

# CSCI 255: Lab #5 BST Recursion

Darwin Jacob Groskleg

Tuesday, October 8th, 2019

## Contents

<b>Contents</b>	<b>i</b>
<b>Questions</b>	<b>1</b>
Question 1 . . . . .	1
Question 2 . . . . .	1
Question 3 . . . . .	1
<b>test__bst.cpp</b>	<b>2</b>
<b>bst.hpp</b>	<b>4</b>

## Questions

### Question 1

What is the output from `reverse()` with input “CS255”:

- input: “CS255”
- output: “552SC”

### Question 2

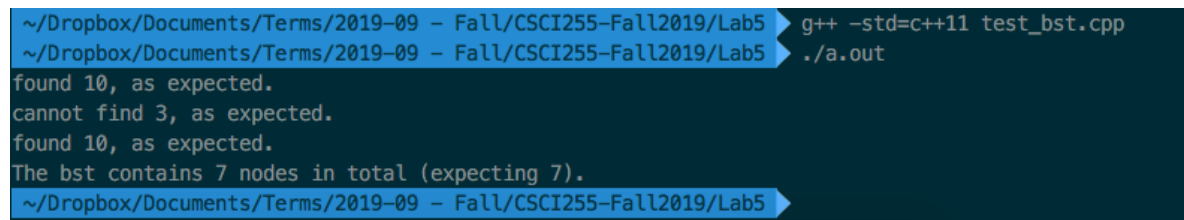
Output from `cubes` with `n=5` : “1 8 27 64 125”.

Recursive definition of `cubes`:

```
void recursiveCubes(int n) {  
    if (n > 0) {  
        recursiveCubes(n-1);  
        cout << n * n * n << " ";  
    }  
}
```

### Question 3

Add the recursive implementations of `count` and `search` on BST. See code on following pages.



```
~/Dropbox/Documents/Terms/2019-09 - Fall/CSCI255-Fall2019/Lab5 g++ -std=c++11 test_bst.cpp  
~/Dropbox/Documents/Terms/2019-09 - Fall/CSCI255-Fall2019/Lab5 ./a.out  
found 10, as expected.  
cannot find 3, as expected.  
found 10, as expected.  
The bst contains 7 nodes in total (expecting 7).  
~/Dropbox/Documents/Terms/2019-09 - Fall/CSCI255-Fall2019/Lab5
```

Figure 1: Console Output

## test\_bst.cpp

```

1  /* test_bst.cpp
2  * -----
3  * CSCI 255
4  * Lab 5:  Recursion, BST
5  * Authors: Darwin Jacob Groskleg, Man Lin
6  * Date:   Tuesday, October 8, 2019
7  *
8  * Purpose: Test the program and make sure that it works correctly.
9  *
10 * CONSOLE
11 * -----
12 * $ g++ -std=c++11 test_bst.cpp
13 * $ ./a.out
14 * found 10, as expected.
15 * cannot find 3, as expected.
16 * found 10, as expected.
17 * The bst contains 7 nodes in total (expecting 7).
18 * -----
19 */
20 #include <iostream>
21 #include "bst.hpp"
22
23 using namespace std;
24
25 int main() {
26     BST<int> bst;
27     bst.insert(5);
28     bst.insert(10);
29     bst.insert(2);
30     bst.insert(15);
31     bst.insert(7);
32     bst.insert(1);
33     bst.insert(4);
34
35     int toSearch;
36     BSTNode<int> *result;
37
38     // search 10 by iterative method
39     toSearch = 10;
40     result = bst.search(toSearch);
41     if (result != nullptr)
42         cout << "found " << result->key << ", as expected." << endl;
43     else
44         cout << "cannot find " << toSearch << endl;
45
46     // search 3 by recursive method
47     toSearch = 3;
48     result = bst.rSearch(toSearch);
49     if (result != nullptr)
50         cout << "found " << result->key << endl;
51     else
52         cout << "cannot find " << toSearch << ", as expected." << endl;
53
54     // search 10 by recursive method

```

```
55     toSearch = 10;
56     result = bst.rSearch(toSearch);
57     if (result != nullptr)
58         cout << "found " << result->key << ", as expected." << endl;
59     else
60         cout << "cannot find " << toSearch << endl;
61
62     // total # of nodes in bst
63     cout << "The bst contains " << bst.count() << " nodes in total"
64         << " (expecting 7).\n";
65 }
```

