CS2040C Data Structure and Algorithm Assignment 3 BST (raded Assignment)

Your tasks in a summary, implement the followings:

1. In-order and post-order traversals
2. The height of each node and the size of the tree
3. Searching for minimum and maximum of a tree
4. Check if an element exist in the tree
5. The successor function
6. AVL tree balancing

__You can assume that a tree is not empty (size > 0) for Tasks 3 and 5__. Therefore, you code should work for the rest even if the tree has no node. First, please aim at finishing Tasks 1 to 4. Tasks 5 and 6 could be considered as more difficult.

Over all, you are allowed to (and you should) add more member variables and functions to the two given classes. However, remember to copy and paste the code for the class definitions also.

In order to avoid lengthy wording, "BST" stands for Binary Search Tree in this worksheet.

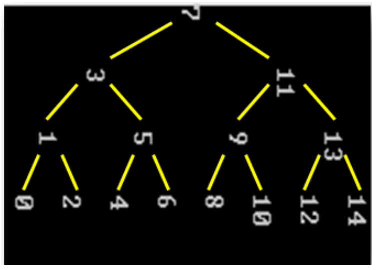
Skeleton Code

You are given three files with some basic implementations of a binary search tree:

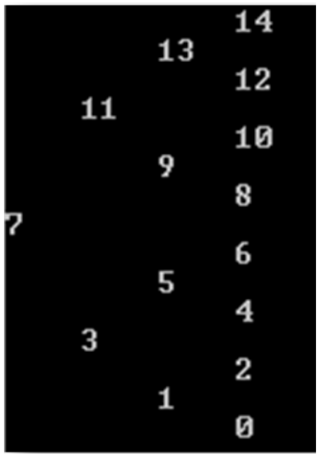
- `main.cpp, BST.h, BST.hpp`

The following functionalities are already implemented for you:

- Basic insertion without balancing
- Print out all the nodes according to pre-order traversal
- Print out the tree structure with a 90 degree anti-clockwise rotation



Print with 90º anti-clockwise rotation

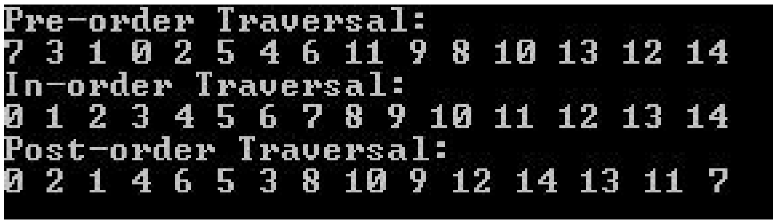


```
            14
        13
            12
    11
            10
        9
            8
7
            6
        5
            4
    3
            2
        1
            0
```

Task 1 In-order and Post-order Traversals

The pre-order traversal is implemented for you. Your job is to implement the in-order and post order traversal.

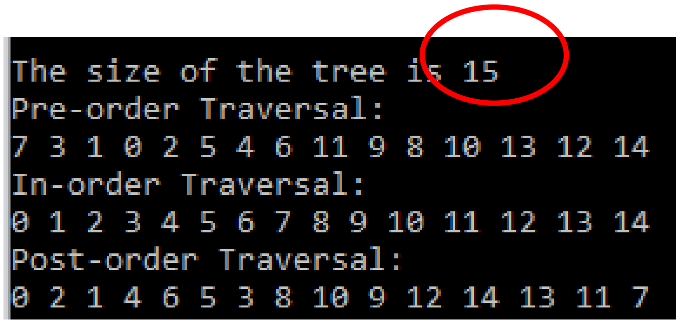
Sample output:



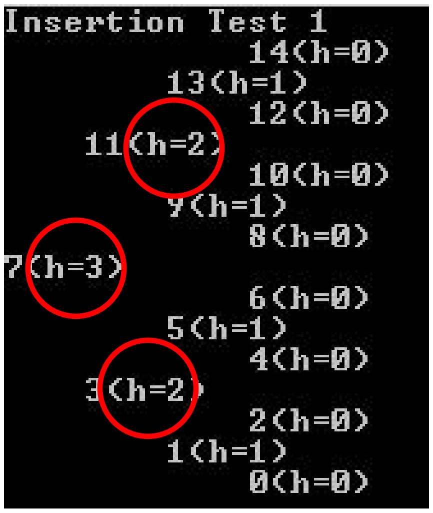
```
Pre-order Traversal:
7 3 1 0 2 5 4 6 11 9 8 10 13 12 14
In-order Traversal:
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14
Post-order Traversal:
0 2 1 4 6 5 3 8 10 9 12 14 13 11 7
```

Task 2 Size of the Tree and Height of Node

The size of the tree is the number of the elements being inserted into the tree. Currently, the **`_size`** member is zero and not set correctly. So does the height of each node (**`_height`**). Here is the correct sample input if you implemented them correctly.



