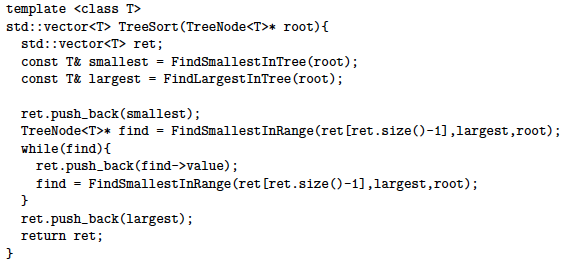
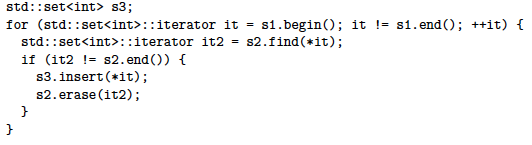
-Function returns a vector with the values of the tree in order from smallest to largest

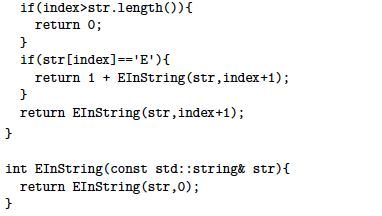


-Program creates a third set of integers containing all of the values from set1 that are also contained in set2

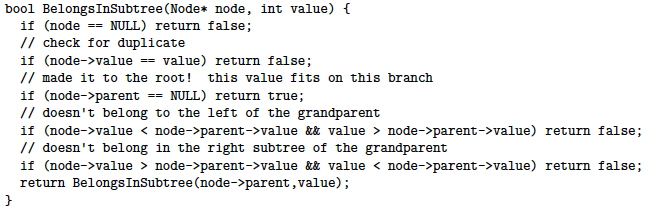


-Function counts how many times the letter ‘E’ shows up in a string

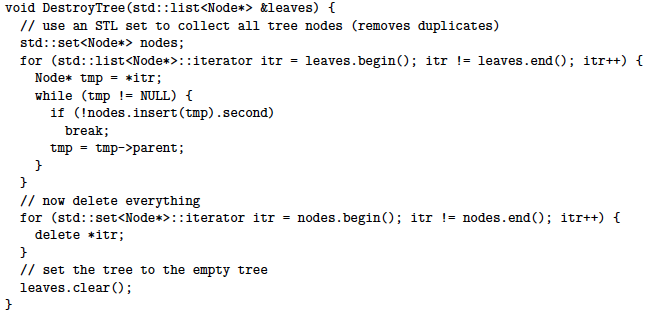




-Recursive function returns false if placing *value* within a subtree of *node* violates the BST property of the whole tree, and true otherwise



-Function clears all allocated memory associated with the upside-down tree





Depth First Traversals:

(a) Inorder (Left,Root,Right) : 4 2 5 1 3

(b) Preorder (Root,Left,Right) : 1 2 4 5 3

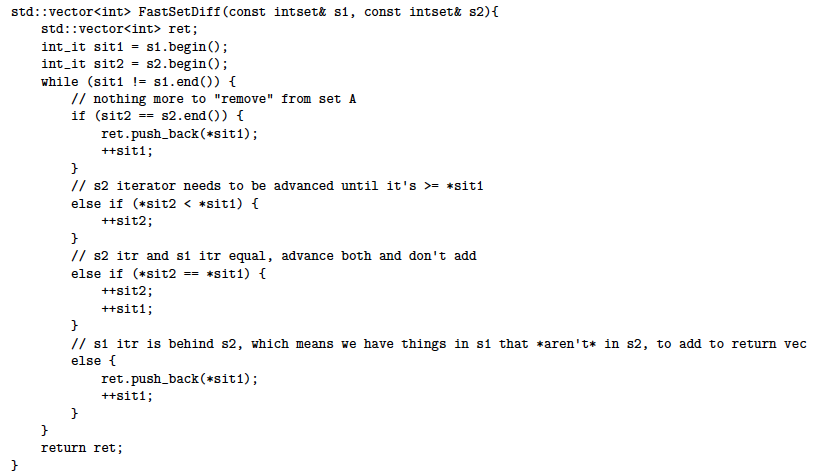
(c) Postorder (Left,Right,Root) : 4 5 2 3 1

