CPSC 221 2016W2 Programming Project 2:
Word Frequency using Trees and Heaps

Due Date and Time:

Due: **Thursday, March 9th, 2017, at 11:59 PM** via electronic handin submission. Be sure to hand in the project using the online handin program described at the end of this document.

The **late penalty** is 3% per hour (or portion thereof), with no late assignments being accepted after 12 hours. For example, if you hand in your program at 05:40 AM on the morning after the due date, then this is 6 hours late; so, you would lose 6(3%) = 18% of the maximum possible mark.

You are allowed to work in pairs. If you work in a pair, then each partner is expected to contribute meaningfully to the project. Both partners will get the same grade. Please review **Academic Conduct** rules and keep in mind that violation of any of these rules constitutes academic misconduct and is subject to substantial penalties. If you are uncertain as to what is, or is not, reasonable collaboration, please contact your TAs or instructor.

Part marks will be awarded, so even if you don't finish everything, make sure that your code compiles on our undergrad machines (these machines tend to have the same software releases, so as long as it works on one, you're OK; our TAs will mark your code based on the undergrad environment). Use your judgment to determine whether it's worth turning the project in late, when considering the late penalties.

Objectives:

- Gain some experience using file I/O in C++
- Understand the notion of a C++ class, including member functions
- Understand how to use pointers in C++
- Work with Trees and Heaps
- More experience writing tests to verify completeness and correctness

C++ References:

Notes on basic C++ programming, and most of the topics covered in this project can be found across numerous sources, including:

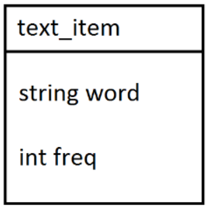
- Both of the course textbooks
- Labs 4 and 5
- Lecture Notes (Heaps begin in Unit #2)
- Online sources mentioned on the course webpage

Introduction and Information about the Project

For this project, you will be counting words in text files and keeping track of the frequency (number of times) each word was found in a text file. To do this, you will be using populating a heap and a binary search tree with `text_item` objects (shown below). A text_item has two fields: **`word`** to represent each word, and **`freq`** to represent how many times each word has been seen.

A `text_item` is defined as follows:

```
struct text_item {
        std::string word;  //the word
        int freq;  //the frequency we have seen the word
};
```



In Parts 1 and 2, you will be implementing a max-heap (prioritized by word frequency), and a binary search tree (sorted alphabetically) to organize text_item elements. In Part 3, you will use the code implemented in Parts 1 and 2 to explore the word frequencies of text files.

Starter files and usage:

Please download the .zip file containing all of the starter files for this project. The file can be found at http://www.ugrad.cs.ubc.ca/~cs221/current/assigns/word_frequency/word_frequency-student_files.zip

The compile from the command-line (assuming in the directory containing the files): `make`

The command to run and test your implementation: `./main`

Part 1: Complete the functions in max_heap.cpp

For Part 1, you will need to complete the functions `insert`, `delete_max`, `swap_up` and `swap_down` in freq_heap.cpp so items can be properly added and removed from the heap. For reference, use the header file freq_heap.hpp, lecture slides from Unit #2, and Section 8.5 from the Koffman textbook.

It is important to remember the Heap-Order Property discussed during lectures (Slide #17 from Unit #2: Priority Queues). The examples covered during lecture demonstrated how to insert and remove from a min-heap, whereas in this assignment you should implement a max-heap, with the following Heap-Order Property: **parent's key >= children's key**. The key for this assignment will be the value of the **`freq`** field of a `text_item`. Remember: this property must hold for every element contained in the heap.

For Part 1, the only files you need to modify are the functions in **max_heap.cpp** and **main.cpp**. Within these two files, there are comments that specify where you need to add functionality or tests.

<u>What you need to do (Part 1A):</u>

1) Complete the code in the `insert` and `swap_up` functions in max_heap.cpp.
2) Add tests in the `heap_insert_tests` function in main.cpp to verify the `insert` and `swap_up` functions are correct.

<u>What you need to do (Part 1b):</u>

1) Complete the code in the `delete_max` and `swap_down` functions in max_heap.cpp.
2) Add tests in the `heap_delete_tests` function in main.cpp to verify the `delete_max` and `swap_down` functions are correct.

Note: If you would like to change the number of items added to the heap for testing in Part 1, look at the `heap_tester` function in main.cpp.

Part 2: Complete a function in string_bst.cpp

For Part 2, you will need to complete the function `word_frequency` in counting_bst.cpp to return the **`freq`** value of a `text_item` by searching through your BST (which is ordered alphabetically by the value of the **`word`** field of each `text_item`).

It is important to remember that Binary *Search* Trees have the following property: all keys in left subtree of a node are smaller than the node's key, and all keys in right subtree of a node are larger than the node's key. The *key* for this assignment will be the value of the **`word`** field of a `text_item`. Words are ordered alphabetically; when comparing two words, the word that would come first in the dictionary would have a smaller key than a word that comes later in the dictionary. Remember: this property must hold for every node in the tree.

For Part 2, the only files you need to modify are the functions in **string_bst.cpp** and **main.cpp**. Within these two files, there are comments that specify where you need to add functionality or tests.

What you need to do:

1) Complete the code in the `word_frequency` function in string_bst.cpp.
2) Add tests in the `tree_tests` function in main.cpp to verify the function is correct.