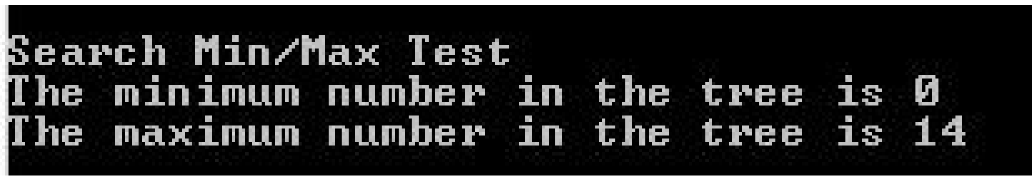
```
The size of the tree is 15
Pre-order Traversal:
7 3 1 0 2 5 4 6 11 9 8 10 13 12 14
In-order Traversal:
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14
Post-order Traversal:
0 2 1 4 6 5 3 8 10 9 12 14 13 11 7
```



```
Insertion Test 1
            14(h=0)
        13(h=1)
            12(h=0)
    11(h=2)
            10(h=0)
        9(h=1)
            8(h=0)
7(h=3)
            6(h=0)
        5(h=1)
            4(h=0)
    3(h=2)
            2(h=0)
        1(h=1)
            0(h=0)
```

Task 3 Search Min/Max

Uncomment `testSearchMinMax()` in `main()`. Implement the functions `searchMin()` and `searchMax()` in BST. Here is the sample output for the tree in the code:



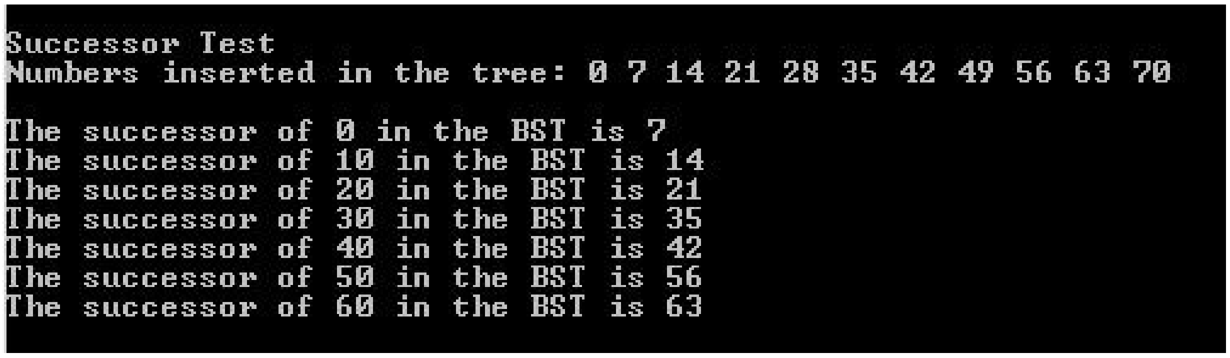
```
Search Min/Max Test
The minimum number in the tree is 0
The maximum number in the tree is 14
```

Task 4 Search for an Element in the Tree

Uncomment `testExist()` in `main()`. Similar to the Linked List assignment, implement a function `exist(x)` to return true if x is in the tree, and false otherwise.

Task 5 Successor

Uncomment `testSuccessor()` in `main()`. Implement the function `successor(x)` in BST such that it will return the successor of x:

