CIT 595 Spring 2018
Homework #5
Due Wednesday, May 2, 5:00pm

In this assignment, you will implement a program that will analyze the sentiment (positive or negative) of a sentence based on the words it contains.

After completing this assignment, you will be able to:

- Work with C++ strings
- Perform file I/O in C++
- Utilize common data structures from the C++ Standard Template Library
- Include external libraries in your program

You must work on this assignment **alone**. Although you are free to discuss the specification with other students, and to help people with general concepts, you should not be sharing code with anyone else. Be sure to see the section on academic honesty below.

Logistics

You can download the starter code and input file for this assignment in Canvas. Go to "Files", then "Homework Files", then "Homework #5."

For this assignment, you also need to write and turn in a Makefile that compiles your code.

In order to standardize the grading of this assignment, **you must to use your VMware image from CIT 593 or the eniac computing cluster** for compiling and running your code; please do not use Windows environments such as Visual Studio, or even your Mac or another Linux platform.

Note that the TAs will also use Valgrind to check your code for memory leaks for **all parts** of the assignment. Be sure to delete any heap memory as necessary.

Background: Sentiment analysis

Sentiment analysis is a task from the field of computational linguistics that seeks to determine the general attitude of a given piece of text. For instance, we would like to have a program that could look at the text "This assignment was joyful and a pleasure" and realize that it was a positive statement while "It made me want to pull out my hair" is negative.

One algorithm that we can use for this is to assign a numeric value to any given word based on how positive or negative that word is and then determine the overall sentiment of the statement based on the average values of the words.

To determine the sentiment of an individual word, we can use a corpus of statements, each of which has an overall score already assigned to it. The sentiment of an individual word equals the average of the statements in which that word appears.

For instance, our corpus may look like this:

```
0 This was not as much fun as I thought it would be .
1 I had a lot of fun on this and learned a lot .
-1 It would be more fun if we had more time to work on it .
2 I didn't think programming in C++ could be so much fun !
-2 I would have preferred an easier assignment .
2 I can't think of anything more fun than learning C++ !
```

Each statement is labeled with a score from -2 to 2 as follows:

- -2: very negative
- -1: somewhat negative
- 0: neutral
- 1: somewhat positive
- 2: very positive

To determine the overall sentiment of the word "fun," we take the average of the sentences in which it appears. In this case, it would be (0 + 1 + -1 + 2 + 2) / 5 = 0.8.

Then, given a new sentence, we can determine its sentiment by computing the average of the sentiments of the individual words it contains. The sentiment of any previously unseen word would be 0.

Your program will be evaluated using the set of ~8,500 movie reviews that are provided to you in Canvas. You can, of course, create your own input file for testing, but the correctness of your program will be determined using this file.

Note that, aside from this input file, we will not be providing any other sample test cases. It is up to you to determine the correctness of your implementation for all parts of this assignment.

Background: the C++ Standard Template Library

C++ provides a comprehensive set of algorithm implementations and data structures in what's known as the Standard Template Library, or STL.

In this assignment, you will use three basic data structures – vectors, sets, and maps – from the STL and will need to read the API documentation to see how they are used. The links to the documentation are provided below, but please use Piazza or speak with a member of the instruction staff if you need help with the STL classes.

Part 1. Reading the input file

Modify the file **Sentence.h** so that it defines a class called Sentence that has two private fields:

- **score**: an int that represents the sentiment score for that sentence
- **text**: a string that contains the text of the sentence

The class should also have a public constructor that initializes the two fields, and public accessor/getter methods named **getScore** and **getText**.

Then, uncomment and implement the **readFile** function in Analyzer.cpp, which you can download from Canvas. You will need to add #include and/or "using namespace" statements in order to get your code to compile; it is up to you to determine what you need.

This function should take the name of the file to read and read it one line at a time, creating Sentence objects and putting them into the vector. Note that the vector includes *pointers* to Sentence objects, and that the function returns a *pointer* to the vector.

More information about vectors is available at
http://www.cplusplus.com/reference/vector/vector/

Reading files in C++ is quite easy: http://www.cplusplus.com/doc/tutorial/files/
Note that you should be using the C++ style of reading files, and not your old C code from previous assignments.

Fort his assignment, we will assume that a well-formatted string means "starts with an int, followed by a single whitespace, and then is followed by more text." Your code should ignore (and not create a Sentence object for) any line that is not well-formatted. If the line is well-formatted, though, then the Sentence object's "score" field should be set to the int at the start of the line, and the "text" field should be set to the text that follows the whitespace after the int.

If the file cannot be opened for reading or if the filename is null, this function should return null.

Do not change the signature of this method! It must remain this way for grading. If you feel that the signature needs to be changed, please speak with the instructor.

