```
/** C++ Program to Construct an Expression Tree for a Given Prefix Expression */
#include <iostream>
#include <cstdlib>
#include <cstdio>
#include <cstring>
using namespace std;
class TreeNode {
    public:
        char data;
        TreeNode *left, *right;
        /** constructor **/
        TreeNode(char data) {
            this->data = data;
            this->left = NULL;
            this->right = NULL;
        }
};
class StackNode {
    public:
        TreeNode *treeNode;
        StackNode *next;
        /** constructor **/
        StackNode(TreeNode *treeNode) {
            this->treeNode = treeNode;
            next = NULL;
        }
};
class ExpressionTree {
    private:  StackNode *top;
    public:
       /** constructor **/
        ExpressionTree() { top = NULL; }
        /** function to clear tree **/
        void clear() { top = NULL; }
        /** function to push a node **/
        void push(TreeNode *ptr) {
            if (top == NULL)
                top = new StackNode(ptr);
            else {
                StackNode *nptr = new StackNode(ptr);
                nptr->next = top;
                top = nptr;
            }
```

```
}
/** function to pop a node **/
TreeNode *pop() {
    if (top == NULL)
        cout<<"Underflow"<<endl;
    else {
        TreeNode *ptr = top->treeNode;
        top = top->next;
        return ptr;
    }
}
/** function to get top node **/
TreeNode *peek() {
    return top->treeNode;
}
/** function to insert character **/
void insert(char val) {
    if (isDigit(val)) {
        TreeNode *nptr = new TreeNode(val);
        push(nptr);
    }
    else if (isOperator(val)) {
        TreeNode *nptr = new TreeNode(val);
        nptr->left = pop();
        nptr->right = pop();
        push(nptr);
    }
    else {
        cout<<"Invalid Expression"<<endl;
        return;
    }
}
/** function to check if digit **/
bool isDigit(char ch) {
    return ch >= '0' && ch <= '9';
}
/** function to check if operator **/
bool isOperator(char ch) {
    return ch == '+' || ch == '-' || ch == '*' || ch == '/';
}
/** function to convert character to digit **/
int toDigit(char ch) {
    return ch - '0';
```

```
}
/** function to build tree from input */
void buildTree(string eqn) {
    for (int i = eqn.length() - 1; i >= 0; i--)
        insert(eqn[i]);
}
/** function to evaluate tree */
double evaluate() {
    return evaluate(peek());
}
/** function to evaluate tree */
double evaluate(TreeNode *ptr) {
    if (ptr->left == NULL && ptr->right == NULL)
        return toDigit(ptr->data);
    else {
        double result = 0.0;
        double left = evaluate(ptr->left);
        double right = evaluate(ptr->right);
        char op = ptr->data;
        switch (op) {
        case '+': result = left + right; break;
        case '-': result = left - right; break;
        case '*': result = left * right; break;
        case '/': result = left / right; break;
        default: result = left + right; break;
        }
        return result;
    }
}
/** function to get postfix expression */
void postfix() {
    postOrder(peek());
}
/** post order traversal */
void postOrder(TreeNode *ptr {
    if (ptr != NULL) {
        postOrder(ptr->left);
        postOrder(ptr->right);
        cout<<ptr->data;
    }
}
/** function to get infix expression */
void infix() {
```

```
            inOrder(peek());
        }
        /** in order traversal */
        void inOrder(TreeNode *ptr) {
            if (ptr != NULL) {
                inOrder(ptr->left);
                cout<<ptr->data;
                inOrder(ptr->right);
            }
        }
        /** function to get prefix expression */
        void prefix() {
            preOrder(peek());
        }
        /** pre order traversal */
        void preOrder(TreeNode *ptr) {
            if (ptr != NULL) {
                cout<<ptr->data;
                preOrder(ptr->left);
                preOrder(ptr->right);
            }
        }
};

/** Main Contains menu **/
int main()
{
    string s;
    cout<<"Expression Tree Test"<<endl;
    ExpressionTree et;
    cout<<"\nEnter equation in Prefix form: ";
    cin>>s; //input = +-+7*/935/82*/625
    et.buildTree(s);
    cout<<"\nPrefix  : ";
    et.prefix();
    cout<<"\n\nInfix   : ";
    et.infix();
    cout<<"\n\nPostfix : ";
    et.postfix();
    cout<<"\n\nEvaluated Result : "<<et.evaluate();
    return 0;
}
```