National University of Sciences and Technology (NUST)
School of Electrical Engineering and Computer Science

Department of Computing

School of Electrical Engineering and Computer Science

CS-250: Data Structure and Algorithms

Class: BESE 13A

Lab 9: Implementation of Binary Search Tree

Date: 24th November, 2023

Time: 10 am - 1 pm

Name: Aimen Munawar

Class: BESE-13-A

CMS ID: 415867

Lab Engineer: Anum Asif

Lab 9: Implementation of Binary Search Tree

Introduction

This lab is based on the implementation of Binary Search tree and its functions.

Objectives

The objectives of this lab are the following:

- Become familiar with implementation of binary search trees
- Study some statistics of binary search trees
- Write simple applications using binary search tree

Tools/Software Requirement

Visual Studio 2012 or gcc or g++

Helping Material

Lecture slides, text book

Description

In computer science, a binary search tree (BST), which may sometimes also be called an ordered or sorted binary tree, is a node-based binary tree data structure which has the following properties:

- The left sub-tree of a node contains only nodes with keys less than the node's key.
- The right sub-tree of a node contains only nodes with keys greater than the node's key.
- Both the left and right sub-trees must also be binary search trees.
- There must be no duplicate nodes.

In this lab, you will expand implement binary search tree, study some statistical properties of BST and write a simple application using the BST.

Here is a template of how your class/structure looks like.

```
struct tree_node{
        tree_node* left;
        tree_node* right;
        int data;
};
```

Skeleton File: A skeleton file with partial code in it has been created for you (*lab-11-skeleton.cpp*). You should update it as per instructions given below.

Tasks

Implement the following operations of Binary Search Tree ADT. A skeleton file is already created for you

1. Write a main method which contains a menu to take user's inputs. The menu will allow the user to do the following operations:
   a) Insert new data
   b) In-Order Traversal
   c) Pre-Order Traversal
   d) Post-Order Traversal
   e) Delete an item
2. For insertion operation, the user will give a value which will be added in the binary search tree at its correct position.
3. For in-order, pre-order, and post-order traversals, the user should see the traversed tree as output in correct order.
4. On removal of an item, the nodes should connect correctly so that it does not break the data hierarchy within the tree.
5. Test your program with different inputs from smaller array e.g. of size 10 to larger arrays of size 100.

Important Note: Practice your knowledge of OOP with C++ when creating a solution. Remember to comment your code properly. Inappropriate or no comment may result in deduction of marks.

Solution

Solution
Task 1 Screen Shots (execution: Insertion, 3 types of traversals, deletion): **Testing Program with Smaller inputs:** 1. **Insertion:**

Inserting Single value:

```
        ****************************************
        * Select Option:                       *
        *       1. Insert Single Value         *
        *       2. Insert Multiple Values              *
        *       3. Traversal (In-Order) *
        *       4. Traversal (Pre-Order)       *
        *       5. Traversal (Post-Order)      *
        *       6. Delete a Value              *
        *       7. Exit                        *
        ****************************************
> 1
Enter value to insert: 3
```

Inserting Multiple Values:

```
        ****************************************
        * Select Option:                       *
        *       1. Insert Single Value         *
        *       2. Insert Multiple Values              *
        *       3. Traversal (In-Order) *
        *       4. Traversal (Pre-Order)       *
        *       5. Traversal (Post-Order)      *
        *       6. Delete a Value              *
        *       7. Exit                        *
        ****************************************
> 2
Enter the Total number of values you want to insert: 4
Enter values: 6
8
4
5
```

2. **Traversals:**

Pre-order:

```
        ****************************************
        * Select Option:                       *
        *       1. Insert Single Value         *
        *       2. Insert Multiple Values              *
        *       3. Traversal (In-Order) *
        *       4. Traversal (Pre-Order)       *
        *       5. Traversal (Post-Order)      *
        *       6. Delete a Value              *
        *       7. Exit                        *
        ****************************************
> 4
Pre-Order Traversal: 3 6 4 5 8
```

