```cpp
/* DSL - EXPERIMENT 10 - D27 */

#include <iostream>
using namespace std;


#define MAX 50


// Class Stack
class Stack {
    private:
        char *data;                         // Dynamic allocation or else 'char data[MAX]'
        int top;

    public:
        Stack() {                           // Constructor initialization
            data = new char[MAX];           // Allocate 'data' with 'new char[MAX]'
            top = -1;
        }
        bool isEmpty();                     // Prototypes
        bool isFull();
        bool push(char x);
        char pop();
        char peek();
};


// Stack class function defintions
bool Stack::isEmpty() {
    return (top == -1);
}


bool Stack::isFull() {
    return (top == (MAX - 1));
}


bool Stack::push(char x) {
    if (isFull()) {
        cout << "Stack Overflow\n";
        return false;
    }
    else {
        data[++top] = x;
        return true;
    }
}
```

```
char Stack::pop() {
    char x;
    if (isEmpty()) {
        cout << "Stack Underflow\n";
        return 0;
    }
    else {
        x = data[top--];
        return x;
    }
}


char Stack::peek() {
    if (isEmpty()) {
        cout << "Stack is Empty\n";
        return 0;
    }
    else
        return data[top];
}


//Function to return precedence of operators
int prec(char c) {
    if(c == '^')
        return 3;
    else if(c == '*' || c == '/')
        return 2;
    else if(c == '+' || c == '-')
        return 1;
    else
        return -1;
}


bool isOperand(char c) {
    return (('a' ⩽ c && c ⩽ 'z') || ('A' ⩽ c && c ⩽ 'Z') || ('0' ⩽ c && c ⩽ '9'));
}


// The main function to convert infix expression to postfix expression
string infixToPostfix(string s) {
    Stack stack;
    string postfix;

    for(int i = 0; i < s.length(); i++) {

        // If the scanned character is an operand, add it to output string.
        if (isOperand(s[i]))
            postfix += s[i];
```

```
        // If the scanned character is an ‘(‘, push it to the stack.
        else if (s[i] == '(')
            stack.push('(');

        // If the scanned character is an ‘)’,
        // pop and to output string from the stack until an ‘(‘ is encountered.
        else if (s[i] == ')') {
            while (!stack.isEmpty() && stack.peek() != '(')
                postfix += stack.pop();

            if (stack.peek() == '(')
                stack.pop();
        }

        //If an operator is scanned
        else {
            while (!stack.isEmpty() && prec(s[i]) <= prec(stack.peek()))
                postfix += stack.pop();

            stack.push(s[i]);
        }

    }

    // Pop all the remaining elements from the stack
    while(!stack.isEmpty())
        postfix += stack.pop();

    return postfix;
}


// Evaluate Postfix expression
int evalPostfix(string pfexp) {
    Stack stack;

    for (char c : pfexp) {

        // If operand then push to stack
        if (isdigit(c))
            stack.push(c - '0');

        // If operator then pop 2 digits from sack,
        // use operator and push value to stack
        else {
            int val1 = stack.pop();
            int val2 = stack.pop();

            switch (c) {
                case '+':
```

```
                stack.push(val2 + val1);
                break;
            case '-':
                stack.push(val2 - val1);
                break;
            case '*':
                stack.push(val2 * val1);
                break;
            case '/':
                stack.push(val2 / val1);
                break;
        }
    }
}

    return stack.pop();
}


//Driver program to test above functions
int main() {
    string exp;
    string pfexp;
    cout << "\nEnter an Infix Expression : "; cin >> exp;
    pfexp = infixToPostfix(exp);
    cout << "\nPostfix Expression is given by : " << pfexp << endl;
    cout << "\nThis Postfix expression is evaluated as : " << evalPostfix(pfexp) << endl;
    return 0;
}
```

```
/*
_____________ OUTPUT _____________

Enter an Infix Expression : 1+2-(4*(8/2-4))/3+6

Postfix Expression is given by : 12+482/4-*3/-6+

This Postfix expression is evaluated as : 9

*/
```