EECS 560 Lab 6 – Implementation of Binary Search Trees

Prof.: Dr.Shontz, GTAs: Chiranjeevi Pippalla, Prashanthi Mallojula

Maximum possible points: 100

Due date:

11:59pm, Monday, 03/16/2020 for Tuesday labs.

11:59pm, Wednesday, 03/18/2020 for Thursday labs.

Purpose:

The purpose of this lab is to implement a Binary Search Tree (BST) in C++.

General Requirements:

In this lab, you are required to implement a binary search tree using a pointer-based implementation (do not use an array). Each node of the tree will have a key and left and right pointers, where the left pointer will point to its left child, and the right pointer will point to its right child. You are to read in a collection of integers from a data file called data.txt and insert them into the BST based on the BST property. **Duplicate keys are allowed in this lab, and your input data.txt file should have at least one duplicate value.** You may hard code the file name if you wish.

The binary search tree should be implemented with an appropriate constructor and destructor. The rest of the methods should be implemented as follows:

- **AddItem(x)** – This function inserts the integer element into the tree. If there is a duplicate integer as the input, it should go to the right subtree.
- **DeleteItem(x)** – This function deletes the integer from the binary search tree. The function should delete the first occurrence of the integer only. In order to delete the integer, the tree must be traversed in order to find the integer. This should be done using the < or >= property. Once the node is located, it is deleted. (Note: This will be explained in more detail in lab.)
- **InorderSuccessor()** – This function should return the inorder successor of the given element.
- **LevelOrder() –** This function should print the elements in the tree in level order fashion.
- **SpiralLevelOrder()** – This function should print the elements in the tree in a spiral order fashion, i.e., the alternating levels from the root should be printed in reverse order. The even level elements should be printed in the forward order (from left to right), and the odd level elements should be printed in the reverse order (right to left).
- **LeftSideView() –** This function should print the visible elements of the tree when looking at it from its left side. The elements should be printed from top to bottom in a list format.
- **RightSideView() –** This function should print the visible elements of the tree when looking at it from its right side. The elements should be printed from top to bottom in a list format.
- **KthSmallestItem() –** This function should return the kth smallest integer from the tree. If the tree is empty, it should return null. The value of k must be a valid integer in the range from 1 to the number of elements in the tree. For example, in the list [10, 20, 20, 30], the third smallest

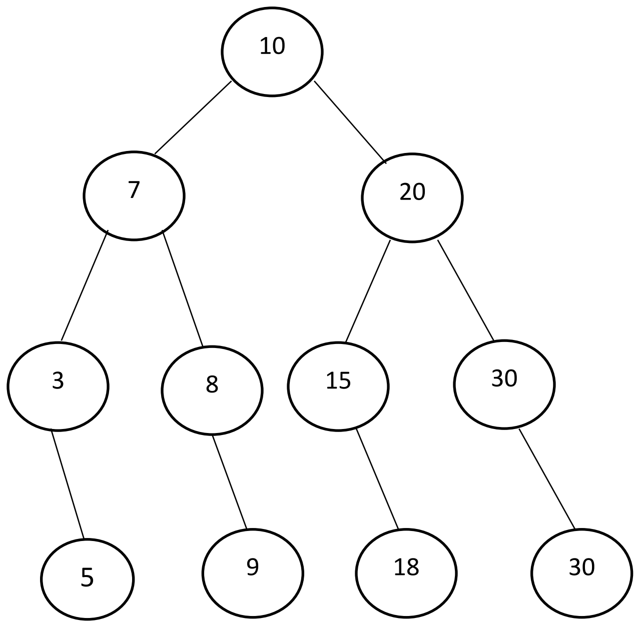
integer is 30 as opposed to 20. You should consider the value of the integer and not the position of the integer in the list.

- **Exit() –** This function should exit the program.

Expected Results:

data.txt: 10 7 20 3 8 15 30 5 9 18 30

We will insert these keys, in the given order, into an initially empty BST.



Please note that you are not expected to show the tree graphically in your output. This is just for your reference.

Now that you have built the BST, these are the expected results of performing the various options on the BST.