Note: The tree is generated from an input text file. To change the input file, change the first parameter in the **`load_bst`** function call in the main function of main.cpp (at the bottom). Two sample text files are provided (sample1.txt and sample2.txt). The first file, sample1.txt is a rather small file, whereas sample2.txt includes the full text from *Les Miserables*. Feel free to create your own text files to test with.

Part 3: Complete functions in main.cpp

For Part 3, you will need to complete functions `overall_most_freq`, `at_least_length`, and `starts_with` in main.cpp. For Part 3, the only files you need to modify are the functions **main.cpp**. Within this file, there are comments that specify where you need to add functionality.

What you need to do:

1) Complete the code in the `overall_most_freq` to print out the 5 most common words found in a text file (which have been moved into a heap for you, and should be ordered by **`freq`** if you have completed Part 1). If the heap contains less than 5 elements, all of the elements should be printed out in order. (For example, if the heap only contains 3 elements, all three should be printed out in order based on the value of **`freq`**).
2) Complete the code in the `at_least_length` to print out the 5 most common words of a specific length. For example, if the value passed into the function for num_letters is 12, the function should print out the 5 most common words that have 12 or more characters. Similar to in 1), if the heap does not contains less than 5 elements, then all of elements should be printed out in order.
3) Complete the code in the `starts_with` to print out the 5 most common words of that start with a specific letter. For example, if the value passed into the function for starts_with_letter is 'd', the function should print out the 5 most common words that begin with the letter 'd'. Similar to in 1), if the heap does not contains less than 5 elements, then all elements should be printed out in order.
4) In Part 2, you wrote a function for a BST populated from a text file. In main.cpp, the function `text_analysis_tester` copies a tree to a heap and calls the 3 test functions you must complete for 1), 2), and 3) above. You should change the second parameter in the function calls to both functions `at_least_length and starts_with` to ensure your solution works with different parameters. (For example, your solution should be able to find the most frequent words with a different minimum string length and words that start with a different first letter, based on the parameters passed into the functions).

Sample output for Part 3:
(from sample2.txt)

```
*** Top 5 most frequent words: ***
text_item{"the",40506}
text_item{"of",19652}
text_item{"and",14786}
text_item{"a",14315}
text_item{"to",13761}

*** Top 5 most frequent words with at least 6 letters ***
text_item{"marius",1347}
text_item{"valjean",1098}
text_item{"himself",1063}
text_item{"cosette",1000}
text_item{"little",971}

*** Top 5 most frequent words that begin with c ***
text_item{"cosette",1000}
text_item{"could",675}
text_item{"come",548}
text_item{"child",453}
text_item{"can",432}
```

What to submit:

- Please submit all of the .hpp and .cpp files provided to you for the project.
- Also include a `README.txt` file that gives any special instructions or comments to the marker:
  - Your name or, if you have a programming partner, both your names
  - Important: Only one partner should submit the files, but be sure to list both students' CS userids, names, and student numbers.
      - Partners: Please make sure you communicate with each other as to who is turning in the files—and when. You can always resubmit your files before the due time. After the deadline, you'll only get one submission attempt.
    - Acknowledgment of any assistance you received from anyone but your team members, the 221 staff, or the 221 textbooks, but please cite code quoted or adapted directly from the texts (per the course's Collaboration policy)
- A file `output.txt` consisting of output from your testing of Part 1 through Part 3.

To submit, please read through the '*How to Use the Online "handin" Program'* section below.

What else you need to know:

- Besides the functions mentioned above, you should not need to add or modify the code in any of the functions in the `string_bst` or `max_heap` files, and can assume the functions work as intended. If you find any issues with the implementation of these functions, please let a TA or instructor know.

- **Do not change the function signatures** of any of the functions you have been asked to modify (`insert`, `delete_max`, `swap_up` and `swap_down` in freq_heap.cpp), (`word_frequency` in counting_bst.cpp), and (`overall_most_freq`, `at_least_length`, and `starts_with` in main.cpp). If you think you need to change the signatures based on your implementation, please contact a TA or instructor for assistance.

- You should not need to make any changes to the header (.hpp) files, util.cpp, or text_item.cpp.

- You may create helper functions to help to complete the tasks assigned in Parts 1 through 3, but the helper functions should be called from within the functions you were assigned to complete.

How to Use the Online "handin" Program

Please pay attention to the messages displayed on your screen during handin, to verify success. All submissions—on-time or not—are timestamped.

1. Create a directory called ~/cs221/proj2 (i.e., create directory cs221 in your home directory, and then create a subdirectory within cs221 called proj2).

2. To prepare to submit, first move or copy all of the files that you wish to hand in, to the proj2 directory that you created in Step 1.

3. Before the due deadline, hand in your directory electronically, as follows:

```
handin cs221 proj2
```

Note that you will receive a set of confirmation messages. If you don't get an acknowledgement, then you probably did something wrong. Please re-read the instructions and try again.

4. You can overwrite an earlier submission—unless it is now past the deadline—by including the -o flag, and re-submitting, as follows:

```
handin -o cs221 proj2
```

You can hand in your files electronically as many times as you want. The latest timestamp will determine your official handin time.

5. Additional instructions about handin, if you need them, are listed in the man pages (i.e., by typing: man handin). At any time, you can see what files you have already handed in (and their sizes) by typing the command:

```
handin -c cs221 proj2
```

Note: If your files have zero bytes, then something went wrong and you should run the original handin command (without the -c) again.

6. To see the due dates and times for our course, type:

```
handin –l cs221
```

The first pair of dates and times is the actual deadline; the second pair is the last date and time after which late assignments will no longer be accepted.