#include <iostream>

#include "stack.h"

#include <cstring>

using namespace std;

// Structure to represent a variable

struct Variable {

    char vName;               // Variable name

    int val;                  // Variable value

};

class Conversion {

public:

    Stack<char>s;

    char prefix[100];       // Array to store the prefix expression

    char postfix[100];      // Array to store the postfix expression

    Variable variables[26];

    bool isOperand(char a);                         //Function to check if character is Operand

    bool isOperator(char a);                        //Function to check if character is Operator

    int isp(char x);                                //Function to define InStack Priority

    int icp(char x);                                //Function to define InComing Priority

    void infixToPostfix(const char infix[]);        //Function to Convert infix expression to postfix expression

    void infixToPrefix(const char infix[]);         //Function to Convert infix expression to prefix expression

    int evaluatePostfix();                          //Function to evaluate postfix expression

    int evaluatePrefix();                           //Function to evaluate prefix expression

};

// Function to check if a character is an operand

bool Conversion::isOperand(char a){                     //Function to check if character is Operand

    return isalpha(a);

}

// Function to check if a character is an operator

bool Conversion::isOperator(char a){                    //Function to check if character is Operator

    return (a == '+' || a == '-' || a == '\*' || a == '/' || a == '^' || a == '(' || a == ')');

}

// Function to get the in-stack priority of an operator

int Conversion::isp(char x){

    if (x == '^')                           //InStack Priority

        return 3;                           // 3: ^

    else if (x == '\*' || x == '/')          // 2: \* and /

        return 2;                           // 1: + and -

    else if (x == '+' || x == '-')          // 0: ( and )

        return 1;

    else if (x == '(' || x == ')')

        return 0;

    else

        return -1;

}

// Function to get the incoming priority of an operator

int Conversion::icp(char x) {

    if (x == '(')                           // 4: (

        return 4;                           // 3: ^

    else if (x == '^')                      // 2: \* and /

        return 3;                           // 1: + and -

    else if (x == '\*' || x == '/')          // 0: )

        return 2;

    else if (x == '+' || x == '-')

        return 1;

    else if (x == ')')

        return 0;

    else

        return -1;

}

// Function to convert infix to postfix

void Conversion::infixToPostfix(const char infix[]){

    int i = 0;

    int j = 0;

    while (infix[i] != '\0') {

        char curr = infix[i];

        if(curr=='('){                       // if current character is opening bracket

            s.push(curr);

            i++;

        }

        else if(curr==')'){                  // if current character is closing bracket

            while(!s.isEmpty() && s.peek()!='('){

                postfix[j++]=s.pop();

            }

            if(!s.isEmpty() && s.peek()=='('){

                s.pop();

            }

            i++;

        }

        else if(isOperand(curr)){             //if current character is operand

            postfix[j++]=curr;

            i++;

        }

        else if(isOperator(curr)){           //if current character is operator

            if(s.isEmpty()){

                s.push(curr);

            }

            else if(icp(curr)>=isp(s.peek())){

                s.push(curr);

            }

            else{

                while((!s.isEmpty()) && icp(curr)<isp(s.peek())){

                    postfix[j++] = s.pop();

                }

                s.push(curr);

            }

            i++;

        }

        else{

            cout<<"Invalid character entered. Please enter correct character! "<<endl;       // invalid character validation

            return;

        }

    }

    while(!s.isEmpty()){

        postfix[j++]=s.pop();               // Popping rest of the elements from stack

    }

    postfix[j]='\0';

}

// Function to convert infix to prefix

void Conversion::infixToPrefix(const char infix[]){

    Stack <char>s;

    int i = 0;

    int j = 0;

    char rev[1000];

    int size = strlen(infix);

    for(int k=size-1; k>=0; k--){                // Reverse the infix expression

        rev[j] = infix[k];

        j++;

    }

    rev[j]='\0';

    j=0;

    // Iterate through the reversed infix expression

    while(rev[i] != '\0'){

        char curr = rev[i];

        if(curr == ')'){

            s.push(curr);

            i++;

        }

        else if(curr == '('){                                       // Pop operators from the stack and add to the prefix expression until '(' is encountered

            while(!s.isEmpty() && s.peek()!=')'){

                prefix[j++] = s.pop();

            }

            if(!s.isEmpty() && s.peek()==')'){

                s.pop();                                            // Pop '(' from the stack

            }

            i++;

        }

        else if(isOperand(curr)){                                   // If the current character is an operand, add it to the prefix expression

            prefix[j++]=curr;

            i++;

        }

        else if(isOperator(curr)){                                  // If the current character is an operator

            if(s.isEmpty()){

                s.push(curr);

            }

            else if(icp(curr)>=isp(s.peek())){

                s.push(curr);

            }

            else{

                while((!s.isEmpty()) && icp(curr)<isp(s.peek())){   // Pop operators with higher precedence from the stack and add to the prefix

                    prefix[j++] = s.pop();

                }

                s.push(curr);

            }

            i++;

        }

        else{

            cout<<"Invalid character entered. Please enter correct character! "<<endl;       // invalid character validation

            return;

