Maximum possible points: 100

Due date:
11:59pm, Sunday 10/06/2019 for Monday labs.
11:59pm, Tuesday 10/08/2019 for Wednesday labs.
11:59pm, Thursday 10/10/2019 for Friday 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 implementation only (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 characters from a data file called data.txt in level order only. Duplicate keys are allowed in this lab, and your input data.txt file should have at least one duplicated 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 character element into the tree. The correct position of the character should be per the ASCII value of the character. If there is a duplicate character as the input, it should go to the right sub tree.
- **delete(x)** – This function deletes the character from the binary search tree. In order to delete the character, the tree must be traversed in order to find the character. 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.)
- **Leaf(x)** – This function will check whether or not x has a child. If x does not have a child, then it is a leaf node, and in that case the function should return true. The output should be: x is/isn’t a leaf node.
- **printLeaf()** – This function prints the leaf nodes from left to right.
- **treeHeight()** – Prints the height of the tree.
- **preorder()** – Traverses the tree in pre-order and prints all the characters.
- **postorder()** – Traverses the tree in post-order and prints all the characters.
- **inorder()** – Traverses the tree in in-order and prints all the characters.
- **searchElement()** – This function will search for the character. If it finds the character, it should return true. Otherwise, it should return false.
Expected Results:
data.txt: f c a i d g j e b h

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

Please note that you are not expected to show 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. Delete
3. Leaf
4. PrintLeaf
5. TreeHeight
6. Preorder
7. Postorder
8. Inorder
9. LevelOrder
10. SearchItem
11. Exit

> 6
> Output: Preorder Traversal - f c a b d e i g h j

Please choose one of the following commands:

1. AddItem
2. Delete
3. Leaf
4. PrintLeaf
5. TreeHeight
6. Preorder
7. Postorder
8. Inorder
9. LevelOrder
10. SearchItem
11. Exit

> 8
> Output: Inorder Traversal – a b c d e f g h i j

Please choose one of the following commands:

1. AddItem
2. Delete
3. Leaf
4. PrintLeaf
5. TreeHeight
6. Preorder
7. Postorder
8. Inorder
9. LevelOrder
10. SearchItem
11. Exit

> 7
> Output: Postorder Traversal – b a e d c h g j i f
Please choose one of the following commands:

1. AddItem
2. Delete
3. Leaf
4. PrintLeaf
5. TreeHeight
6. Preorder
7. Postorder
8. Inorder
9. LevelOrder
10. SearchItem
11. Exit

>9

> Output: Levelorder Traversal - f c i a d g j b e h

Please choose one of the following commands:

1. AddItem
2. Delete
3. Leaf
4. PrintLeaf
5. TreeHeight
6. Preorder
7. Postorder
8. Inorder
9. LevelOrder
10. SearchItem
11. Exit

>1

> Enter the character to be inserted:

> k

> Output: The element was inserted successfully.
Please choose one of the following commands:

1. AddItem
2. Delete
3. Leaf
4. PrintLeaf
5. TreeHeight
6. Preorder
7. Postorder
8. Inorder
9. LevelOrder
10. SearchItem
11. Exit

> 6

> Output: Preorder traversal - f c a b d e i g h j k

Please choose one of the following commands:

1. AddItem
2. Delete
3. Leaf
4. PrintLeaf
5. TreeHeight
6. Preorder
7. Postorder
8. Inorder
9. LevelOrder
10. SearchItem
11. Exit

> 4
> Output: The leaves are: b e h k

Please choose one of the following commands:

1. AddItem
2. Delete
3. Leaf
4. PrintLeaf
5. TreeHeight
6. Preorder
7. Postorder
8. Inorder
9. LevelOrder
10. SearchItem
11. Exit

> 5
> Output: The height of the tree is 3.