____

Please choose one of the following commands:

1. AddItem
2. DeleteItem
3. InorderSuccessor
4. LevelOrder
5. SpiralLevelOrder
6. LeftSideView
7. RightSideView
8. KthSmallestItem

9. Exit

> 1

> Enter the element to be added: 2

> Element 2 was successfully added.

Please choose one of the following commands:

1. AddItem
2. DeleteItem
3. InorderSuccessor
4. LevelOrder
5. SpiralLevelOrder
6. LeftSideView
7. RightSideView
8. KthSmallestItem
9. Exit

> 2

> Enter the element to be deleted: 8

> Element 8 was successfully deleted.

Please choose one of the following commands:

1. AddItem
2. DeleteItem
3. InorderSuccessor
4. LevelOrder
5. SpiralLevelOrder
6. LeftSideView
7. RightSideView
8. KthSmallestItem
9. Exit

> 3

> Enter the element to which you want to know the inorder successor: 9

> The inorder successor of 9 is 10.

EECS 560 Lab 6 – Implementation of Binary Search Trees
Prof.: Dr.Shontz, GTAs: Chiranjeevi Pippalla, Prashanthi Mallojula

Please choose one of the following commands:

1. AddItem
2. DeleteItem
3. InorderSuccessor
4. LevelOrder
5. SpiralLevelOrder
6. LeftSideView
7. RightSideView
8. KthSmallestItem
9. Exit

>3

> Enter the element to which you want to know the inorder successor: 3

> The inorder successor of 3 is 5.

Please choose one of the following commands:

1. AddItem
2. DeleteItem
3. InorderSuccessor
4. LevelOrder
5. SpiralLevelOrder
6. LeftSideView
7. RightSideView
8. KthSmallestItem
9. Exit

>4

> Level Order: 10, 7, 20, 3, 9, 15, 30, 2, 5, 18, 30

Please choose one of the following commands:

1. AddItem
2. DeleteItem
3. InorderSuccessor
4. LevelOrder
5. SpiralLevelOrder
6. LeftSideView
7. RightSideView
8. KthSmallestItem
9. Exit

> 5

> Spiral Level Order: 10, 20, 7, 3, 9, 15, 30, 30, 18, 5, 2

Please choose one of the following commands:

1. AddItem
2. DeleteItem
3. InorderSuccessor
4. LevelOrder
5. SpiralLevelOrder
6. LeftSideView
7. RightSideView
8. KthSmallestItem
9. Exit

> 6

> Left SideView: 10, 7, 3, 2

Please choose one of the following commands:

1. AddItem
2. DeleteItem
3. InorderSuccessor
4. LevelOrder
5. SpiralLevelOrder
6. LeftSideView
7. RightSideView
8. KthSmallestItem
9. Exit

> 7

> Right Side View: 10, 20, 30, 30

Please choose one of the following commands:

1. AddItem
2. DeleteItem
3. InorderSuccessor
4. LevelOrder
5. SpiralLevelOrder
6. LeftSideView
7. RightSideView

8. KthSmallestItem
9. Exit

```
>8

> Enter the value of k: 3

> 5
```

Please choose one of the following commands:

1. AddItem
2. DeleteItem
3. InorderSuccessor
4. LevelOrder
5. SpiralLevelOrder
6. LeftSideView
7. RightSideView
8. KthSmallestItem
9. Exit

```
>9
> Done.
```

Submission:

Follow the conventions below to facilitate grading:

Grading rubric:

- Full grade: The program should execute without any issues with all the options executed and with no memory leaks.
- Points will be taken off for execution errors, such as memory leaks, segmentation/program abort issues and missing handling of invalid cases.
- Programs that are compiled but do not execute will earn in the range of 0 to 50% of the possible points. Your grade will be determined based on the program design and the options implemented in the code.

Submission instructions:

- All files, i.e., the source files and Makefile, should be zipped in a folder.
- Include a ReadMe.txt if your code requires any special instructions to run.
- The naming convention of the folder should be LastName_Lab6.zip (or .tar or .rar or .gz).
- Email it to your respective lab instructor: chiranjeevi.pippalla@ku.edu (Chiru) or prashanthi.mallojula@ku.edu (Prashanthi) with subject line EECS 560 Lab6.
- Your program should compile and run on the **Linux machines** in **Eaton 1005D using g++.**