In-Order:

```
        ****************************************
        * Select Option:                       *
        *       1. Insert Single Value         *
        *       2. Insert Multiple Values              *
        *       3. Traversal (In-Order) *
        *       4. Traversal (Pre-Order)       *
        *       5. Traversal (Post-Order)      *
        *       6. Delete a Value              *
        *       7. Exit                        *
        ****************************************
> 3
In-Order Traversal: 3 4 5 6 8
```

Post-Order:

```
        ****************************************
        * Select Option:                       *
        *       1. Insert Single Value         *
        *       2. Insert Multiple Values              *
        *       3. Traversal (In-Order) *
        *       4. Traversal (Pre-Order)       *
        *       5. Traversal (Post-Order)      *
        *       6. Delete a Value              *
        *       7. Exit                        *
        ****************************************
> 5
Post-Order Traversal: 5 4 8 6 3
```

3. **Deletion:**

```
        ****************************************
        * Select Option:                       *
        *       1. Insert Single Value         *
        *       2. Insert Multiple Values              *
        *       3. Traversal (In-Order) *
        *       4. Traversal (Pre-Order)       *
        *       5. Traversal (Post-Order)      *
        *       6. Delete a Value              *
        *       7. Exit                        *
        ****************************************
> 6
Enter value to delete: 8
Value 8 deleted.
        ****************************************
        * Select Option:                       *
        *       1. Insert Single Value         *
        *       2. Insert Multiple Values              *
        *       3. Traversal (In-Order) *
        *       4. Traversal (Pre-Order)       *
        *       5. Traversal (Post-Order)      *
        *       6. Delete a Value              *
        *       7. Exit                        *
        ****************************************
> 3
In-Order Traversal: 3 4 5 6
```

Testing Program with large inputs (100 values):

Insertion:

```
        ****************************************
        * Select Option:                       *
        *       1. Insert Single Value         *
        *       2. Insert Multiple Values              *
        *       3. Traversal (In-Order) *
        *       4. Traversal (Pre-Order)       *
        *       5. Traversal (Post-Order)      *
        *       6. Delete a Value              *
        *       7. Exit                        *
        ****************************************
> 2
Enter the Total number of values you want to insert: 100
Enter values: 1
2
3
4
23
7
42
15
67
3
51
10
36
78
2
49
19
88
29
58
13
63
91
5
24
```

```
63
91
5
24
77
39
54
8
45
30
74
12
60
26
82
17
69
35
53
9
46
18
81
14
66
21
73
37
95
28
55
6
89
16
68
31
75
```

```
98
4
50
27
83
40
65
34
79
48
93
1
61
32
76
20
97
43
70
52
86
25
94
41
64
33
80
47
92
59
85
44
62
84
56
74
99
```

Traversal:

In-Order:

```
       ****************************************
       * Select Option:                       *
       *       1. Insert Single Value         *
       *       2. Insert Multiple Values              *
       *       3. Traversal (In-Order) *
       *       4. Traversal (Pre-Order)       *
       *       5. Traversal (Post-Order)      *
       *       6. Delete a Value              *
       *       7. Exit                        *
       ****************************************
> 3
In-Order Traversal: 1 1 2 2 3 3 4 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49
50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 72 73 74 74 75 76 77 78 79 80 81 82 83 84 85 86 88 89 91 92 93 94 95 97 98 99
```

Pre-Order:

```
        ****************************************
        * Select Option:                       *
        *       1. Insert Single Value         *
        *       2. Insert Multiple Values              *
        *       3. Traversal (In-Order) *
        *       4. Traversal (Pre-Order)       *
        *       5. Traversal (Post-Order)      *
        *       6. Delete a Value              *
        *       7. Exit                        *
        ****************************************
> 4
Pre-Order Traversal: 1 2 1 3 2 4 3 23 7 5 4 6 15 10 8 9 13 12 11 14 19 17 16 18 21 20 22 42 36 29 24 26 25 28 27 30 35 31 34 32 33 39 37 38 40 41 67 51 49 45 43 44 46
 48 47 50 58 54 53 52 55 57 56 63 60 59 61 62 66 65 64 78 77 74 69 68 73 72 70 75 74 76 88 82 81 79 80 83 86 85 84 91 89 95 93 92 94 98 97 99
```

