KATHMANDU UNIVERSITY

DHULIKHEL, KAVRE

School of Engineering

Lab Report No.: 04

Data Structures and Algorithm (COMP 202)

Submitted By:

Aayush Pokhrel

Roll No:42

Group: CE

Level: 2nd Year/1st Sem

Submitted To

Dr. Rajani Chulyadyo

Department of Computer Science And Engineering

Objective: Implemention of Binary Search Tree with the following operations:

Using:

(a) isEmpty(): Returns true if the tree is empty, and false otherwise

(b) addBST(key, value): Inserts an element to the BST

(c) removeBST(keyToDelete): Removes the node with the given key from the BST

(d) searchBST(targetKey): Returns true if the key exists in the tree, and false otherwise

Description

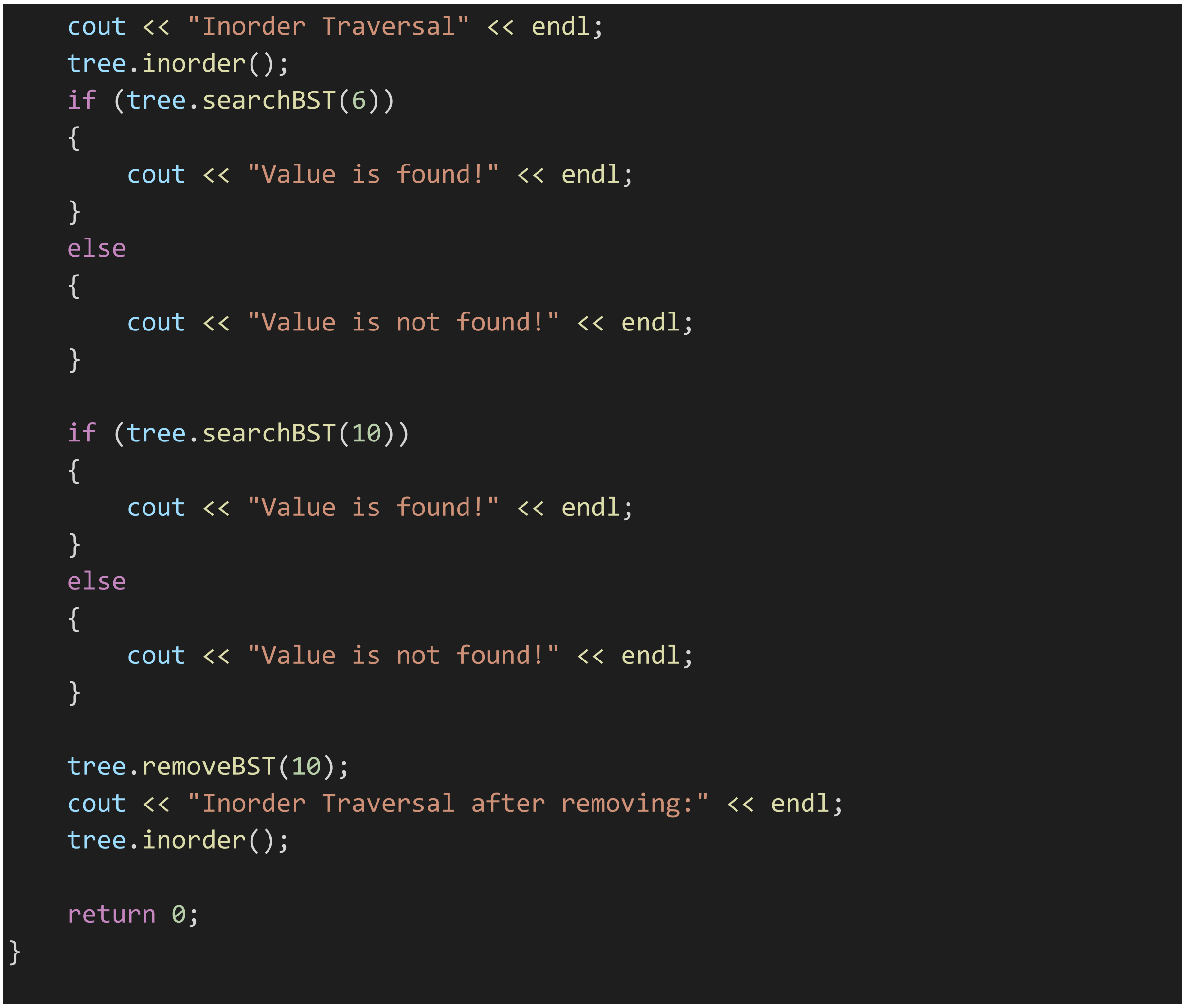
About BST

A binary search tree (BST) is a binary tree that is either empty or (in which each node contains a key that) satisfies the following properties.