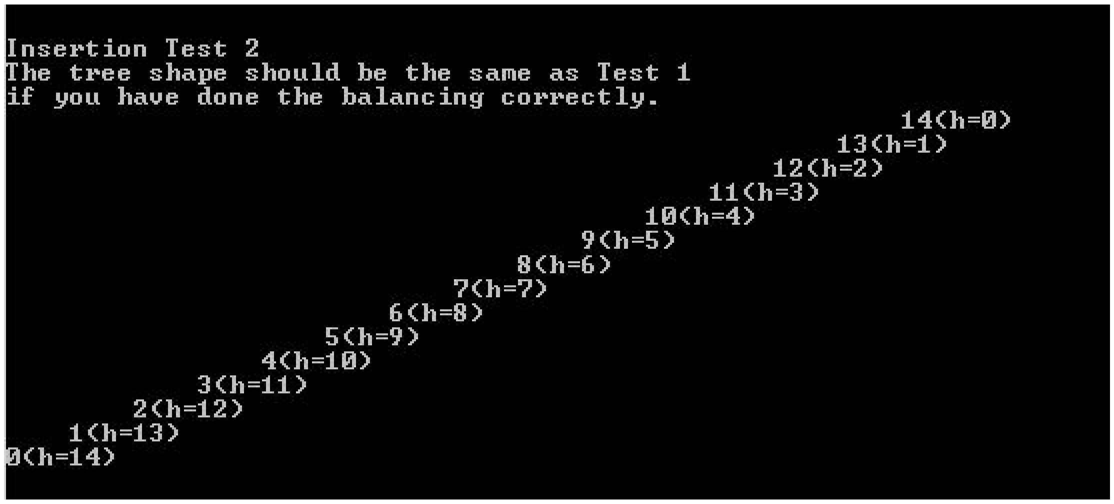


```
Successor Test
Numbers inserted in the tree: 0 7 14 21 28 35 42 49 56 63 70

The successor of 0 in the BST is 7
The successor of 10 in the BST is 14
The successor of 20 in the BST is 21
The successor of 30 in the BST is 35
The successor of 40 in the BST is 42
The successor of 50 in the BST is 56
The successor of 60 in the BST is 63
```

Task 6 AVL Tree Balancing

If you have reached this point, congratulation! However, here is the "final boss" you have to fight. Your tree should work fine so far for the above functionalities. **At this point, we highly recommend you to save and backup all your work so far**. Zip them up and copy them to a safe location.

Uncomment `testInsertion2 ()` in `main()`and you will get a skewed tree like this. This is because the balancing has not been implemented yet.



```
Insertion Test 2
The tree shape should be the same as Test 1
if you have done the balancing correctly.
                                                                14(h=0)
                                                            13(h=1)
                                                        12(h=2)
                                                    11(h=3)
                                                10(h=4)
                                            9(h=5)
                                        8(h=6)
                                    7(h=7)
                                6(h=8)
                            5(h=9)
                        4(h=10)
                    3(h=11)
                2(h=12)
            1(h=13)
0(h=14)
```

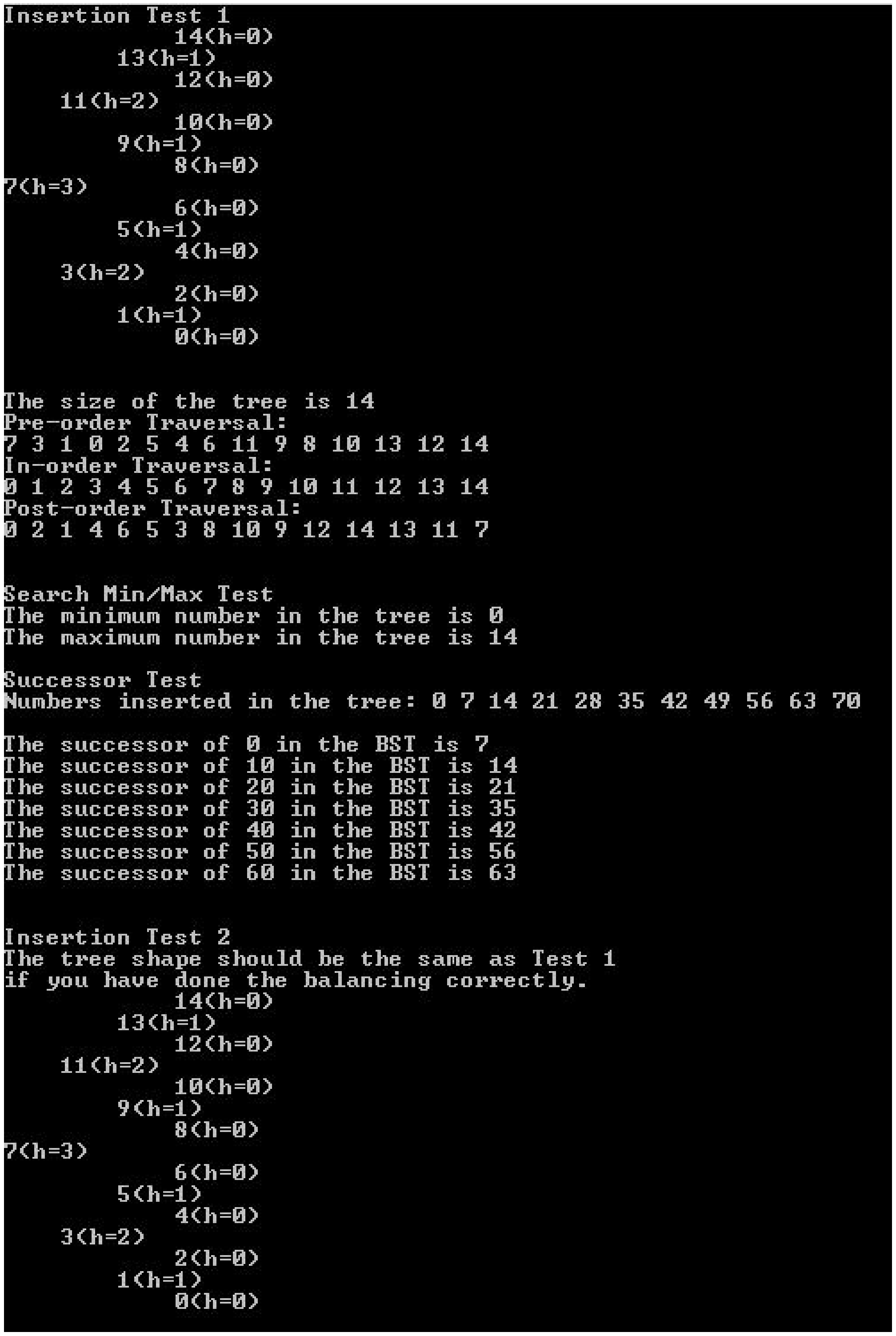
In order to balance the tree, you should

