```
#include <iostream>
#include <ctype.h>
#include <string.h>
#include <math.h>

using namespace std;

struct node {
   char data;
   struct node *next;
};

class stack {
   node *top;
public:
   stack() {
      top = NULL;
   }

   char Top() {
      return (top->data);
   }

   void push(char x) {
      node *temp;
      temp = new node;
      temp->data = x;
      temp->next = top;
      top = temp;
   }

   char pop() {
      if (top == NULL) return '\0';
      char value = top->data;
      node *temp = top;
      top = top->next;
      delete temp;
      return value;
   }

   int isempty() {
      return (top == NULL);
   }
};

int priority(char op) {
   if (op == '(' || op == ')')
      return 0;
   else if (op == '+' || op == '-')
      return 1;
   else if (op == '*' || op == '/' || op == '%')
      return 2;
```

```
    else if (op == '^')
        return 3;
    else
        return -1;
}

int operation(char op, int A, int B) {
    if (op == '*')
        return A * B;
    else if (op == '/')
        return A / B;
    else if (op == '^')
        return pow(A, B);
    else if (op == '+')
        return A + B;
    else if (op == '-')
        return A - B;
    else
        return -1;
}

void infixtopostfix(char infix[50]) {
    char token, operand, post[50];
    int i, j = 0;
    stack S;

    for (i = 0; infix[i] != '\0'; i++) {
        token = infix[i];
        if (isalnum(token))
            post[j++] = token;
        else if (token == '(')
            S.push(token);
        else if (token == ')') {
            while (!S.isempty() && (operand = S.pop()) != '(')
                post[j++] = operand;
        } else {
            while (!S.isempty() && priority(S.Top()) >= priority(token))
                post[j++] = S.pop();
            S.push(token);
        }
    }

    while (!S.isempty())
        post[j++] = S.pop();

    post[j] = '\0';
    cout << "\nPostfix Expression: " << post << endl;
}

void infixtoprefix(char infix[50]) {
    char token, operand, pre[50];
    int i, j = 0;
```

```
    stack S;

    for (i = strlen(infix) - 1; i >= 0; i--) {
        token = infix[i];
        if (isalnum(token))
            pre[j++] = token;
        else if (token == ')')
            S.push(token);
        else if (token == '(') {
            while (!S.isempty() && (operand = S.pop()) != ')')
                pre[j++] = operand;
        } else {
            while (!S.isempty() && priority(S.Top()) > priority(token))
                pre[j++] = S.pop();
            S.push(token);
        }
    }

    while (!S.isempty())
        pre[j++] = S.pop();

    pre[j] = '\0';

    cout << "\nPrefix Expression: ";
    for (i = strlen(pre) - 1; i >= 0; i--)
        cout << pre[i];
    cout << endl;
}

int main() {
    int choice;
    char expression[50];

    do {
        cout << "\nEnter Choice of Operation:\n";
        cout << "1. Infix to Postfix\n";
        cout << "2. Infix to Prefix\n";
        cout << "3. Exit\n";
        cout << "Enter your choice: ";
        cin >> choice;

        switch (choice) {
        case 1:
            cout << "Enter Infix Expression: ";
            cin >> expression;
            infixtopostfix(expression);
            break;

        case 2:
            cout << "Enter Infix Expression: ";
            cin >> expression;
            infixtoprefix(expression);
```

```
            break;

        case 3:
            cout << "End of program.\n";
            break;

        default:
            cout << "Wrong Choice!\n";
            break;
        }
    } while (choice != 3);

    return 0;
}
```