CSCI 200 - Fall 2023
Foundational Programming Concepts & Design

AXC - Forest Walk: Traversing BSTs

→This assignment is due by Thursday, December 07, 2023, 11:59 PM.←

→ As with all assignments, this must be an individual effort and cannot be pair programmed. Any debugging assistance must follow the course collaboration policy and be cited in the comment header block for the assignment.←

→ Do not forget to complete the following labs with this set: LXC ←

Jump To: **Rubric Submission**

For this assignment, we will verify our tree is created correctly by printing out the tree in different ways and some additional stats.

Tree Traversal

We'll expand our Tree class by adding the following public methods:

- `printInOrder()`: Recursively print the left subtree, then print the current node, then recursively print the right subtree.
- `printPreOrder()`: Print the current node, then recursively print the left subtree, then recursively print the right subtree.
- `printPostOrder()`: Recursively print the left subtree, then recursively print the right subtree, then print the current node.

To test your implementation, perform in `main.cpp` the following steps in order:

RMP-H

1. Create a BinarySearchTree of integers
2. Add the value 6
3. Add the value 5
4. Add the value 7
5. Add the value 1
6. Add the value 2
7. Add the value 9
8. Add the value 3
9. printInOrder()
10. printPreOrder()
11. printPostOrder()

Your program should display:

```
In Order:   1 2 3 5 6 7 9
Pre Order:  6 5 1 2 3 7 9
Post Order: 3 2 1 5 9 7 6
```

Create a second tree with the following steps:

1. Create a BinarySearchTree of integers
2. Add the value 5
3. Add the value 1
4. Add the value 9
5. Add the value 7
6. Add the value 6
7. Add the value 3
8. Add the value 2
9. printInOrder()
10. printPreOrder()
11. printPostOrder()

Your program should display:

```
In Order:   1 2 3 5 6 7 9
Pre Order:  5 1 3 2 9 7 6
Post Order: 2 3 1 6 7 9 5
```

We'll now add two more ways to traverse the tree:

- `printBreadthOrder()` : Print the tree level by level using a breadth first search.

- `printDepthOrder()` : Print the tree branch by branch using a depth first search.

Your first tree should display:

```
In Order:      1 2 3 5 6 7 9
Pre Order:     6 5 1 2 3 7 9
Post Order:    3 2 1 5 9 7 6
Breadth Order: 6 5 7 1 9 2 3
Depth Order:   6 5 1 2 3 7 9
```

Your second tree should display:

```
In Order:      1 2 3 5 6 7 9
Pre Order:     5 1 3 2 9 7 6
Post Order:    2 3 1 6 7 9 5
Breadth Order: 5 1 9 3 7 2 6
Depth Order:   5 1 3 2 9 7 6
```

We'll now add a way to pretty print by level:

- `printByLevels()` : Print the tree level by level, each on their own line.

Your first tree should display:

```
In Order:      1 2 3 5 6 7 9
Pre Order:     6 5 1 2 3 7 9
Post Order:    3 2 1 5 9 7 6
Breadth Order: 6 5 7 1 9 2 3
Depth Order:   6 5 1 2 3 7 9
By Levels:
Level 1: 6
Level 2: 5 7
Level 3: 1 9
Level 4: 2
Level 5: 3
```

Your second tree should display:

```
In Order:      1 2 3 5 6 7 9
Pre Order:     5 1 3 2 9 7 6
Post Order:    2 3 1 6 7 9 5
Breadth Order: 5 1 9 3 7 2 6
Depth Order:   5 1 3 2 9 7 6
By Levels:
```

```
Level 1: 5
Level 2: 1 9
Level 3: 3 7
Level 4: 2 6
```

Finally, add a method to print the height, or number of levels, of the tree.

- `height()` : Print the number of levels in the tree.

Your first tree should display:

```
In Order:      1 2 3 5 6 7 9
Pre Order:     6 5 1 2 3 7 9
Post Order:    3 2 1 5 9 7 6
Breadth Order: 6 5 7 1 9 2 3
Depth Order:   6 5 1 2 3 7 9
By Levels:
Level 1: 6
Level 2: 5 7
Level 3: 1 9
Level 4: 2
Level 5: 3
Height: 5
```

Your second tree should display:

```
In Order:      1 2 3 5 6 7 9
Pre Order:     5 1 3 2 9 7 6
Post Order:    2 3 1 6 7 9 5
Breadth Order: 5 1 9 3 7 2 6
Depth Order:   5 1 3 2 9 7 6
By Levels:
Level 1: 5
Level 2: 1 9
Level 3: 3 7
Level 4: 2 6
Height: 4
```

For a final test, create a third tree as follows:

1. Create a BinarySearchTree of integers
2. Add the value 5
3. Add the value 2
4. Add the value 1
5. Add the value 7

6. Add the value 9
7. Add the value 6
8. Add the value 3

This tree will print:

```
In Order:      1 2 3 5 6 7 9
Pre Order:     5 2 1 3 7 6 9
Post Order:    1 3 2 6 9 7 5
Breadth Order: 5 2 7 1 3 6 9
Depth Order:   5 2 1 3 7 6 9
By Levels:
Level 1: 5
Level 2: 2 7
Level 3: 1 3 6 9
Height: 3
```

Grading Rubric

Your submission will be graded according to the following rubric:

Points	Requirement Description
0.5	Submitted correctly by Thursday, December 07, 2023, 11:59 PM
0.5	Project builds without errors nor warnings.
2.0	**Best Practices** and **Style Guide** followed.
0.5	Program follows specified user I/O flow.
0.5	Public and private tests successfully passed.
2.0	Fully meets specifications.
6.00	**Total Points**

Submission

Always, **always**, **ALWAYS** update the header comments at the top of your main.cpp file. And if you ever get stuck, remember that there is LOTS of **help** available.

Zip together your `BinarySearchTree.hpp`, `main.cpp`, `Makefile` files and name the zip file `AXC.zip` . Upload this zip file to Canvas under AXC.

→This assignment is due by Thursday, December 07, 2023, 11:59 PM.←

→ As with all assignments, this must be an individual effort and cannot be pair programmed. Any debugging assistance must follow the course collaboration policy and be cited in the comment header block for the assignment.←

→ Do not forget to complete the following labs with this set: LXC ←

Last Updated: 11/09/22 21:02

Any questions, comments, corrections, or request for use please contact jpaone {at} mines {dot} edu.

[Jump to Top] [Site Map]