Post-Order:

```
        ****************************************
        * Select Option:                       *
        *       1. Insert Single Value         *
        *       2. Insert Multiple Values              *
        *       3. Traversal (In-Order) *
        *       4. Traversal (Pre-Order)       *
        *       5. Traversal (Post-Order)      *
        *       6. Delete a Value              *
        *       7. Exit                        *
        ****************************************
> 5
Post-Order Traversal: 1 2 3 4 6 5 9 8 11 12 14 13 10 16 18 17 20 22 21 19 15 7 25 27 28 26 24 33 32 34 31 35 30 29 38 37 41 40 39 36 44 43 47 48 46 45 50 49 52 53 56
57 55 54 59 62 61 60 64 65 66 63 58 51 68 70 72 73 69 74 76 75 74 77 80 79 81 84 85 86 83 82 89 92 94 93 97 99 98 95 91 88 78 67 42 23 4 3 2 1
```

Deletion:

```
        ****************************************
        * Select Option:                       *
        *       1. Insert Single Value         *
        *       2. Insert Multiple Values              *
        *       3. Traversal (In-Order) *
        *       4. Traversal (Pre-Order)       *
        *       5. Traversal (Post-Order)      *
        *       6. Delete a Value              *
        *       7. Exit                        *
        ****************************************
> 6
Enter value to delete: 93
Value 93 deleted.
        ****************************************
        * Select Option:                       *
        *       1. Insert Single Value         *
        *       2. Insert Multiple Values              *
        *       3. Traversal (In-Order) *
        *       4. Traversal (Pre-Order)       *
        *       5. Traversal (Post-Order)      *
        *       6. Delete a Value              *
        *       7. Exit                        *
        ****************************************
> 3
In-Order Traversal: 1 1 2 2 3 3 4 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49
50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 72 73 74 74 75 76 77 78 79 80 81 82 83 84 85 86 88 89 91 92 94 95 97 98 99
```

