Expression Tree

SUBMITTED BY:

JASKIRAT SINGH

2020CSC1008

Code

```
#include <iostream>

#include <stack>

#include <string>

#include <cmath>

#include <sstream> // use stringstream class

using namespace std;


class Node

{
```

```
public:
    string data;
    Node *left, *right;
    Node() // Default constructor.
    {
        data = "";
        left = right = NULL;
    }
    Node(int value) // Parameterized constructor.
    {
        data = value;
        left = right = NULL;
    }
};

class BST
{
public:
    Node *root;

    BST()
    {
        root = NULL;
    }
    void inorder(Node *troot);
```

```
    Node *constructTree(char postfix[], int n);
    int eval(Node *cur);
};

void BST::inorder(Node *troot)
{
    if (troot != NULL)
    {
        inorder(troot->left);
        cout << troot->data << " ";
        inorder(troot->right);
    }
}

Node *BST::constructTree(char postfix[], int n)
{
    stack<Node *> st;
    Node *t = new Node;
    for (int i = 0; i < n; i++) // Traverse through every character of input expression
    {

        if (postfix[i] == '+' || postfix[i] == '-' || postfix[i] == '*' || postfix[i] == '/' || postfix[i] ==
'^') // operator
        {
            t = new Node(postfix[i]);
```

```
            t->right = st.top(); // Store top node
            st.pop();
            t->left = st.top();
            st.pop();
            st.push(t); // Add this subexpression to stack
        }
        else // If operand, simply push into stack
        {
            t = new Node(postfix[i]);
            st.push(t);
        }
    }
    t = st.top(); // only element will be root of expression tree
    st.pop();
    return t;
}

int BST::eval(Node *cur)
{
    if (!cur) // empty tree
        return 0;
    if (!cur->left && !cur->right) // leaf node i.e, an integer
    {
        int num = stoi(cur->data);
```

```
        return num;
    }

    int l_val = eval(cur->left);  // Evaluate left subtree
    int r_val = eval(cur->right); // Evaluate right subtree
    if (cur->data == "+")         // Check which operator to apply
        return l_val + r_val;
    if (cur->data == "-")
        return l_val - r_val;
    if (cur->data == "*")
        return l_val * r_val;
    if (cur->data == "/")
        return l_val / r_val;
    if (cur->data == "^")
        return pow(l_val, r_val);

    return -1; // invalid operator
}

int main()
{
    char arr[] = {'3', '5', '9', '+', '2', '-', '+'};

    BST bst;
    bst.root = bst.constructTree(arr, 7);
```

```
    cout << "\nElements in the tree... ";
    Node *head = bst.root;
    bst.inorder(head);
    cout << "\nEvaluation result : " << bst.eval(head);

    return 0;
}
```

Output