## bst.hpp

```

1  /* bst.hpp
2  * -----
3  * CSCI 255
4  * Lab 5:   Recursion, BST
5  * Authors: Darwin Jacob Groskleg, Man Lin
6  * Date:    Tuesday, October 8, 2019
7  */
8  #ifndef BST_HPP_INCLUDED
9  #define BST_HPP_INCLUDED
10
11 #if __cplusplus >= 202002L // if c++20
12 concept LessThanComparable<typename T> {
13     bool operator < (const T& lhs, const T& rhs);
14 }
15 #endif
16
17 // BSTNode class: representing A Node in a Binary Tree
18 template<class T>
19 class BSTNode {
20 public:
21     T key;
22     BSTNode<T> *left, *right;
23
24     BSTNode() {
25         left = right = nullptr;
26     }
27     BSTNode(const T& el, BSTNode *l = nullptr, BSTNode *r = nullptr) {
28         key = el; left = l; right = r;
29     }
30 };
31
32 // BST class: Binary Search Tree
33 //      DJ Groskleg (modified)
34 template<class T>
35     #if __cplusplus >= 202002L // if c++20
36     requires LessThanComparable<T>
37     #endif
38 class BST {
39     using node = BSTNode<T>;
40 public:
41     BST();
42
43     void insert(const T & el); //insert el to the BST
44
45     //iterative implementation of search
46     auto search(const T & el) -> node*;
47     //recursive search
48     auto rSearch(const T & el) -> node*;
49
50     int count(); //count the number of nodes in the BST
51
52 protected:
53     node *root;
54

```

```

55 private:
56     //recursive search helper method: starting from cur BSTNode
57     auto recursiveSearch(node *cur, const T &el) -> node*;
58
59     //helper method: count recursively from BSTNode r
60     int recursiveCount(node *r);
61 };
62
63
64 // DJ Groskleg (added)
65 template <class T>
66 int BST<T>::recursiveCount(node *r) {
67     if (r == nullptr) return 0;
68     return recursiveCount(r->left) + 1 + recursiveCount(r->right);
69 }
70
71 // DJ Groskleg (added)
72 template <class T>
73 int BST<T>::count() {
74     return recursiveCount(root);
75 }
76
77 // DJ Groskleg (added)
78 template <class T>
79 auto BST<T>::rSearch(const T &el) -> node* {
80     return recursiveSearch(root, el);
81 }
82
83 // DJ Groskleg (added)
84 template <class T>
85 auto BST<T>::recursiveSearch(node *cur, const T &el) -> node* {
86     if (cur == nullptr) return nullptr;
87     if (cur->key == el) return cur;
88
89     node* left = recursiveSearch(cur->left, el);
90     if (left != nullptr && left->key == el) return left;
91
92     node* right = recursiveSearch(cur->right, el);
93     if (right != nullptr && right->key == el) return right;
94
95     return nullptr;
96 }
97
98 // M.Lin (pre-existing)
99 template <class T>
100 BST<T>::BST() {
101     root = nullptr;
102 }
103
104 // Insert el to the BST
105 // M.Lin (pre-existing)
106 template <class T>
107 void BST<T>::insert(const T &el) {
108     node *p=nullptr, *prev=nullptr;
109     p = root;
110     while (p != nullptr){

```

```
111     prev = p;
112     if(p -> key > el)
113         p = p-> left;
114     else
115         p = p->right;
116 }
117 if (root == nullptr)
118     root = new node(el);
119 else {
120     if (el > prev->key)
121         prev->right = new node(el);
122     else
123         prev->left = new node(el);
124 }
125 }
126
127 // M.Lin (pre-existing)
128 template <class T>
129 auto BST<T>::search(const T & el) -> node* {
130     node * p = root;
131     while (p != 0){
132         if (p->key == el)
133             return p;
134         else{
135             if(p-> key < el)
136                 p = p-> right;
137             else
138                 p = p->left;
139         }
140     }
141     return 0;
142 }
143
144 #endif // BST_HPP_INCLUDED
```