

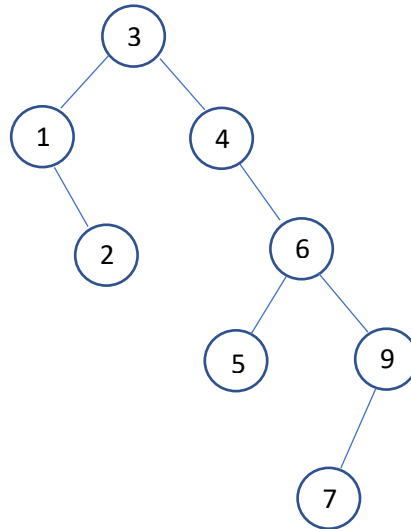
Assignment 3

Name: Michael Skarlatov

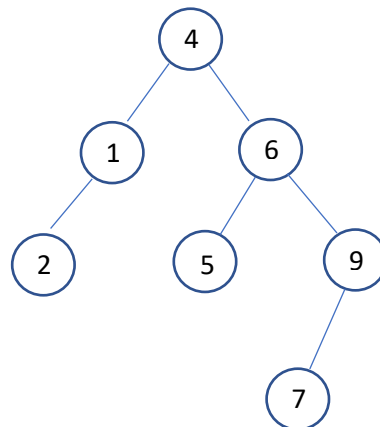
Date: 10/29/2018

Part 1:

In this part of the assignment I was given a list of numbers, 3, 1, 4, 6, 9, 2, 5, 7, of which I was supposed to put into a binary search tree and then show how this binary search tree would be structured. The structure of the tree would look like:



When deleting the root, the tree looks like:



Part 2:

This part of the assignment wanted me to create three functions that would take the pointer to a root of a binary search tree and compute different information about that tree.

- For the number of nodes, I made a recursive function that checked if the node being passed was a nullptr and if it was not the function add a one to the numberOfNodes count. It would then keep calling itself while moving left or right to cycle through the whole tree and count each node. This would eventually add up all the nodes and return the total number of nodes.
- For the number of leaves, I made a similar function to that of the number of nodes function but instead of just adding a one when ever the node was not a nullptr, I checked if the left and right nodes at that node were also nullptrs and incremented the count by one only if this was true.

- c) For the number of full nodes, I made an almost exact function to that of the number of leaves function but instead of checking if the left and right nodes were nullptrs I checked if they were not nullptrs. If the left and right nodes of the node being checked were not nullptrs then the count for number of full nodes would increment by one.

Part 3:

This assignment wanted me to answer how to implement the operation findKth without sacrificing the time bounds of any operation by modifying the binary tree. To do this, it would be best to for binary search tree, called T, to add a value called size to the nodes of the tree T which will indicate the size of the item at that node. By doing this the tree can maintain the information in same complexity as a usual binary search tree. This means when using the function findKth it can be done using the in-order traversal for checking the nodes, which won't sacrifice any time bonds.

Part 4:

This part of the assignment wanted me to write an implementation of iterators in the set class implemented as a binary search tree. To do this I made node struct with the usual node elements that are in a binary search tree but also with a parent pointer which allows the iterator to traverse the tree in a way that organizes it into a set. After creating the struct I made the setTree class which housed the primary functions, operators, and iterator creation function for the set. After which I made a class called Const_Iterator which had the general information for an iterator as well as the operators for the iterator that allowed it to check if the nodes that were being compared to that of the nodes in the iterator, where equal or needed to be added. Finally the main was used to insert a variety of values into the tree and then was printed out using the iterator to print out the set.

Input/Output Screen Shots:

Part 2:

<pre>Input numbers 7, 5, 9, 2, 1, 6, 4, 3 1 2 3 4 5 6 7 8 9 number of nodes 8 number of leaves 4 number of full nodes 3</pre>	<pre>Input numbers 3, 1, 4, 6, 9, 2, 5, 7 1 2 3 4 5 6 7 8 9 number of nodes 8 number of leaves 3 number of full nodes 2</pre>
---	---

The screen shot on the left is the building of binary search tree with the inputted numbers in order of 7, 5, 9, 2, 1, 6, 4, 3

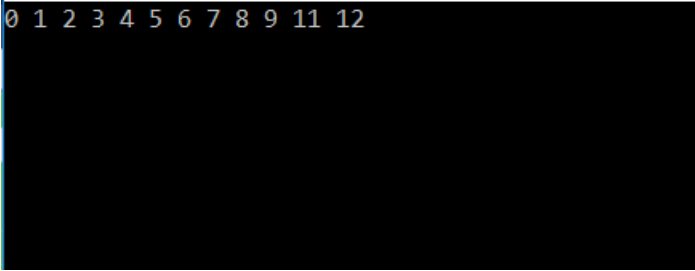
The screen shot on the right is the building of a binary search tree with the inputted numbers in order of

3, 1, 4, 6, 9, 2, 5, 7

The screen shot is the same as the tree from part 1 so in reference of what the tree looks like, refer back to the image in part 1.

Part 4:

```
235 //Stuff in the main to insert into the set and then print out to show the set is working
236 int main()
237 {
238     SetTree<int> set;
239     set.insert(4); set.insert(0);
240     set.insert(2); set.insert(7);
241     set.insert(0); set.insert(2);
242     set.insert(6); set.insert(4);
243     set.insert(12); set.insert(11);
244     set.insert(8); set.insert(1);
245     set.insert(5); set.insert(0);
246     set.insert(3); set.insert(2);
247     set.insert(7); set.insert(1);
248     set.insert(1); set.insert(1);
249     set.insert(9); set.insert(7);
250     set.insert(11); set.insert(1);
251
252     SetTree<int> set2 = set;
253
254     for (SetTree<int>::iterator it = set2.begin(); it != set2.end(); it++)
255     {
256         cout << *it << ' ';
257     }
258
259     return 0;
```



The screenshot shows the output of the program, which is a sequence of numbers: 0 1 2 3 4 5 6 7 8 9 11 12. The numbers are displayed in a monospaced font on a black background, with a white cursor at the end of the line.

These screenshots show a portion of my part4 program in the main where multiple numbers were inputted into the tree called set and then outputted using the iterator to move through the tree while ordering the values in the order of a set while making sure no values are repeated.