

## Programming Assignment 5

Name: Dewey Kincheloe

Assignment Number: Programming Assignment 5

Assignment Title: Programming Assignment 5

Due Date: November 11, 2016

## Programming Assignment 5

### Abstract

#### Problem Description:

This program creates a spell checking “dictionary” then uses it to spell check a book. An array of 26 Binary Search Trees (BST) is created. One for each letter of the alphabet. Thus the first element would contain only those words starting with letter 'a', while the last would contain only those words starting with the letter 'z'. This acts as the directory for all the words from A to Z. The program reads the “random\_dictionary” file, converts all the words to lower case then uses these words to populate all the words in the dictionary from a to z.

The Oliver book is then read one word at a time. As each word is read, it is converted to lower case then ran through an algorithm to ensure it is a word containing only the letters a-z. Another algorithm is used to select the first character of this word then compare it to the words in the dictionary based on the words available in the dictionary starting with that letter. If that word is not found, it is either misspelled, or not in the dictionary. It is then counted as a word not found. If the word is found, it is counted as a word found.

The program also counts the number of string comparisons that are done during search using two counters. One is for the number of comparisons for words that were found in the dictionary, and the other is for the number of comparisons for words that were not found in the dictionary. For example, if the word is “apple” and “apple” is a word in the dictionary, the counter for words found will be incremented. Similarly, if the word “apple” is not found in the dictionary then that counter would be incremented.

At the end, the program computes and displays the average number of string comparisons for words that were found and the words that were not found.

## Programming Assignment 5

```
/**
 * C202 Assignment 5
 * TestBinarySearchTree.java
 * Purpose: This program creates a spell checking "dictionary" then uses it to
 * spell check a book.
 *
 * @author Dewey Kincheloe
 * @version 1.0 11/11/2016
 */

import java.util.*;
import java.io.*;

public class TestBinarySearchTree {

    /**
     * The removeSpecials method receives a word from the oliver text file and
     * removes all special characters thus ensuring it is only composed of the
     * letters a-z. If it is a word composed entirely of special characters, it is
     * converted to an x so it will still count as a word to be checked.
     *
     */

    public String removeSpecials(String word)
    {
        String cleanWord = "";
        for (int i=0;i<word.length();i++)
        {
            //Ascci range for a-z 97-122

```

## Programming Assignment 5

```
        if (word.charAt(i)>96&&word.charAt(i)<123)
        {
            cleanWord += word.charAt(i);
        }
    }
    if (cleanWord == "")
    cleanWord = "x";
    return cleanWord;
}

/**
 * The main method controls program execution.
 *
 */
public static void main(String[] args) {

    long wordsFound = 0;
    long wordsNotFound = 0;
    long compsFound = 0;
    long compsNotFound = 0;
    int count = 0;
    TestBinarySearchTree obj = new TestBinarySearchTree();
    BinarySearchTree<String> dictionary = new BinarySearchTree<>();

    File f = new File ("random_dictionary.txt");
    String inputWord;
    try{
        Scanner in = new Scanner(f);
        while ( in.hasNext())
        {
```

## Programming Assignment 5

```
inputWord = in.next();
inputWord = inputWord.toLowerCase();
dictionary.insert(inputWord);
```

```
}
System.out. println("Processing complete");
in.close();
}
catch(IOException e)
{
    System.out.println("Unable to read file");
}
```

```
File book = new File ("oliver.txt");
try{
    Scanner in = new Scanner(book);
    while ( in.hasNext())
    {
        inputWord = in.next();
        inputWord = inputWord.toLowerCase();
        inputWord = obj.removeSpecials(inputWord);
        int i = (inputWord.charAt(0)-97);
        count ++;
        if (dictionary.search(inputWord))
        {
            wordsFound++;
            compsFound += count;
        }
    }
    else
```

## Programming Assignment 5

```
{
    wordsNotFound++;
    compsNotFound += count;
}
}
System.out. println("Processing complete");
in.close();
}
catch(IOException e)
{
    System.out.println("Unable to read file");
}
System.out.println("wordsfound " + wordsFound);
System.out.println("wordsnotfound " + wordsNotFound);
System.out.println("compsfound " + compsFound);
System.out.println("compsnotfound " + compsNotFound);
double avgcompswordsfound = (double) compsFound / wordsFound;
System.out.println("avgcompswordsfound " + avgcompswordsfound);
double avgcompswordsnotfound = (double) compsNotFound / wordsNotFound;
System.out.println("avgcompswordsnotfound " + avgcompswordsnotfound);
}
}
```

## Programming Assignment 5

run:

Processing complete

Processing complete

wordsfound 914054

wordsnotfound 67937

compsfound 451415249017

compsnotfound 30738404019

avgcompswordsfound 493860.5914059782

avgcompswordsnotfound 452454.53904352564

BUILD SUCCESSFUL (total time: 8 seconds)