CSCI 3110   Project 6

A common problem in textual analysis is to determine the frequency and location of words in a document. The information is stored in a concordance, which lists the distinct words in alphabetical order and makes references to each line on which the word is used. For instance, consider the quotation

> *Peter Piper picked a peck of pickled peppers. A peck of pickled*
> *peppers Peter Piper picked. If Peter Piper picked a peck of*
> *pickled peppers, where is the peck that Peter Piper picked?*

The word “piper” occurs 4 times in the text and appears on lines 1, 2, and 3. The word “pickled” occurs 3 times and appears on lines 1 and 3.

For the text above, the output of the concordance is:

```
Word                Frequency           Occurs on lines:
-----------------------------------------------------------------
a                        3              1, 2
if                       1              2
is                       1              3
of                       3              1, 2
peck                     4              1, 2, 3
peppers                  3              1, 2, 3
peter                    4              1, 2, 3
picked                   4              1, 2, 3
pickled                  3              1, 3
piper                    4              1, 2, 3
that                     1              3
the                      1              3
where                    1              3
```

Write a program to create a concordance for a text file. Since during the process of building the concordance, it is frequently necessary to look up the word in the current words in the concordance, update concordance entries, as well as insert new words into the concordance, a binary search tree data structure is a good candidate for the application.

A word is a consecutive sequence of letters of the alphabet. Your concordance should not be case sensitive (i.e. Pickled and pickled are the same words). Your program should input the text word by word, keeping track of the current line (line 1, line 2, etc.). Extract each word and insert it into a binary search tree. Each node of the tree should have the form:

word	frequency count	list of line numbers	last line number

The node should also have a method that returns the key value for the data. The key of data in this application is the word. All the words are arranged in the binary search tree based on the alphabetical order of the words:

1. If the word is encountered the first time, a new data is created. The data includes the word, frequency count of 1, the line number in the list of line numbers, and

   the line number as the "last line number" see so far. Inserted the new data into the tree.
2. If the word is already in the tree, update the frequency and line number list. After reading the file, print an alphabetized list of words, the frequency count, and the ordered list of lines on which the word occurred.

Program Requirements:

1. A node struct is defined with the required data members and key function
   a. Use STL list for the list of lines.
2. Write a **binary search tree** class for organizing and storing the words
3. Develop user defined functions in the main(client) program to support modularity

Helpful hints:

a) Create a Print function in the client program and pass it in the BST traversal function as the *visit* function.

```
For example,
void PrintNode(treeItemType & data)
{
    cout << data.word << " " << data.count << " " << data.lastline << endl;
}
```

In the main function,
Assuming you have an object of the BST class

```
BSTclass mytree;

// code to build the tree by inserting data into the tree one by one
…

// now you want to print the tree with inorder traversal
myTree.InorderTraverse(PrintNode);
```

b) Stringstream: read information from a string

```
// C++ program to count words in a string using stringstream.
#include <iostream>
#include <sstream>
#include<string>
using namespace std;

int countWords(string str)
{
   // Breaking input into word using string stream

   // Used for breaking words
   stringstream s(str);

   // To store individual words
   string word;
```

```
    int count = 0;
    while (s >> word)
        count++;
    return count;
}

// Driver code
int main()
{
    string s = "geeks for geeks geeks "
               "contribution placements";
    cout << " Number of words are: " << countWords(s);
    return 0;
}
```

c) Change string to upper or lower case string

```
#include<bits/stdc++.h>
using namespace std;
 int main(){
    // s1 is the string which is converted to uppercase
    string s1 = "abcde";

    // using transform() function and ::toupper in STL
    transform(s1.begin(), s1.end(), s1.begin(), ::toupper);
    cout<<s1<<endl;

    // s2 is the string which is converted to lowercase
    string s2 = "WXYZ";

    // using transform() function and ::tolower in STL
    transform(s2.begin(), s2.end(), s2.begin(), ::tolower);
   cout<<s2<<endl;

    return 0;
}
```