CSCI 3110 (Barbosa S18)
Project 6: Word Tree

Due: **Apr 4 by 11:59 PM** – may be turned in until Apr 11 by 11:59 PM with reduced points (per Project guidance found in the course syllabus).

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

Above files must be uploaded individually.
Submit only above files. DO NOT submit a zip file.

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

Overview:
Write a program that reads an input text file and counts the occurrence of individual words in the file. You will use a binary tree to keep track of words and their counts.

Project description:
The program should open and read an input file (named ***input.txt***) in turn, 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 to be different, and should be stored in separate nodes. All words will be stored in the tree in their lowercase form. Help is provided on how to accomplish this.

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.
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.

The nodes for the word tree should be a *struct* with the following members

a. A *string* containing the word itself.
b. One pointer each for the left and right subtrees.
c. An *int* containing the count of the number of occurrences of the word.

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***
- You should implement the following member functions in the class
  - A ***constructor*** – Creates an empty tree
  - A ***destructor*** – Recursive function that explicitly releases all nodes allocated during program execution. You **may not** rely on program termination to release memory.

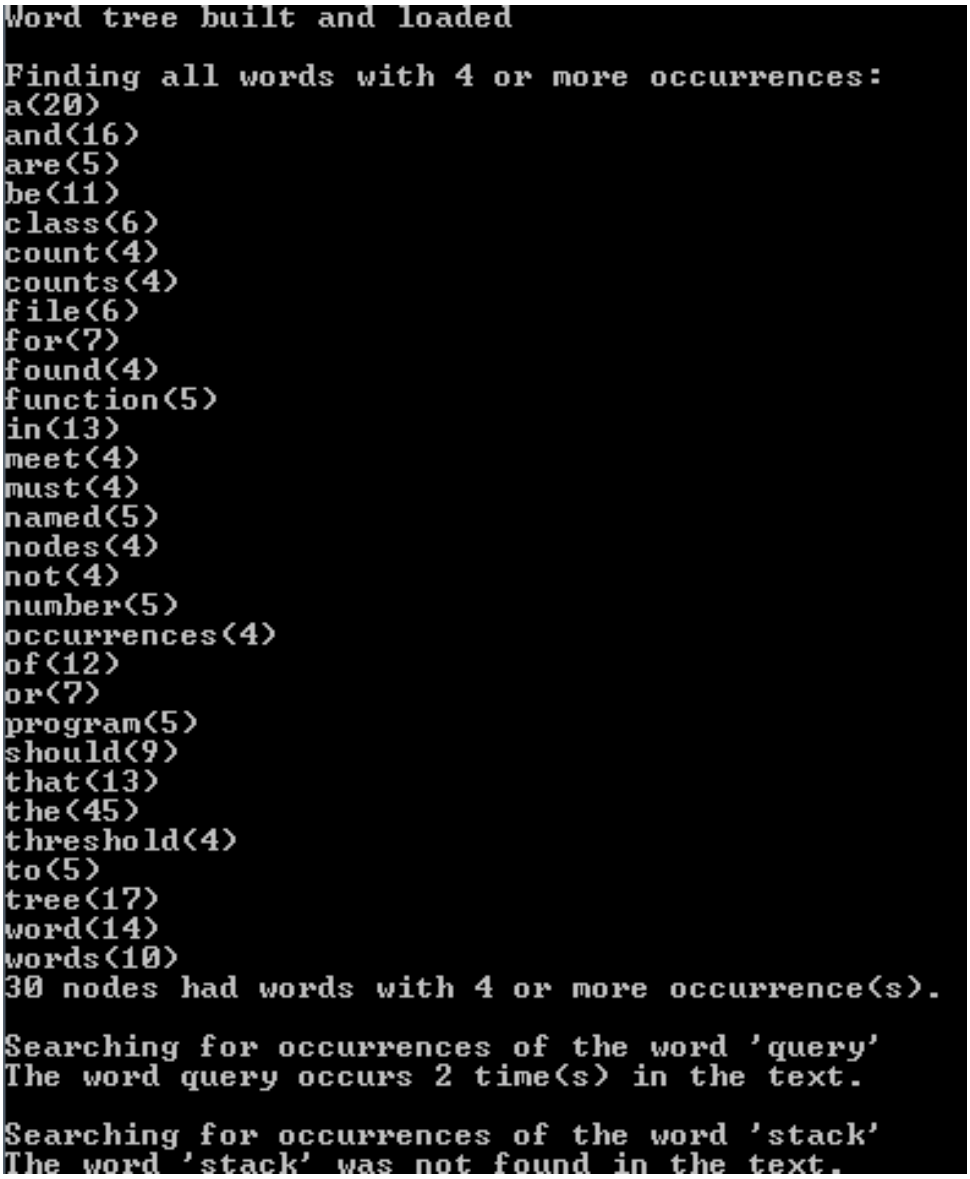
- ***insert*** – Recursive function that adds a word to the tree, if it is not found, or increments its count if it is already in the tree.
- ***findNode*** – accepts a string argument (a word) and searches for the word in the tree. This function should be public, but should call a private function, so as to not expose the tree's root, its implementation, etc. If found, outputs the word and its count. Otherwise displays a message stating that the word was not found.
- ***printInOrder*** – Recursive function that accepts a single integer argument (a threshold value), and traverses the tree in order, outputting the words (and their counts) that meet or exceed the threshold count. The function should also output the number of nodes meeting the criteria (see sample output).

b) A driver, or client, file that

- Must be named ***proj6.cpp***
- Instantiates the word tree object
- Opens and reads the text file named ***input.txt*** and builds the word tree from the file by invoking the ***insert*** function described above. The input file should contain a single line of alphabetic characters (no numbers or punctuation). It should be split using a single space character via 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"
- Invokes queries for individual words, or for words whose counts meet or exceed a threshold number of occurrences, as shown in the sample output below.

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. That text is supplied to aid you in validation/testing



3. Test your program - Use different data files and queries 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.