C++ Program for Construction of Expression Tree-In-order traversal

```
#include
using namespace std;
// Tree Node Definition
class TreeNode
{
public:
    char val;
    TreeNode *left,*right;
    TrreeNode()
    {
        this->left = NULL;
        this->right = NULL;
    }
    // Constructor Method
    TreeNode(char val)
    {
        this->val = val;
        this->left = NULL;
        this->right = NULL;
    }
};
// Stack to hold the latest node
class Stack
{
public:
    TreeNode *treeNode;
    Stack *next;
    // Constructor Method
    Stack(TreeNode *treeNode)
    {
        this->treeNode = treeNode;
        next = NULL;
    }
};
class ExpressionTree
{
private:
    Stack *top;
public:
    // Constructor Method
    ExpressionTree()
```

```
{
   top = NULL;
}
// function to push a node in stack
void push(TreeNode *ptr)
{
   if (top == NULL)
      top = new Stack(ptr);
   else
   {
      Stack *nptr = new Stack(ptr);
      nptr->next = top;
      top = nptr;
   }
}
TreeNode *pop()
{
   TreeNode *ptr = top->treeNode;
   top = top->next;
   return ptr;
}
TreeNode *peek()
{
   return top->treeNode;
}
// function to insert character
void insert(char val)
{
   // If the encountered character is Number make a node an push it on stack
   if (isOperand(val))
   {
      TreeNode *nptr = new TreeNode(val);
      push(nptr);
   }
   // else if it is operator then make a node and left and
   else if (isOperator(val))
   {
      TreeNode *nptr = new TreeNode(val);
      nptr->left = pop();
      nptr->right = pop();
      push(nptr);
   }
```

```
    }
    //function to check if operand
    bool isOperand(char ch)
    {
        return ch >= '0' && ch <= '9' || ch>='A' && ch<='Z' || ch>='a' && ch<='z';
    }
    //function to check if operator
    bool isOperator(char ch)
    {
        return ch == '+' || ch == '-' || ch == '*' || ch == '/';
    }
    //function to construct expression Tree
    void construct(string eqn)
    {
        for (int i = eqn.length() - 1; i >= 0; i--)
            insert(eqn[i]);
    }
    void inOrder(TreeNode *ptr)
    {
        if (ptr != NULL)
        {
            inOrder(ptr->left);
            cout<<ptr->val;
            inOrder(ptr->right);
        }
    }
};
int main()
{
    string exp;
    ExpressionTree et;
    cout<<"Enter expression in Prefix form: ";
    cin>>exp;
    et.construct(exp);
    cout<<"In-order Traversal of Expression Tree : ";
    et.inOrder(et.peek());
    return 0;
}
}
```