Data Structures and Algorithms

Lab 3 – Binary Tree

The following code is applied to all questions.

```c
#define NODE_TYPE_OPERATOR 0
#define NODE_TYPE_OPERAND 1
#define OPERATOR_ADD 0 // +
#define OPERATOR_MINUS 1 // -

struct NodeEntry
{
    int type;
    int value;
    void printNode()
    {
        if (type == NODE_TYPE_OPERAND)
        {
            cout<<value;
        }
        else
        {
            switch (value)
            {
                case OPERATOR_ADD:
                    cout<<"+";
                    break;
                case OPERATOR_MINUS:
                    cout<<"-";
                    break;
            }
        }
    }
};

struct TreeNode
{
    NodeEntry entry;
    TreeNode *left, *right;

    TreeNode() { left = right = NULL; }
    TreeNode(NodeEntry item, TreeNode * left = NULL, TreeNode * right = NULL)
    {
        this->entry = item;
        this->left = left;
        this->right = right;
    }
};
```
```cpp
class BinaryTree {
public:
    BinaryTree()
    {
        root = NULL;
    }

    ~BinaryTree()
    {
        destroy(root);
        root = NULL;
    }

    bool empty()
    {
        return (root == NULL);
    }

    bool insertAt(TreeNode *parent, bool leftOrRight, NodeEntry data, TreeNode * &newNode)
    {
        if (parent == NULL) {
            if (root != NULL)
                return false;
        } else {
            if ((leftOrRight && (parent->left != NULL))
                || (!leftOrRight && (parent->right != NULL)))
                return false;
        }

        newNode = new TreeNode(data);

        if (parent == NULL)
            root = newNode;
        else {
            if (leftOrRight)
                parent->left = newNode;
            else
                parent->right = newNode;
        }
        return true;
    }
};
```
void printPreOrder()  //NLR
{
    // add your code here for question 2a
}

void printInOrder()  //LNR
{
    // add your code here for question 2b
}

void printPostOrder()
{
    // add your code here for question 2c
}

int heightTree()
{
    // add your code here for question 3
}

int calculateBlanceFactor ()
{
    //add your code here for question 4
}

int countLeaf()
{
    //add your code here for question 5
}

void deleteLeaves()
{
    //add your code here for question 6
}

TreeNode* findValue(NodeEntry value)
{
    //add your code here for question 7
}

void swapNode()
{
    //add your code here for question 8
}

int caculateTree()
{
    //add your code here for question 9
}
```cpp
void build_tree_from_keyboard ()
{
    root = build_tree_from_keyboard_recur();
}

protected:
    TreeNode *root;

void destroy(TreeNode *subroot)
{
    if (subroot != NULL) {
        destroy(subroot->left);
        destroy(subroot->right);
        delete subroot;
    }
}

TreeNode * build_tree_from_keyboard_recur ()
{
    char ans;
    cout << "Enter more (Y/N)? ";
    cin >> ans;
    if (ans == 'Y') {
        NodeEntry data;
        cout << "Enter an entry (type, data): \n";
        cin >> data.type >> data.value;
        
        TreeNode * p = new TreeNode(data);
        cout << "Enter the left sub-tree \n";
        p->left = build_tree_from_keyboard_recur ();
        cout << "Enter the right sub-tree \n";
        p->right = build_tree_from_keyboard_recur ();
        return p;
    }
    return NULL;
}
```
**Question 1:** Use the already implemented method `insertAt` to construct binary tree as follow:

![Binary Tree Diagram]

**Question 2:** Implement method to print the tree.
- a) PreOrder
- b) InOrder
- c) PostOrder

**Question 3:** Implement method `heightTree` to calculate the height of the tree.

**Question 4:** Implement method `calculateBalanceFactor` to calculate the balance factor of the tree.

**Question 5:** Implement method `countLeaf` to count the number of leaves in the tree.

**Question 6:** Implement method `deleteLeaves` to delete all leaves from the tree.

**Question 7:** Implement method `findValue` to find a node with value ‘value’.

**Question 8:** Implement method `swapNode` to swap the left and the right sub-trees at any node.

**Question 9:** Support that an expression can be presented as a binary tree as in question 4.1 (*)

In that expression tree, leaves are used to present operands, which are numbers. Other nodes on the tree are used to present operators (plus and minus).

Implement method `calculateTree` to calculate an expression expressed as a tree above.

---

(*) Data Structures and Algorithms in C++, 4th, Adam Drozdek, chapter 6, section 6.12, page 286.