Assignment #5

Topic(s):	Binary Search Trees	
Date assigned:	Friday, November 1, 2019	9:15 am
Date due:	Monday, November 11, 2019	9:15 am
Points:	50	

For this assignment, you are to implement a Binary Search Tree in a file called tree.c using the header file tree.h. You can find this header file on zeus in **/home/CS300Public/2019/05Files**. All of the data structures and function prototypes are defined in the header file. This tree contains a string (C char array) as a key and an int as a value. Further, each function prototype has been described to the point that you should be able to implement each function. The error codes that can be produced are listed for each function. Higher precedence error codes are listed first.

I recommend that you produce a test driver, treeDriver.c, that thoroughly tests your tree. For treeDriver.c only, your testing code can reach into the Tree structure to determine if your functions are operating correctly.

Finally, write a driver, wordCountDriver.c that solves the problem listed below using your tree. You may any helper functions to tree.c as necessary. Be sure to mark helper functions as **static** so they are not visible in your driver.

Name your Eclipse project **TreeWordCount**. You must produce a Makefile that, at minimum, will build the project, run **Valgrind** (make valgrind) as described below, and build a **tarball** (make tarball) for submission.

This assignment does not rely on any previous projects.

The Tree:

Each TreeNode contains a statically allocated character array, an int for the count, and two child pointers (left and right).

Example driver:

```
TreeNodePtr psRoot;
char szData[WORD_MAX+1];
int value;

trLoadErrorMessages();
trCreate(&psRoot);

strncpy(szData, "Hello", WORD_MAX);

trInsert(&psRoot, szData, 5);
trFind(psRoot, szData, &value);

printf(“>> %d\n”, value);

trTerminate(&psRoot);
```

wordCountDriver:

Your driver must read words from a file. The file is given as a command line argument as in Airport. Words are always separated by at least one whitespace. No numbers or punctuation will be present in the file. The maximum printable characters in a word is WORD_MAX. There may be any number of words in the file. The file may or may not end with a blank line.

For each unique word, you must determine how many times that word appears in the file. The key for your tree is the word, the value stored in your tree is the number of times that word appears.

Once the file is completely read, print each word you found in alphabetical order and the number of times that word was found. See example output below.

Output	Input File. **data/words.txt**
Computer 2 **Done 2** **Eclipse 1** **Oregon 2** **Pacific 2** **Science 1** **word 3**	**word word word Computer** **Computer Science** **Oregon** **Pacific Oregon Pacific Done** **Done** **Eclipse**

Valgrind

Your valgrind target must give the file **data/words.txt** (the file above) as the command line argument. Use the options specified for Valgrind in the Airport assignment to check your tree.

Submit a color, double-sided, stapled packet of code by the deadline. The packet must be in the following order:

Makefile
tree.c
wordCountDriver.c

String processing in C

To compare NULL-terminated strings in C, you will need to use
`int strncmp(const char *s1, const char *s2, size_t n)`.

To copy a string in C, use **`strncpy(char *dest, char* src, size_t n).`**

See ZyBook: sections 15.15, 16.12, 18.11, 21.5. Note the ZyBook discusses strcmp() and not str**n**cmp().