Task 1 Code:

```
#include <iostream>
#include <vector>

using namespace std;

class BinarySearchTree {
public:
    class TreeNode {
    public:
        int value;
        TreeNode* right;
        TreeNode* left;

        // Constructor for the tree node
        TreeNode() {
            value = 0;
            right = left = nullptr;
        }

        // Constructor with initial value for the tree node
        TreeNode(int x) {
            value = x;
            left = right = nullptr;
        }
```

```
};

TreeNode* root; // Root of the Binary Search Tree

// Constructor for the Binary Search Tree
BinarySearchTree() {
    root = nullptr;
}

// Check if the Binary Search Tree is empty
bool isEmpty() {
    return root == nullptr;
}

// Public interface for inserting a single value into the tree
void insertValue(int data) {
    root = insertNode(root, data);
}

// Public interface for inserting an array of values into the tree
void insertArray(const vector<int>& values) {
    for (int value : values) {
        insertValue(value);
    }
```

```
}

// Public interface for removing a value from the tree
void removeValue(int key) {
    if (!isEmpty()) {
        root = deleteNode(root, key);
        cout << "Value " << key << " deleted.\n";
    } else {
        cout << "Tree is empty. Cannot delete.\n";
    }
}

// Private helper function for deleting a node with a specific value
TreeNode* deleteNode(TreeNode* currentNode, int key) {
    if (currentNode == nullptr)
        return currentNode;

    if (key < currentNode->value) {
        currentNode->left = deleteNode(currentNode->left, key);
    } else if (key > currentNode->value) {
        currentNode->right = deleteNode(currentNode->right, key);
    } else {
        if (currentNode->left == nullptr) {
            TreeNode* temp = currentNode->right;
```

```
        delete currentNode;
        return temp;
    } else if (currentNode->right == nullptr) {
        TreeNode* temp = currentNode->left;
        delete currentNode;
        return temp;
    }

    TreeNode* successorParent = currentNode;
    TreeNode* successor = currentNode->right;
    while (successor->left != nullptr) {
        successorParent = successor;
        successor = successor->left;
    }

    if (successorParent != currentNode)
        successorParent->left = successor->right;
    else
        successorParent->right = successor->right;

    currentNode->value = successor->value;
    delete successor;
}
return currentNode;
```

```
}

// Private helper function for inserting a value into the BST
TreeNode* insertNode(TreeNode* currentNode, int data) {
    if (!currentNode) {
        return new TreeNode(data);
    }

    if (data < currentNode->value) {
        currentNode->left = insertNode(currentNode->left, data);
    } else if (data >= currentNode->value) {
        currentNode->right = insertNode(currentNode->right, data);
    }

    return currentNode;
}

// Public interface for tree traversal
void traverse(int type) {
    switch (type) {
    case 1:
        cout << "In-Order Traversal: ";
        traverseInOrder(root);
        break;
```

```
    case 2:
        cout << "Pre-Order Traversal: ";
        traversePreOrder(root);
        break;
    case 3:
        cout << "Post-Order Traversal: ";
        traversePostOrder(root);
        break;
    default:
        cout << "Invalid traversal type.\n";
    }
    cout << endl;
}

// Recursive function for in-order traversal
void traverseInOrder(TreeNode* currentNode) {
    if (currentNode == nullptr) {
        return;
    }
    traverseInOrder(currentNode->left);
    cout << currentNode->value << " ";
    traverseInOrder(currentNode->right);
}
```

```
    // Recursive function for pre-order traversal
    void traversePreOrder(TreeNode* currentNode) {
        if (currentNode == nullptr) {
            return;
        }
        cout << currentNode->value << " ";
        traversePreOrder(currentNode->left);
        traversePreOrder(currentNode->right);
    }

    // Recursive function for post-order traversal
    void traversePostOrder(TreeNode* currentNode) {
        if (currentNode == nullptr) {
            return;
        }
        traversePostOrder(currentNode->left);
        traversePostOrder(currentNode->right);
        cout << currentNode->value << " ";
    }
};

int main() {
    BinarySearchTree bst; // Create a Binary Search Tree
```

```
int menuOption = 0;

while (menuOption != 7) {

   cout << "\t*****************************************" << endl;

   cout << "\t* Select Option:\t\t\t*" << endl;

   cout << "\t*\t1. Insert Single Value\t\t*" << endl;

   cout << "\t*\t2. Insert Multiple Values\t\t*" << endl;

   cout << "\t*\t3. Traversal (In-Order)\t*" << endl;

   cout << "\t*\t4. Traversal (Pre-Order)\t*" << endl;

   cout << "\t*\t5. Traversal (Post-Order)\t*" << endl;

   cout << "\t*\t6. Delete a Value\t\t*" << endl;

   cout << "\t*\t7. Exit\t\t\t\t*" << endl;

   cout << "\t*****************************************" << endl;


   cout << "> ";

   cin >> menuOption;


   switch (menuOption) {

   case 1: {

      int valueToInsert;

      cout << "Enter value to insert: ";

      cin >> valueToInsert;

      bst.insertValue(valueToInsert);

      break;

   }
```

```
case 2: {
    int arraySize;
    cout << "Enter the Total number of values you want to insert: ";
    cin >> arraySize;
    vector<int> values(arraySize);
    cout << "Enter values: ";
    for (int i = 0; i < arraySize; i++) {
        cin >> values[i];
    }
    bst.insertArray(values);
    break;
}
case 3:
case 4:
case 5: {
    bst.traverse(menuOption - 2);
    break;
}
case 6: {
    int valueToDelete;
    cout << "Enter value to delete: ";
    cin >> valueToDelete;
    bst.removeValue(valueToDelete);
    break;
```

```
    }

    }

  }


  return 0;

}
```

Deliverables

Compile a single word document by filling in the solution part and submit this Word file on LMS. Insert the solution/answer in this document. You must show the implementation of the tasks in the designing tool, along with your complete Word document to get your work graded. You must also submit this Word document on the LMS.