        }

    }

    while(!s.isEmpty()){                                            // Pop any remaining operators from the stack and add to the prefix expression

        prefix[j++] = s.pop();

    }

    postfix[j] = '\0';

    char finalPrefix[1000];                                         // Reverse the prefix expression to get the final result

    int k=0;

    for (int i=j-1; i>=0; i--){

        finalPrefix[k]=prefix[i];

        k++;

    }

    finalPrefix[k]='\0';

    strcpy(prefix, finalPrefix);

}

int Conversion::evaluatePostfix(){

    int i = 0;

    Stack <char>s1;

    for(int k=0; k<26; k++){                                    // Initialize variables array

        variables[k].vName=' ';

        variables[k].val=0;

    }

    while (postfix[i] != '\0'){

        char curr=postfix[i];

        if (isOperand(curr)){                                    // If the current character is an operand, get its value from the user

            int index=curr-'a';

            int value;

            cout<<"Enter value for variable " <<curr<< ": ";

            cin>>value;

            variables[index].vName = curr;

            variables[index].val = value;

            s1.push(value);

        }

        else if (isOperator(curr)){                              // If the current character is an operator, perform the operation

            int op2 = s1.pop();

            int op1 = s1.pop();

            int result;

            switch(curr){

                case '+':

                    result=op1+op2;

                    break;

                case '-':

                    result=op1-op2;

                    break;

                case '\*':

                    result=op1\*op2;

                    break;

                case '/':

                    if(op2 != 0){

                        result = op1/op2;

                    }

                    else{

                        cout << "Division by zero error." << endl;

                        return 0;

                    }

                    break;

            }

            s1.push(result);

        }                               // Push the result back to the stack

        i++;

    }

    if (!s1.isEmpty()){

        return s1.pop();            // Return the final result

    }

    return 0;

}

// Function to evaluate a prefix expression

int Conversion::evaluatePrefix(){

    int i = 0;

    Stack <char>s1;

    for (int k=0; k<26; k++){

        variables[k].vName=' ';

        variables[k].val=0;

    }

    int len=strlen(prefix);

    for(i=len-1; i>=0; i--){

        char curr = prefix[i];

        if(isOperand(curr)){

            int index=curr-'a';

            int value;

            cout<<"Enter value for variable "<<curr<<": ";

            cin>>value;

            variables[index].vName = curr;

            variables[index].val = value;

            s1.push(value);

        }

        else if (isOperator(curr)){

            int op1=s1.pop();

            int op2=s1.pop();

            int result;

            switch(curr){

                case '+':

                    result=op1+op2;

                    break;

                case '-':

                    result=op1-op2;

                    break;

                case '\*':

                    result=op1\*op2;

                    break;

                case '/':

                    if(op2 != 0){

                        result = op1/op2;

                    }

                    else{

                        cout<<"Division by zero error."<<endl;

                        return 0;

                    }

                    break;

            }

            s1.push(result);

        }

    }

    if(!s1.isEmpty()){

        return s1.pop();

    }

    return 0;

}

int main() {

    Conversion c;

    char infix[1000];

    int j = 0;

    while (true) {

        cout << "Choose an operation:" << endl;

        cout << "1. Enter Infix Expression " << endl;

        cout << "2. Convert to Postfix" << endl;

        cout << "3. Convert to Prefix" << endl;

        cout << "4. Evaluate Postfix" << endl;

        cout << "5. Evaluate Prefix" << endl;

        cout << "6. Exit" << endl;

        int choice;

        cin>>choice;

        cin.ignore();

        switch (choice) {

            case 1:

                cout<<"Enter infix expression: ";

                cin.getline(infix, 1000);

                break;

            case 2:

                if(strlen(infix)==0){

                    cout<<"No infix expression entered. Please enter an infix expression first."<<endl;

                }

                else{

                    c.infixToPostfix(infix);

                    cout<<"Postfix expression: "<<c.postfix<<endl;

                }

                break;

            case 3:

                if(strlen(infix)==0) {

                    cout<<"No infix expression entered. Please enter an infix expression first." << endl;

                }

                else{

                    c.infixToPrefix(infix);

                    cout<<"Prefix expression: "<<c.prefix<<endl;

                }

                break;

            case 4:

                if(strlen(infix)==0){

                    cout<<"No infix expression entered. Please enter an infix expression first." <<endl;

                }

                else{

                    c.infixToPostfix(infix);

                    cout<<"Postfix expression: "<<c.postfix<<endl;

                    cout<<"Postfix evaluation result: "<<c.evaluatePostfix()<<endl;

                }

                break;

            case 5:

                if(strlen(infix)==0){

                    cout<<"No infix expression entered. Please enter an infix expression first."<<endl;

                }

                else{

                    c.infixToPrefix(infix);

                    cout<<"Prefix expression: "<<c.prefix<<endl;

                    cout<<"Prefix evaluation result: "<<c.evaluatePrefix()<<endl;

                }

                break;

            case 6:

                return 0;

            default:

                cout<<"Invalid choice. Please try again."<<endl;

        }

    }

    return 0;

}