CSCI 3110 (Barbosa)
Project 6: Word Tree

Due: **See course calendar** – may not be turned in late.

Assignment ID: **proj6**
File(s) to be submitted: **proj6.cpp, wordtree.h, wordtree.cpp, makefile**

Above files must be uploaded individually.
Submit only above files.

Objectives: 1) Understand and use recursion; 2) Understand and implement binary search trees

Overview:
Write a program that reads an input text file, stores those words in a binary search tree (BST), and maintains counts the occurrence of individual words. Execute queries on the tree.

Project description:
The program should open and read an input file (named ***input.txt***), and build a binary search tree of the words and their counts. The words will be stored in alphabetical order in the tree. The program should ignore the case of the words, so that "Branch" and "branch" are considered the same. However, words that are actually spelled differently — such as "tree" and "trees" — are considered distinct, and should be stored in separate nodes. All words will be stored in the tree in their lowercase form. See attached help file for tips on how to accomplish this.

Upon reading the file and placing it into the BST, queries will be executed on the tree. These queries are stored in a file named ***queries.txt***. There are an indeterminate number of queries, and you must process them all. Two forms of queries are to be supported by the tree class:

1. A query for an individual word should return the number of occurrences of that word in the input file, as retrieved from searching the tree, or should display a message stating that the word was not found in the tree. This query will be represented in the queries file by the character ***F*** (for find) followed by the word being searched for (these 2 elements are separated by a space):
   F class – This sample query retrieves the number of times the word *class* shows up (if at all)
2. A query for words that meet or exceed a threshold number of occurrences, should result in the output of the words (and their counts) that meet that criteria, using ***inorder*** tree traversal. This type of query is shown in the file by the character ***C*** (for count), followed by the threshold number of occurrences:
   C 15 – Retrieves all words with 15 or more occurrences, and prints the actual # of occurrences
   Note: The query type character (*F* or *C*) must be in upper case.

The nodes for the word tree are a *struct* with the following members: a) A *string* containing the word itself; b) An *int* containing the count of the number of occurrences of the word; c) One pointer each to the left and right subtrees.

Requirements:

1. Your program must be split into 3 files. There will be a class (with separate interface and implementation files), and a driver file. The requirements for these are specified below:

a) The ***WordTree*** class– This class represents a word binary search tree

- Files must be named ***wordtree.h*** and ***wordtree.cpp***
- Class must be named ***WordTree***

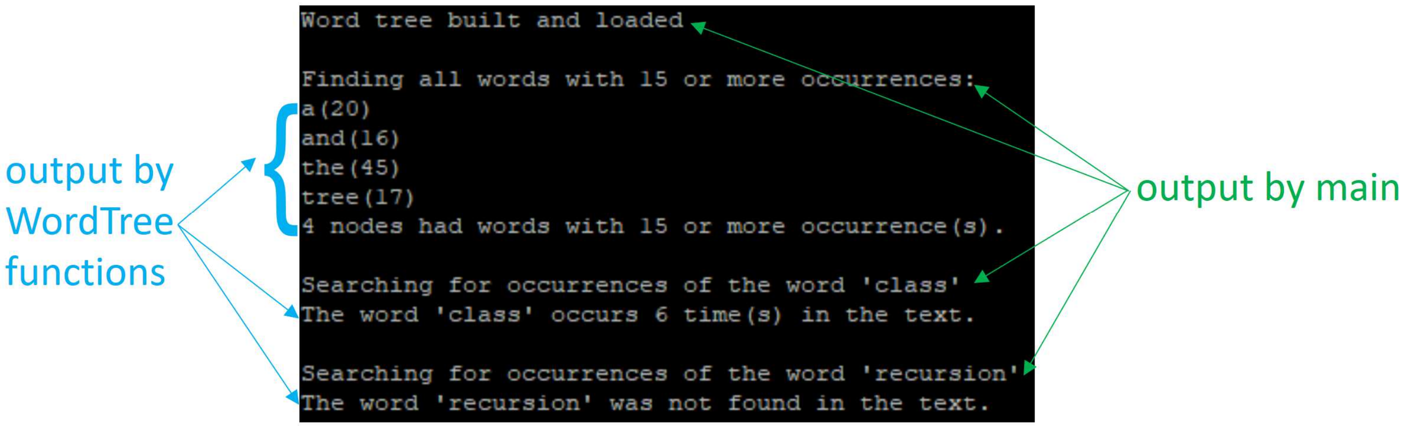
- The interface/header file is provided
- You must use appropriately named guards in the header file
- You must implement the following member functions in the class (as specified in header file)
    - A ***constructor*** – Initializes an empty tree
    - A ***destructor*** – Must have a recursive helper function that explicitly releases all nodes allocated during program execution. You **may not** rely on program termination to release memory.
    - Three functions to provide the tree's functionality. Note that the recursive functions are private, but must have a public helper function that is callable from client/driver programs. This is done so as to not expose the tree's root, its implementation, etc. to calling code.
        - ***addWord*** – adds a word to the tree. The helper function should take a string (the word to be inserted into the tree), and call the recursive function. The recursive function should add a node containing the word to the tree, if the word is not found, or increment the word's count, if it is already in the tree.
        - ***findWord*** – Iterative function that searches for a word in the tree. If the word is found, it outputs the word and its occurrence count. Otherwise it displays a message stating that the word was not found (see sample output)
        - ***getCounts*** – searches for words that meet or exceed a threshold count. The helper function takes an integer threshold value and calls the recursive function. It also outputs the number of words meeting the query, upon returning from the recursive function. The recursive function traverses the tree in order, outputting the words that meet or exceed the threshold, along with their counts (see sample output), and updating the accumulator.

