100 Points

Due date:
11:59pm, Sunday 09/29/2019 for Monday lab.
11:59pm, Tuesday 10/01/2019 for Wednesday lab.
11:59pm, Thursday 10/03/2019 for Friday lab.

Purpose:
The purpose of this lab is to implement a binary tree in C++.

General Requirements:
In this lab, you are required to implement a binary 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 numbers from a data file called data.txt in level-order only. Duplicate keys are not allowed in the binary tree. You may hard code the file name if you wish.
The binary tree should be implemented with an appropriate constructor and destructor. The rest of the methods should be implemented as follows:

- **IsFull()** – This function should return true if the tree is full. Only Print ‘True’ or ‘False’ in the output. To check the correctness of the function, test this function twice, i.e., once after reading in the values and constructing the tree from data.txt and again after adding a user typed value using AddItem(x) when the tree will no longer be full.

- **AddItem(x)** – This function adds an element to the binary tree in a level-order fashion. Every new element should be added in the next position. Consider a constructed binary tree from the given data.txt file.

- **Delete()** – This function deletes the element that occurs at the last position according to a level order traversal in the binary tree and returns the value that it has deleted.

- **Leaf(x)** – This function will check whether or not the node containing x has a child. If it does not have a child, the node is a leaf node. In that case, the function should return true. The output should be: The node containing x is/isn’t a leaf node.

- **PrintLeaf()** – This function prints leaf nodes one after the other. The order of traversal/printing should be in Inorder only.

- **PrintTreeHeight()** – Prints the height of the tree.

- **Preorder()** – Traverses the tree in preorder and prints all of the elements.

- **Postorder()** – Traverses the tree in post-order and prints all of the elements.

- **Inorder()** – Traverses the tree in in-order and prints all of the elements.

- **Levelorder()** – Traverses the tree in level order and prints all of the elements.

![Fig. 3](image1)

![Fig. 4](image2)
Examples for reference:
These are some examples related to the operations mentioned above:

- **IsFull():** The tree shown in Fig. 1 is a full binary tree, whereas the tree shown in Fig. 2 is not a full binary tree.
- **AddItem(x):** The binary tree should look like Fig. 1, and adding an element (say 24) will change the structure of the binary tree to Fig. 2.
- **Delete():** The function if called on the tree shown in Fig. 2 should alter the state of the tree to Fig. 3 and should return 24. Call the function once more and then the state of Fig. 3 will change to Fig. 4.

**Expected Results:**
data.txt: 20 8 22 4 12 10 14

After building the tree by reading the data.txt in level order, your program should ask the user to choose one of the options below:

*Please note that the outputs shown below are only for your reference. The actual outputs may differ.*

```
Please choose one of the following commands:
1- IsFull
2- AddItem
3- Delete
4- Leaf
5- PrintLeaf
6- PrintTreeHeight
7- Preorder
8- Postorder
9- Inorder
10- Levelorder
11- Exit
>1
>Output: True
```

```
Please choose one of the following commands:
1- IsFull
2- AddItem
3- Delete
4- Leaf
5- PrintLeaf
6- PrintTreeHeight
7- Preorder
```
8- Postorder
9- Inorder
10- Levelorder
11- Exit
>8
> Output: Printing in Post-Order - 4 12 8 10 14 22 20

Please choose one of the following commands:
1- IsFull
2- AddItem
3- Delete
4- Leaf
5- PrintLeaf
6- PrintTreeHeight
7- Preorder
8- Postorder
9- Inorder
10- Levelorder
11- Exit
>2
> Please enter the value which you want to enter into the tree.
>24
> Output: Value inserted successfully!

Please choose one of the following commands:
1- IsFull
2- AddItem
3- Delete
4- Leaf
5- PrintLeaf
6- PrintTreeHeight
7- Preorder
8- Postorder
9- Inorder
10- Levelorder
11- Exit
>10
> Output: Printing in Level-Order – 20 8 22 4 12 10 14 24

Please choose one of the following commands:
1- IsFull
2- AddItem
3- Delete
4- Leaf  
5- PrintLeaf  
6- PrintTreeHeight  
7- Preorder  
8- Postorder  
9- Inorder  
10- Levelorder  
11- Exit  

>4  
> Please enter the value which you want to test as a leaf node.  
> 10  
> Output: The node with 10 is a leaf node.  

Please choose one of the following commands:

1- IsFull  
2- AddItem  
3- Delete  
4- Leaf  
5- PrintLeaf  
6- PrintTreeHeight  
7- Preorder  
8- Postorder  
9- Inorder  
10- Levelorder  
11- Exit  

>3  
> The last value is 24, and it has been deleted.  

Please choose one of the following commands:

1- IsFull  
2- AddItem  
3- Delete  
4- Leaf  
5- PrintLeaf  
6- PrintTreeHeight  
7- Preorder  
8- Postorder  
9- Inorder  
10- Levelorder  
11- Exit  

>5
The leaf nodes are: 4 12 10 14.

Please choose one of the following commands:
1- IsFull
2- AddItem
3- Delete
4- Leaf
5- PrintLeaf
6- PrintTreeHeight
7- Preorder
8- Postorder
9- Inorder
10- Levelorder
11- Exit

>6
> Output: The height of the tree is 2.

Please choose one of the following commands:
1- IsFull
2- AddItem
3- Delete
4- Leaf
5- PrintLeaf
6- PrintTreeHeight
7- Preorder
8- Postorder
9- Inorder
10- Levelorder
11- Exit

>7
> Output: Printing the tree in Pre-Order- 20 8 4 12 22 10 14

Please choose one of the following commands:
1- IsFull
2- AddItem
3- Delete
4- Leaf
5- PrintLeaf
6- PrintTreeHeight
7- Preorder
8- Postorder
9- Inorder
10- LevelOrder
11- Exit
>9
> Output: Printing the tree in In-Order - 4 8 12 20 10 22 14

------------------------------------------------------------
Please choose one of the following commands:
12- IsFull
13- AddItem
14- Delete
15- Leaf
16- PrintLeaf
17- TreeHeight
18- Preorder
19- Postorder
20- Inorder
21- LevelOrder
22- Exit
>11
> Output: Goodbye!

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_Lab04).
• 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_Lab04.zip).

Email
• Use the following subject for your email: Lastname_LabX (e.g., Smith_Lab04).
• Send you 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.