- The keys (if any) in left sub-tree are smaller than the key in the root.
- The keys (if any) in the right subtree are larger than the key in the root.
- The left and right subtrees are also binary search trees.

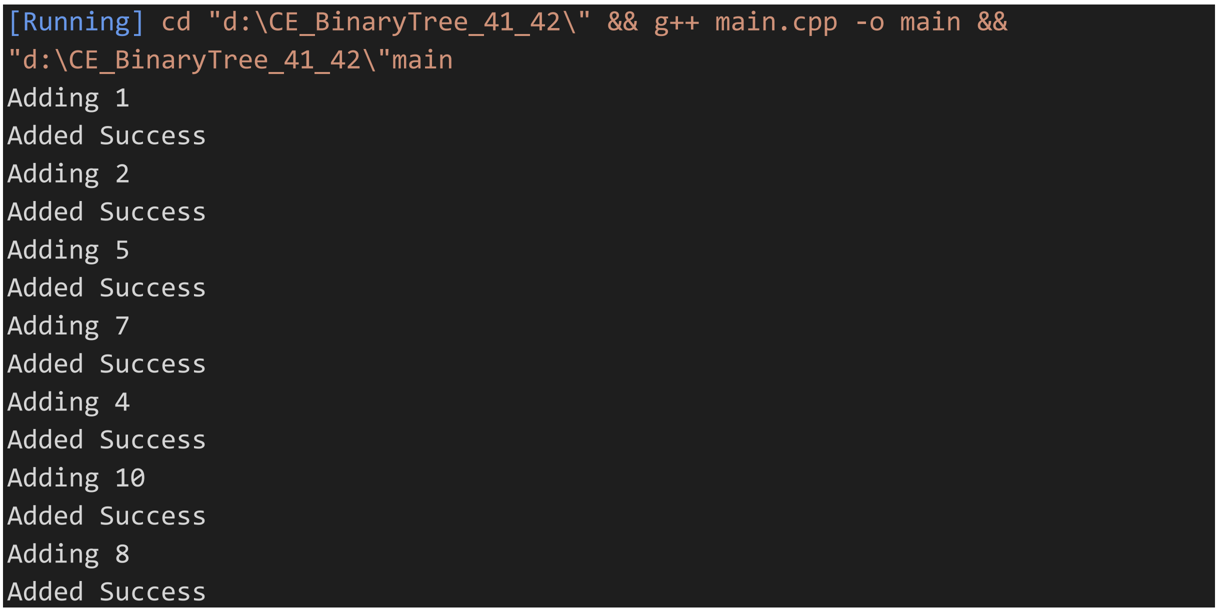
Main.cpp Code:

```
#include <iostream>
// #include "./header/arrayBST.h"
// #include "./cpp/arrayBST.cpp"
#include "./header/AbstractBST.h"
#include "./cpp/LinkedListBST.cpp"

using namespace std;

int main()
{
    LinkedBST tree;
    tree.addBST(1);
    tree.addBST(2);
    tree.addBST(5);
    tree.addBST(7);
    tree.addBST(4);
    tree.addBST(10);
    tree.addBST(8);
```