Last, implement the "main" function in **Main.cpp** so that it reads the name of the input file as a command-line argument and invokes the readFile function that is defined in Analyzer.cpp. Use this program to determine whether your readFile function is properly returning the vector.

You will need to add #include and/or "using namespace" statements in Main.cpp to get this to compile and run, and you will need to uncomment the declaration of the readFile,

You should be able to compile the code like this:

```
c++ Main.cpp Analyzer.cpp
```

Depending on functions that you are using, you may need to tell the compiler to use the C++ 11 version like this:

```
c++ -std=c++11 Main.cpp Analyzer.cpp
```

Note: although it is generally considered good practice, we do not recommend creating an Analyzer.h header file with the function prototypes declared. This may cause problems with Part 2. It is okay for Analyzer.cpp to not have any function prototypes and for them to be declared only in Main.cpp.

Be sure your program is able to read the input file and convert it into a vector of Sentence pointers before proceeding!

Part 2. Calculating the sentiment of each word

Modify the file **Word.h** so that it defines a class called Word that has three private fields:

- **word**: a string that represents the text of the word itself
- **count**: an int indicating the number of times the word appears in the corpus
- **total**: an int that keeps track of the total sentiment value for each sentence in which the word appears

This class should also have these public methods:

- a constructor that takes a string and initializes the word field to the parameter, and initializes count and total to 0
- a method **increaseTotal** which takes an int and increases the total field by that value; it should also increment the count field
- a method **calculateScore** which returns the average (as a double) of the sentiment values, i.e., total / count; this method should return 0 if count is 0
- accessor/getter methods named **getCount** and **getWord** for the count and word fields, respectively

Then, modify Analyzer.cpp so by uncommenting the **wordComparator** struct definition, and uncommenting and implementing the **allWords** function. You may need to add #include statements in order to get your code to compile.

The allWords function should find all of the individual words in the text of each Sentence in the vector and create Word objects for each distinct word. The Word objects should keep track of the number of occurrences of that word and the total score of all sentences in which it appears. The function should then return a pointer to the set of those Word objects.

If a word appears more than once in a sentence, then that should be reflected in the Word object's "count" and "total" fields. For instance, if you had a sentence like this:

```
2 I like cake and could eat cake all day .
```

Then the Word object for "cake" should have its "count" field incremented by two (since the word appears twice in the sentence) and its "total" field incremented by four (two occurrences times a score of 2).

Tokenizing strings

As you can guess, this function needs to tokenize/split the text of each sentence to get the individual words. Unfortunately, this isn't as straightforward in C++ as it is in, say, Java.

You can certainly implement your own function to tokenize the text, or convert it to a C-style string (using the string class' c_str function) and use strtok, but a simpler way is to use the C++ Boost library, which provides numerous helpful C++ functions and classes.

You can download this file from Canvas or from https://sourceforge.net/projects/boost/files/boost/1.60.0/ and unzip the distribution file.

Then see http://www.cplusplus.com/faq/sequences/strings/split/#boost-split for an example of how to use the "split" function. Note that you will need to add the #include and "using namespace" statements as in the example. There is more information about the Boost string library at http://www.boost.org/doc/libs/1_60_0/doc/html/string_algo.html

To get your compiler to find the Boost library, you need to add it to the "include path" of your compile command using the "-I" flag like this:

```
c++ Main.cpp Analyzer.cpp -I /path/to/boost_1_60_0/
```

Note: Be careful about copy/pasting the above line from this document into your terminal window. The character before the "I" should be a regular dash, i.e. the "minus sign."

Using sets

In STL, sets are represented as binary search trees and thus require a comparator function to determine which branch to look in and whether two elements are equal. In this case, use the wordComparator struct that is provided to you; do not change it unless suggested by a member of the instruction staff. More information is available at http://www.cplusplus.com/reference/set/set/

Keep in mind that when you tokenize the text of each sentence, you will be getting strings, but the set contains Words; thus, to take advantage of the structure of the set, you will have to figure out how to use the set's "find" method instead of iterating through all elements of the set.

Note also that if the set already contains a Word with the string from the text of the sentence, you should be updating its count and total fields using the methods you created above, and not creating a new entry in the set.

Other guidelines
As you may have noticed in the data file we provided, some tokens start with punctuation and the first word of each sentence starts with a capital letter.

In producing the set of Words, your program should ignore any string that does not start with a letter, but should consider any other string, even if it contains a non-letter character, as long as it's not the first character.

Also, your program should convert all strings to lowercase so that it is case-insensitive. Unfortunately, there is no support for this in the C++ string class, so you may use the "to_lower" function from the Boost library (http://www.boost.org/doc/libs/1_60_0/doc/html/string_algo/usage.html#idm45555128664032) or modify the string one character at a time.