(d) Breadth-First/Level Order : 1 2 3 4 5

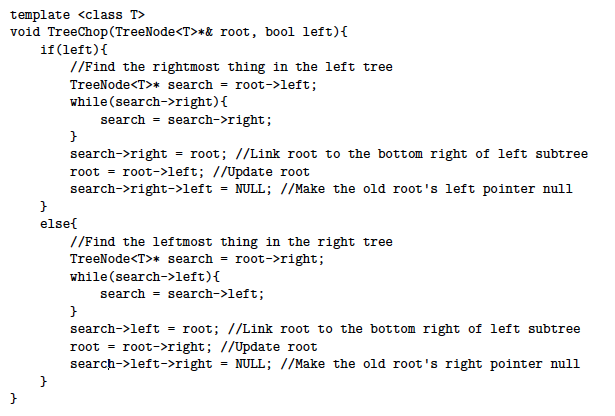
Time Complexity:

O(g(n)+n\*f(n))

-Function returns everything in set s1 that is not in set s2



-Function takes in a TreeNode\* pointing to the root of a BST, and a bool which is true if you want to chop the left side, and false if you want to chop the right



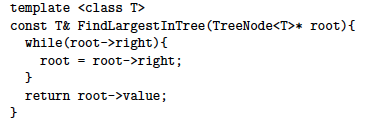
-std::map and std::set iterator’s operator++ use an in-order transversal to move through a tree

-ds\_set::find is an example of a depth-first search

-There exists a valid BST if [6,12,19,4,100,-100] is the post-order traversal of the tree

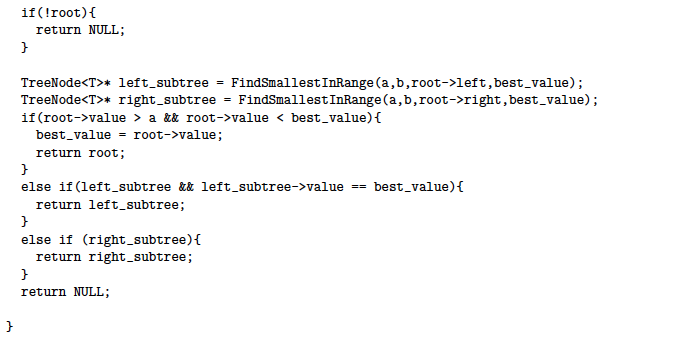
-If a tree is balanced, then the height will be O(logn). If it’s unbalanced, the height may be O(n).

-Function takes a TreeNode pointer to the root of a tree and returns the largest value in the tree



-Function finds the smallest value in the tree starting from root such that a<ret<b





**Lab Section 6**

**TA: Maurício**

**Mentors: John, Fred, Alec, Matt**

-the keys in the left subtree are less than the key in the parent node; **L < P**

- the keys in the right subtree are greater than the key in the parent node; **R > P**

Time Complexity:

O(nlog(n))

Time Complexity:

O(k(log(n)+log(k))

Time Complexity:

O(n)

Time Complexity:

O(h)

-Hash tables don’t store data in sorted order

-A hash table is implemented with an array at the top level

-A hash table has constant time access

-A hash function takes in 1 argument, and returns an integer index

-Hash function has a fast O(1) computation

-STL maps store pairs of “associated” values

-Map iterators refer to pairs

-Map search,insert,and erase:O(logn)

-Maps are ordered by increasing value of the key

-pairs are a templated struct with just 2 members

-Every leaf has the same depth in an exactly balanced tree

Abstract Representation of **orders**

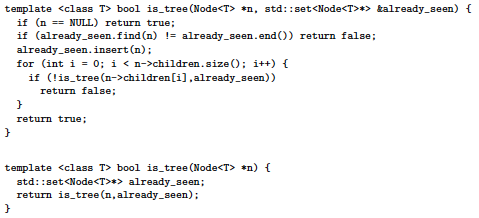
Time Complexity:

O(n\*i\*log(d))

Time Complexity:

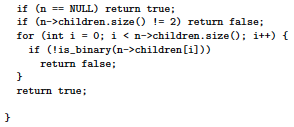
O(n+log(n))=>O(n)

-Function determines if a Node\* can be considered a binary tree



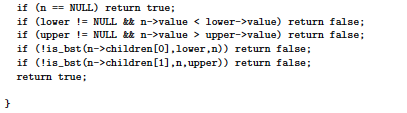
-Function determines if a Node\* can be considered a binary tree