Please choose one of the following commands:

1. AddItem
2. Delete
3. Leaf
4. PrintLeaf
5. TreeHeight
6. Preorder
7. Postorder
8. Inorder
9. LevelOrder
10. SearchItem
11. Exit

>8

> Output: Inorder Traversal - a b c d e f g h i j k

------------------------------------------------------------

Please choose one of the following commands:

1. AddItem
2. Delete
3. Leaf
4. PrintLeaf
5. TreeHeight
6. Preorder
7. Postorder
8. Inorder
9. LevelOrder
10. SearchItem
11. Exit

>9

> Output: Levelorder Traversal - f c i a d g j b e h k

------------------------------------------------------------

Please choose one of the following commands:

1. AddItem
2. Delete
3. Leaf
4. PrintLeaf
5. TreeHeight
6. Preorder
7. Postorder
8. Inorder
9. LevelOrder
10. SearchItem
11. Exit

>2

>Enter the character to be deleted:

> z

> Output: Deletion failed. The character is not present in tree
Please choose one of the following commands:

1. AddItem
2. Delete
3. Leaf
4. PrintLeaf
5. TreeHeight
6. Preorder
7. Postorder
8. Inorder
9. LevelOrder
10. SearchItem
11. Exit

> 2

> Enter a character to be deleted:

> b

> Output: Deletion was successful.

Please choose one of the following commands:

1. AddItem
2. Delete
3. Leaf
4. PrintLeaf
5. TreeHeight
6. Preorder
7. Postorder
8. Inorder
9. LevelOrder
10. SearchItem
11. Exit

> 2

> Enter a character to be deleted:

> c

> Output: Deletion was successful.
3. Leaf
4. PrintLeaf
5. TreeHeight
6. Preorder
7. Postorder
8. Inorder
9. LevelOrder
10. SearchItem
11. Exit

>2

Enter a character to be deleted:

> f

Deletion was successful.

Please choose one of the following commands:

1. AddItem
2. Delete
3. Leaf
4. PrintLeaf
5. TreeHeight
6. Preorder
7. Postorder
8. Inorder
9. LevelOrder
10. SearchItem
11. Exit

> 3

> Please enter the value which you want to test as a leaf node.

> k

> Output: k is a leaf node.

Please choose one of the following commands:

1. AddItem
2. Delete
3. Leaf
4. PrintLeaf
5. TreeHeight
6. Preorder
7. Postorder
8. Inorder
9. LevelOrder
10. SearchItem
11. Exit

> 10
> Enter the character to be found.
> x
> Output: The character is not present in the tree.

Please choose one of the following commands:

1. AddItem
2. Delete
3. Leaf
4. PrintLeaf
5. TreeHeight
6. Preorder
7. Postorder
8. Inorder
9. LevelOrder
10. SearchItem
11. Exit

> 10
> Enter character to be found.
> j
> Output: The character is present in the tree.

Please choose one of the following commands:

1. AddItem
2. Delete
3. Leaf
4. PrintLeaf
5. TreeHeight
EECS 560 Lab 5 – Implementation of Binary Search Tree
Prof.: Dr. Shontz, GTAs: Chiranjeevi Pippalla, Anubhav Ghosh

6. Preorder
7. Postorder
8. Inorder
9. LevelOrder
10. SearchItem
11. Exit

>10

>Output: Done!

Submission:
Follow the conventions below to facilitate grading:

Source Code

Place all your source files (*.cpp, *.hpp) and input files in a single folder with no subfolders. Name your folder using the convention Lastname_LabX (e.g., Smith_Lab05). Include a functioning Makefile inside the folder. (The makefile should also include the clean command.) Verify that your code runs on the Linux machines in the lab.

Compressed File

Compress using .zip, .rar, or .tar.gz. Name your file using the convention Lastname_LabX (e.g., Smith_Lab05.zip).

Email

Use the following subject for your email: Lastname_LabX (e.g., Smith_Lab05). Send your code to anubhav@ku.edu if you are in one of Anubhav’s sections or, to chiranjeevi.pippalla@ku.edu if you are in one of Chiranjeevi’s sections. Anytime you have a question about the lab, put the word question in the subject of the email.