

## CSCI 235 – Software Design & Analysis II

### Assignment 6

---

### **Introduction**

This assignment gives you practice programming with trees. You may complete this assignment alone or in a group of 2 – 3 people. If you implement this program in a group, all group members receive the same grade.

### **Submission and Grading**

Submit your program's source code on Blackboard. Your program will be graded as follows:

- 55% = Correctness
- 15% = Performance
- 10% = Style
- 10% = Documentation
- 10% = Design

### **Assignment**

Write a program which does the following:

- 1) Reads a text file
- 2) Parses and processes the file
- 3) Creates an in-memory index of the words
- 4) Allows the user to interact with the index

The program must read the file, the name of which is provided by the user. When the program reads a new text file, it must remove index data left over from any previous run. When a file is parsed and processed, the program must create a binary search tree of the words in the file and allow the user to search for a word. If the word is present in the processed file, the program must display all line numbers on which the word occurs. If the word is not present, the user must be so informed. This search must be performed so that line numbers of words are displayed regardless of whether the source has uppercase, lowercase and mixed case versions of the word. Also, the search must be insensitive to the punctuation around it.

For example, if the file is as follows:

```
I, too, dislike it.  
    Reading it, however, with a perfect contempt for it, one  
    discovers  
in it after all, a place for the genuine.
```

... the user's "search" interaction may be as follows (with the user's responses in italics):

Command: *find reading*

Word "reading" appears on line: 2  
Command: *find it*  
Word "it" appears on lines: 1, 2, 4  
Command: *find Marianne*  
Word "marianne" does not appear in this file

The program must allow the user to add or remove words from the index. Adding a word means that the index will contain the word. The program may claim that any word added in this way is on line "0." Any word the user requests be removed from the index must be actually deleted (i.e. not "soft" deleted). Any subsequent searches for the removed word will treat the word as though it does not exist in the file. The interaction may be as follows (with the user's responses in italics):

Command: *add Moore*  
Word "moore" added to index  
Command: *remove the*  
Word "the" removed from index

The program must allow the user to load a new file (which removes the index of any previously-read file). The interaction may be as follows:

Command: *load poetry.txt*  
File "poetry.txt" read & index generated