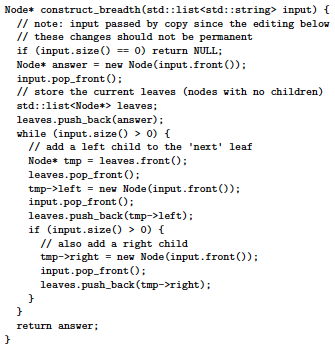


-Function determines if Node\* can be considered a BST

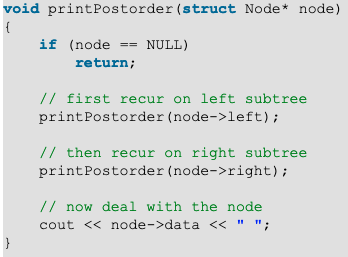




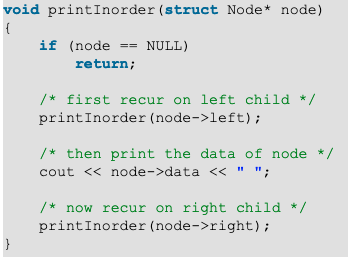
-Function takes in a list of strings and creates and returns a pointer to a well-balanced binary tree of Nodes with breadth-first traversal order that matches the input



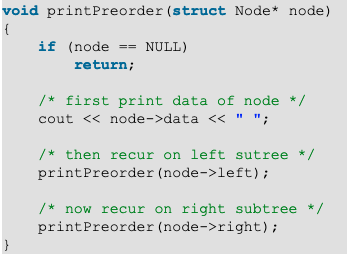
-Function prints nodes of binary tree in postorder traversal

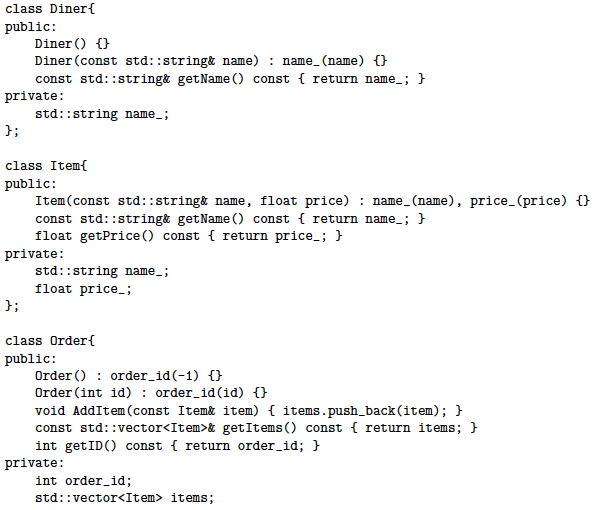


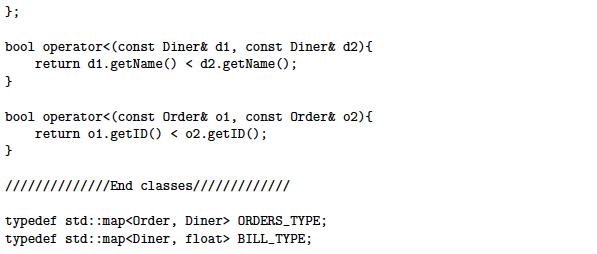
-Function prints nodes of binary tree in inorder traversal

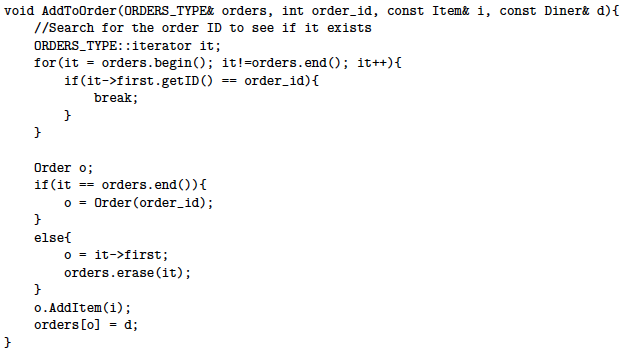


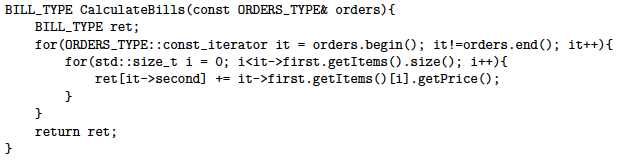
-Function prints nodes of binary tree in preorder traversal

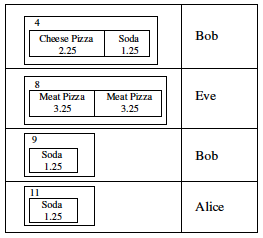












-Function determines if a Node\* can be considered a tree