- Implement the left and right rotations (refer to the notes and Lab 4 slides)
- Add additional code in the insertion function to balance the tree after insertion.

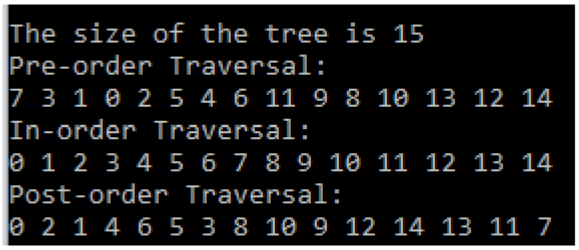
If you have implemented correctly, your tree in the "Insertion Test 2" should be the same as the one before.

Final Result

If you have implemented everything correctly, your final output should be like this



```
Insertion Test 1
            14(h=0)
        13(h=1)
            12(h=0)
    11(h=2)
            10(h=0)
        9(h=1)
            8(h=0)
7(h=3)
            6(h=0)
        5(h=1)
            4(h=0)
    3(h=2)
            2(h=0)
        1(h=1)
            0(h=0)


The size of the tree is 14
Pre-order Traversal:
7 3 1 0 2 5 4 6 11 9 8 10 13 12 14
In-order Traversal:
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14
Post-order Traversal:
0 2 1 4 6 5 3 8 10 9 12 14 13 11 7


Search Min/Max Test
The minimum number in the tree is 0
The maximum number in the tree is 14

Successor Test
Numbers inserted in the tree: 0 7 14 21 28 35 42 49 56 63 70

The successor of 0 in the BST is 7
The successor of 10 in the BST is 14
The successor of 20 in the BST is 21
The successor of 30 in the BST is 35
The successor of 40 in the BST is 42
The successor of 50 in the BST is 56
The successor of 60 in the BST is 63


Insertion Test 2
The tree shape should be the same as Test 1
if you have done the balancing correctly.
            14(h=0)
        13(h=1)
            12(h=0)
    11(h=2)
            10(h=0)
        9(h=1)
            8(h=0)
7(h=3)
            6(h=0)
        5(h=1)
            4(h=0)
    3(h=2)
            2(h=0)
        1(h=1)
            0(h=0)
```



```
The size of the tree is 15
Pre-order Traversal:
7 3 1 0 2 5 4 6 11 9 8 10 13 12 14
In-order Traversal:
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14
Post-order Traversal:
0 2 1 4 6 5 3 8 10 9 12 14 13 11 7
```