to the other deductions listed below or due to the student not attempting the entire assignment.

AUTOMATIC GRADING POINT DEDUCTIONS

Element	Percentage Deduction	Details
Does Not Compile	30%	Program does not compile on deepthought cluster!
Does Not Match Output	10%-30%	The program compiles but doesn't match all output exactly

ADDITIONAL GRADING POINT DEDUCTIONS FOR RANDOMLY SELECTED PROGRAMS

Element	Percentage Deduction	Details
Correct file structure	10%	Does not use both .cc and .h files, implementing class prototype correctly
Encapsulation	10%	Does not use correct encapsulation in object-oriented objects
Setters/Getters	10%	Does not use setters and getters for each data member.
Constructors	10%	Does not implement constructors with the correct functionality.
Clear Self-Documenting Coding Styles	5%-15%	This can include incorrect indentation, using unclear variable names, unclear comments, or compiling with warnings. (See Appendix D)

LATE POLICY

Element	Percentage Deduction	Details
First Day	20%	This is within 24 hours of program due date
Each Additional Day	20% per day	The weekend (Sat/Sun) will count as one day

Appendix D: Good Programming Practices

Indentation

When using *if/for/while* statements, make sure you indent 2 to 4 spaces for the content inside those. For example…

```
for(int i; i < 10; i++)
  j = j + i;
```

If you have nested statements, you should use multiple indentions. Your *if/for/while* statement brackets *{ }* can follow two possible conventions. Each both getting their own line (like the *for* loop) OR the open bracket on the same line as the statement (like for the *if/else* statement) and closing bracket its own line. If you have *else* or *else if* statements after your *if* statement, they should be on their own line.

```
for(int i; i < 10; i++)
{
  if(i < 5) {
    counter++;
    k -= i;
  }
  else {
    k += i;
  }
  j += i;
}
```

Camel Case (Suggested But Not Required)

This naming convention has the first letter of the variable be lower case, and the first letter in each new word be capitalized (e.g. firstSecondThird). This applies for functions and member functions as well! The main exception to this is class names, where the first letter should also be capitalized.

Variable and Function Names

Your variable and function names should be clear about what that variable or function is. Do not use one letter variables, but use abbreviations when it is appropriate (for example: “imag" instead of “imaginary”). The more descriptive your variable and function names are, the more readable your code will be. This is the idea behind self-documenting code.

Clear Comments

Some good opportunities to use comments are…

- Introducing a member function or class
- Introducing a section of code with long implementation
- Your name, class information, etc. at the beginning of the file

APPENDIX E: C++ **string** Library

String objects in C++ are powerful objects that help with string manipulation. Here is a list of useful string member functions that will be helpful in the lab and I will expect you to know for each exam. To use these, remember that you must use the following preprocessor directive.

```
#include<string>

stringobject.size();

    /*This returns the length of the stringobject*/

stringobject.find ("jeff");

    /* The member function find() returns the starting position of the first occurrence of the substring
    "jeff" in stringobject. If "jeff" is NOT found in stringobject, then string::npos is returned. Zero indexing
    is used.*/

stringobject.rfind ("jeff");

    /* Same as find() EXCEPT that it starts search from the end of the stringobject */

stringobject.find_first_of ("aeiou");

    /*This member function returns the position of the first occurance of the any of the characters in the
    string "aeiou". So in this example, this would return the first occurance of a vowel in string object */

stringobject.find_last_of ("aeiou");

    /*Same as find_first_of EXCEPT it finds the last occurrence in stringobject */

stringobject.find_first_not_of ("aeiou");

    /*This returns the first of occurrence of characters NOT in the string "aeiou".

stringobject.c_str();

    /*This converts the C++ string object into a c-style string with a NULL character at the end. */


stringobject.substr( beginningPosition, length);

    /*This returns a substring contained in the stringobject starting at beginningPosition and have a
    given length. Putting in string::npos for length will go to the end of the string *

stringobject.erase( beginningPosition, length);

    /*This will change the stringobject. It will erase the section specified by the beginningPosition and
    goes for a given length. If string::npos is put in for the length, then it will go to the end of the string.


stringobject.insert( location, "insert this text");

    /*This will insert the substring, which is "insert this text" for this example, into the stringobject start
    at location. */
```

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

int main()
{
string examplestring = "This is an example of a string example";
string fruit = "apple ";

cout << examplestring << endl;
cout <<"01234567890123456789012345678901234567" << endl;

cout << "The length of the string " << examplestring.size() << endl;

cout << "The position of \"example\" using find();" <<endl;
cout << examplestring.find("example") << endl;

cout << "The position of \"example\" using rfind();" <<endl;
cout << examplestring.rfind("example") << endl;

cout << "The position of using find_first_of(\"aeiou\");" <<endl;
cout << examplestring.find_first_of("aeiou") << endl;

cout << "The position of using find_last_of(\"aeiou\");" <<endl;
cout << examplestring.find_last_of("aeiou") << endl;

cout << "The position of using find_first_not_of(\"aeiou\");" <<endl;
cout << examplestring.find_first_not_of("aeiou") << endl;

cout << "The substr(0,6) example (rem 6 = length)" << endl;
cout << examplestring.substr(0,6) << endl;  //Do string::npos example to end

cout << "The erase(0,4) changes the string in this way." << endl;
examplestring.erase(0,4);
cout << examplestring << endl;

cout << "The insert(5,fruit) changes the string in this way." << endl;
examplestring.insert(5, fruit);
cout << examplestring << endl;

cout <<"Example of overloaded + : examplestring+examplestring+fruit " <<endl;
cout << examplestring+examplestring+fruit << endl;

}
```

```
This is an example of a string example
01234567890123456789012345678901234567
The length of the string 38
The position of "example" using find();
11
The position of "example" using rfind();
31
The position of using find_first_of("aeiou");
2
The position of using find_last_of("aeiou");
37
The position of using find_first_not_of("aeiou");
0
The substr(0,6) example (rem 6 = length)
This i
The erase(0,4) changes the string in this way.
 is an example of a string example
The insert(5,fruit) changes the string in this way.
 is aapple n example of a string example
Example of overloaded + : examplestring+examplestring+fruit
 is aapple n example of a string example is aapple n example of a string exampleapple
```