Check that it works!
Last, modify the "main" function in Main.cpp so that it takes the vector returned by readFile and passes it to allWords. You will need to add #include statements and function prototypes to get this to compile and run, and you will also need to uncomment the declaration of the "allWords" function.

Be sure your program is able to read the input file, convert it into a vector of Sentence pointers, and then convert that vector into a set of Word pointers before proceeding!

Part 3. Storing the sentiment of each word

Uncomment and implement the **calculateScores** function in Analyzer.cpp. You may need to add #include statements in order to get your code to compile.

This function should iterate over each Word in the set parameter, use its calculateScore method to get the average sentiment score for that Word, and then add the word (as key) and score (as value) to a map.

More information about maps is available at
http://www.cplusplus.com/reference/map/map/

Now modify the "main" function in Main.cpp so that it takes the set returned by allWords and passes it to calculateScores. You may need to add #include statements and function prototypes to get this to compile and run, and you will also need to uncomment the declaration of the "calculateScores" function.

Be sure your program is able to execute all three functions to produce the map of words and scores before proceeding!

Part 4. Determining the sentiment of a sentence

Now the fun part! Uncomment and implement the **calculateSentenceScore** function in Analyzer.cpp (the same file as above).

This function should use the map to calculate and return the average score of all the words in the sentence. If a word in the sentence is not present in the map, assign it a score of 0.

Note that you will need to tokenize/split the sentence, as you did in Part 2, and you will need to convert all words in the sentence to lowercase.

Modify the "main" function in Main.cpp so that it includes a loop that does the following:

- prompt the user to enter a sentence
- pass the map returned by calculateScores and the sentence to the calculateSentenceScore function, and print out the result
- then prompt the user again
- however, if the sentence entered by the user contains only the word "quit" (case-sensitive), stop looping and terminate the program.

You can read an entire whitespace-separated sentence from stdin (represented with the variable cin) like this:

```
string line;
getline(cin, line); // reads entire line from cin into line
```

You may need to add #include statements and function prototypes to get this to compile and run, and you will also need to uncomment the declaration of the "calculateSentenceScore" function.

Using Piazza...

Although you are encouraged to post questions on Piazza if you need help or if you need clarification, please be careful about accidentally revealing solutions.

In particular, please do not public post questions along the lines of "*my code says that 'this assignment is fun' has a score of 1.213, is that correct?*" It is important that all students determine the correctness of their program on their own, and we will not be providing test cases for this assignment.

If you think your question might accidentally reveal too much, please post it as a private question and we will redistribute it if appropriate to do so.

Academic Honesty and Collaboration

You must work on this assignment **alone**.

You may discuss the assignment specification with other students, as well as your general implementation strategy and observations, but *you absolutely must **not** discuss or share code with another student* under any circumstances.

Please see the course policy on academic honesty (posted in Canvas on the "Syllabus" page) for more information. Suspected violations of the policy will result in a score of zero for the assignment and/or be reported to the Office of Student Conduct at the instructor's discretion.

Although you can, of course, look online for help with the C++ libraries and things like that, the work you submit **must** be your own. Copy/pasting code written by other people will be considered plagiarism and will be treated as academic dishonesty, even if you were "just looking at it to see how it's done."

If you run into problems, please ask a member of the teaching staff for help before trying to find help online!

Grading

This assignment is worth a total of 100 points. Please do not change the signatures of any of the functions or classes described above.

Part 1 (Sentence class and readFile function) is worth 20 points.
Part 2 (Word class and allWords function) is worth 50 points.
Part 3 (calculateScores function) is worth 20 points.
Part 4 (calculateSentenceScore and main functions) is worth 10 points.

You also may lose up to 10 points if Valgrind reports memory leaks in your program.

The grader may also deduct up to 10 points at her discretion for things like not submitting a Makefile, submitting a Makefile that does not compile the code, egregious violations of C++ coding conventions, etc.

Submission

This assignment is due on **Wednesday, May 2, at 5:00pm**. Late submissions will be penalized by 10% if submitted up to 24 hours late, 20% for 24-48 hours late, and so on, up to one week, after which they will no longer be accepted.

To submit your assignment, put your .cpp and .h files into a *single* zip file. Include your Makefile, too. **Please do not submit the Boost library or the data file.** Also include a plain-text or PDF file describing the platform you used to compile and run your code: it must either be the CIT 593 VMware instance or the eniac cluster.

If you know that your program does not work correctly, e.g. you didn't finish all the functionality, potentially incorrect outputs, things that make the program crash, or other known bugs, please describe these in a plain-text or PDF file and include it in your submission. This will greatly help the TAs grade your assignment and make sure that they give you credit for the things that you *did* get working.

Submit the zip file into the "Homework #5" assignment in Canvas. You may submit as many times as you'd like; only the last version will be graded.

Updated: 17 April 2018, 4:14pm