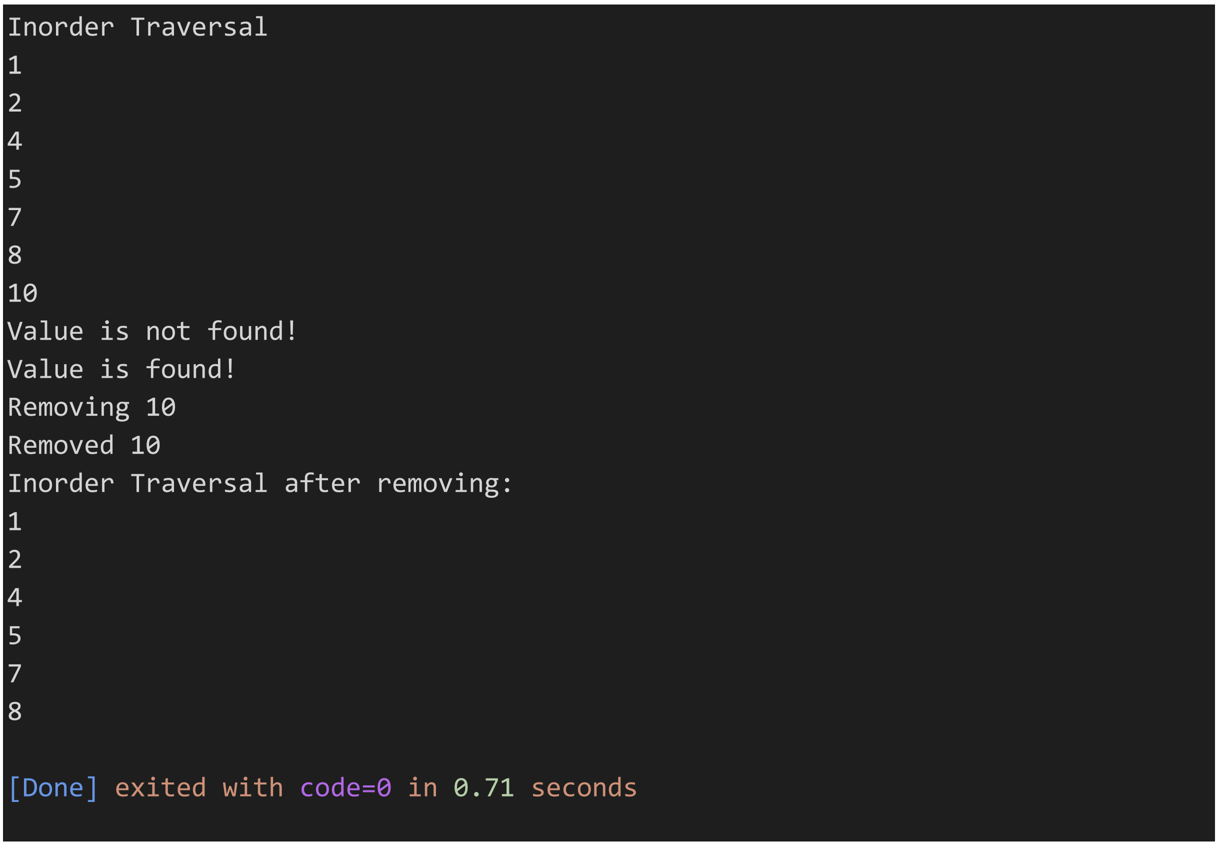


```cpp
    cout << "Inorder Traversal" << endl;
    tree.inorder();
    if (tree.searchBST(6))
    {
        cout << "Value is found!" << endl;
    }
    else
    {
        cout << "Value is not found!" << endl;
    }

    if (tree.searchBST(10))
    {
        cout << "Value is found!" << endl;
    }
    else
    {
        cout << "Value is not found!" << endl;
    }

    tree.removeBST(10);
    cout << "Inorder Traversal after removing:" << endl;
    tree.inorder();

    return 0;
}
```

Output Of The Above Problem:



```
[Running] cd "d:\CE_BinaryTree_41_42\" && g++ main.cpp -o main &&
"d:\CE_BinaryTree_41_42\"main
Adding 1
Added Success
Adding 2
Added Success
Adding 5
Added Success
Adding 7
Added Success
Adding 4
Added Success
Adding 10
Added Success
Adding 8
Added Success
```



```
Inorder Traversal
1
2
4
5
7
8
10
Value is not found!
Value is found!
Removing 10
Removed 10
Inorder Traversal after removing:
1
2
4
5
7
8

[Done] exited with code=0 in 0.71 seconds
```

Description:

The above code implements a Binary Search Tree (BST) using a linked list data structure. It includes the definition of a `Node` struct, which represents a single node in the BST. The class **`LinkedBST`** inherits from the **`AbstractBST`** class, which contains abstract methods for the basic operations of a BST.

1. Node struct:

- It contains three members: `data` to store the value of the node, and `left` and `right` pointers to the left and right child nodes, respectively.

2. LinkedBST class:

- **isEmpty():** Checks whether the BST is empty or not.
- **addBST(int):** Adds a new element to the BST.
- **removeBST(int):** Removes an element from the BST.
- **searchBST(int):** Searches for an element in the BST.
- **inorder():** Performs an inorder traversal of the BST.
- **add(Node*, int)`:** Recursive helper function to add a new node to the BST.
- **search(Node*, int):** Recursive helper function to search for a value in the BST.
- **inorder(Node*):** Recursive helper function to perform an inorder traversal of the BST.
- **Delete(Node*, int):** Recursive helper function to delete a node from the BST.

3. The main() function:

- It demonstrates the usage of the LinkedBST class by creating an instance of it.
- Elements are added to the BST using the addBST() method.
- The inorder() method is called to perform an inorder traversal and print the elements of the BST.
- The searchBST() method is used to search for specific values in the BST.
- The removeBST() method is called to remove an element from the BST.
- Finally, the inorder() method is called again to show the updated BST after the removal.

The code uses recursion for various BST operations like insertion, searching, and deletion. The inorder()` traversal displays the elements in ascending order.

Github Link: 1014Aayush/CE_BinaryTree_41_42 (github.com)