b) A driver, or client, file that

- Must be named ***proj6.cpp***
- Instantiates the WordTree object
- Opens and reads the text file named ***input.txt*** and builds the word tree from the file by invoking the ***addWord*** function described above. The input file should contain a single line of words consisting of only alphabetic characters (no numbers or punctuation). It should be split using a single space character (you may use the provided function, or one you write). Care should be taken to ensure that multiple sequential spaces and trailing spaces are not present in the file/line to be read. Example file: "This is the whole file and is stored and read as a single line into a string"
- Opens and reads the ***queries.txt*** file and invokes each query, as the file is read. The format of this file is as previously described. The number of queries in the file varies. You must handle all queries in this file.

2. Sample output:

a) This output shows the program execution of primary functionality using a file containing excerpts from an earlier version of this assignment (in the attached help file). That text is supplied to aid you in validation/testing. The query file used in example is also provided.



```
Word tree built and loaded

Finding all words with 15 or more occurrences:
a(20)
and(16)
the(45)
tree(17)
4 nodes had words with 15 or more occurrence(s).

Searching for occurrences of the word 'class'
The word 'class' occurs 6 time(s) in the text.

Searching for occurrences of the word 'recursion'
The word 'recursion' was not found in the text.
```

3. Test your program - Use different input and query files to test the program.

4. Code comments - Add the following comments to your code:

- A section at the top of the source file(s) with the following identifying information:

  Your Name
  CSCI 3110-00X (your section #)
  Project #X
  Due: mm/dd/yy

- Below your name add comments in each file that give an overview of the program or class.
- Place a one or two-line comment above each function that summarizes the workings of the function.

5. Rubric

Requirement	Points Off
WordTree Class	
Class name	-5
Include guards	-3
Constructor – create empty tree	-5
Destructor – invokes recursive helper function	-5
Destructor – explicitly and correctly releases all dynamically allocated memory	-10
addWord – (helper function) accepts word string and invokes recursive function	-5
addWord – (recursive function) identifies when word is not in tree and adds its node (in correct spot)	-15
addWord – (recursive function) identifies when word is in tree and increments its count	-5
findWord – iteratively searches trees for a word	-10
findWord – outputs the word and its occurrence count if word is found (see sample output)	-5
findWord – outputs appropriate message when word is not found (see sample output)	-5
getCounts – (helper function) accepts count threshold and invokes recursive function	-10
getCounts – (helper function) outputs the number of nodes meeting the threshold criteria	-5
getCounts – (recursive function) print word (and occurrences) that meet/exceed threshold in order	-15
getCounts – (recursive function) accumulates the number of nodes meeting the threshold criteria	-10
Driver/Client	
main – use specified file names	-5 (per file)
main – correctly opens input file	-5
main – correctly instantiates Tree object	-5
main – correctly loads words into tree	-10
main – outputs tree built message	-3
main – correctly opens queries file	-5
main – outputs each query to be run (in turn)	-3
main – correctly invokes each query in the queries file (in turn)	-5 (per query)
main – output matches sample and specifications (including blank lines)	-5
general – Does not compile (on *ranger* using Linux g++ compiler, and the provided makefile)	-25
general – Runtime crash	-25
general – Other	TBD