typos corrected on 4/11/2013

CS303 Programming Assignment #5: "Implementing a Binary Search Tree (BST)"

Out:	April 8th, 2013. **Due**: April 17, 2013 by 7:00 pm on the CS303 Moodle site.
Total points:	100. approximately 20% of the total homework grade.
Lectures:	Binary search trees were discussed in Lectures 30 - 32.
Useful textbook pages:	Program 5.17 has an example of counting the nodes in a tree and determining the height of a tree. Program 12.9 has an example of non-recursive insertion into a BST.

Students are given the opportunity to write a C program to implement a binary search tree data structure.

You must write a C program which has the following features (70 points):

A command-line program that:

1. Accepts a single string as a command-line parameter.
    a. Extend the usage() function in main.c so that if the string is longer than 10 characters, the program should print a useful error message and exit.
2. Uses createNextPermutation() in algo.c to creates an array containing all permutations of the given string.
3. Prints all of the permutations to the console.
4. Uses shuffle() in algo.c to shuffle the array.
5. Creates a second binary search tree using the randomized array as input.
6. Uses count() in tree.c to count the nodes in the tree.
7. Prints the number of nodes in the tree.
8. Uses printTree() in tree.c to print the contents of the tree.
9. Uses height() in tree.c to calculate the height of the tree.
10. Prints the height of the tree.
11. Frees memory using freePermutations() in main.c and freeTree() in tree.c

Here is an example expectation of the output:

```
./prog5 abv
************************************************************
 Welcome to the tree builder!

All permutations of 'abv':
   abv, avb, bav, bva, vab, vba.

Shuffled permutations:
   bva, abv, vba, avb, bav, vab.

The tree contains 6 nodes:
   abv, avb, bav, bva, vab, vba

The worst-case height of the tree: 6
The best-case height of the tree: 3
The height of the actual tree: 4
************************************************************
```

The main() function that performs all steps above has been written for you. To complete the assignment, students must write these functions: createNextPermutation(), shuffle(), insert(), printTree(), height(), count(), freeTree().

Stubs have been provided in the starter code, prog5.zip.

*

To receive 100/100 points on this assignment, you must utilize good programming practice (30 points):

• Variables must have meaningful names and global variables must not be used.	2
• Preprocessor directives must be used for constant values.	2
• Code must be documented with useful comments and should use standard tabbing rules for good readability.	9
• Code should not be redundant. If two snippets of code have similar functionality, make a function or write a loop.	6
• A makefile must be used to compile the program and should be submitted with your homework submission.	2
• All files opened by the program and all memory allocated to the program should be closed and freed before program exit.	5
• In the final submission, students should not use any printf() statements in the tree.c functions they implement; that said, printf() is a welcome debugging tool.	4
Total	30

*

To achieve maximum points on your submission, consider using this submission checklist before submitting your program to the Moodle course website:

- ☐ "I did not change the name of the source files in the starter code. The names of my program sources are main.c, algo.c, algo.h., tree.c, and tree.h"
- ☐ **"**I submitted a makefile."
- ☐ "My program compiles successfully with the makefile I submitted."
- ☐ "I ran all the tests (see above) to make sure my program executes correctly."
- ☐ "I followed the five pieces of guidance on commenting programs."
- ☐ "I compressed my source *files* into a zipfile named with my username, e.g., crenshaw13.zip"
- ☐ "I did **not** compress my source files using .rar, .z7, or some other proprietary compression program."
- ☐ "I did **not** compress a DIRECTORY of files."
- ☐ "I uploaded my zipfile to Moodle."

Extra Credit. 10 points.

The starter code has defined a Binary Search Tree data structure with the following node type:

```
typedef struct treeNodeTag treeNode;
struct treeNodeTag {
  char * permutation ;
  treeNode *left;
  treeNode *right;
};
```

Add to this definition a count field. Each node's count reflects the number of nodes in the subtree rooted at the current node, including the node itself.

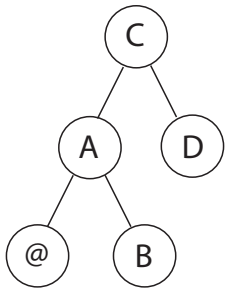

Figure 1. A simple Binary Search Tree with five nodes. The node labeled "A" has a count value of 3. The node labeled "B" has a count value of 1. The node labeled "C" has a count value of 5.

Alter makeNode(), insert(), and printTree() so that the count of each node is initialized, maintained, and printed during program execution.

```
./prog5 abv
************************************************************
 Welcome to the tree builder!

All permutations of 'abv':
   abv, avb, bav, bva, vab, vba.

Shuffled permutations:
   bva, abv, vba, avb, bav, vab.

The tree contains 6 nodes:
   abv (3), avb (2), bav (1), bva (6), vab (1), vba (2)

The worst-case height of the tree: 6
The best-case height of the tree: 3
The height of the actual tree: 4
************************************************************
```