#include <iostream>

#define max 30

using namespace std;

typedef struct node

{

char data;

struct node \*next;

} node;

class Stack

{

node \*top;

public:

Stack()

{

top = NULL;

} int isempty()

{

if (top == NULL)

return 1;

return 0;

}

void push(char x)

{

node \*n;

n = new node();

n->data = x;

n->next = top;

top = n;

}

char pop()

{

node \*n;

char x;

n = top;

x = n->data;

top = top->next;

delete (n);

return x;

}

char topdata()

{

return top->data;

}

};

void infix\_postfix(char infix[], char postfix[]);

void reverse(char a[], char b[]);

void infix\_prefix(char infix[], char prefix[]);

int evaluate(int op1, int op2, char op);

void evaluate\_postfix(char postfix[]);

int precedence(char x);

void evaluate\_prefix(char prefix[]);

int main()

{

char infix[20], tok, postfix[20], prefix[20];

int ch, result;

do

{

cout <<endl<< "1. Infix to Postfix "<<endl;

cout<< "2. Infix to Prefix "<<endl;

cout<< "3. Evaluate Postfix "<<endl;

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

cout<< "5. Exit"<<endl;

cout<< "Enter your choice: ";

cin >> ch;

switch (ch)

{

case 1:

cout<<endl<<"Enter Infix expression :";

cin >> infix;

infix\_postfix(infix, postfix);

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

break;

case 2:

cout<< "Enter infix expression :";

cin >> infix;

infix\_prefix(infix, prefix);

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

break;

case 3:

evaluate\_postfix(postfix);

break;

case 4:

evaluate\_prefix(prefix);

break;

case 5: break;

default: cout<<endl<<"Invalid Choice !!! "<<endl;

break;

}

} while (ch != 5);

return 0;

}

void infix\_postfix(char infix[20], char postfix[20])

{

Stack s;

int i, j = 0;

char tok, x;

for (i = 0; infix[i] != '\0'; i++)

{

tok = infix[i];

if (isalnum(tok))

{

postfix[j] = tok;

j++;

}

else

{

if (tok == '(')

s.push(tok);

else if (tok == ')')

{

while ((x = s.pop()) != '(')

{

postfix[j] = x;

j++;

}

}

else

{

while (s.isempty() != 1 && precedence(tok) <= precedence(s.topdata()))

{

postfix[j] = s.pop();

j++;

}

s.push(tok);

}

}

}

while (s.isempty() != 1)

{

postfix[j] = s.pop();

j++;

}

postfix[j] = '\0';

}

void infix\_postfix1(char infix[20], char postfix[20])

{

Stack s;

int i, j = 0;

char tok, x;

for (i = 0; infix[i] != '\0'; i++)

{

tok = infix[i];

if (isalnum(tok))

{

postfix[j] = tok;

j++;

}

else

{

if (tok == '(')

s.push(tok);

else if (tok == ')')

{

while ((x = s.pop()) != '(')

{

postfix[j] = x;

j++;

}

}

else

{

while (s.isempty() != 1 && precedence(tok) <precedence(s.topdata()))

{

postfix[j] = s.pop();

j++;

}

s.push(tok);

}

}

}

while (s.isempty() != 1)

{

postfix[j] = s.pop();

j++;

}

postfix[j] = '\0';

}

void reverse(char a[20], char b[20])

{

int i, j = 0;

for (i = 0; a[i] != '\0'; i++)

{

}

i--;

for (j = 0; i >= 0; j++, i--)

{

if (a[i] == '(')

b[j] = ')';

else if (a[i] == ')')

b[j] = '(';

else

b[j] = a[i];

}

b[j] = '\0';

}

void infix\_prefix(char infix[20], char prefix[20])

{

char prefix1[20], infix1[20];

reverse(infix, infix1);

infix\_postfix1(infix1, prefix1);

reverse(prefix1, prefix);

}

int precedence(char x)

{

if (x == '(')

return 0;

if (x == '+' || x == '-')

return 1;

if (x == '\*' || x == '/')

return 2;

return 3;

}

int evaluate(int op1, int op2, char op)

{

if (op == '+')

return op1 + op2;

if (op == '-')

return op1 - op2;

if (op == '\*')

return op1 \* op2;

if (op == '/')

return op1 / op2;

if (op == '%')

return op1 % op2;

return 0;

}

void evaluate\_postfix(char postfix[20])

{

Stack s;

int i, op1, op2, result;

char tok;

int x;

for (i = 0; postfix[i] != '\0'; i++)

{

tok = postfix[i];

if (isalnum(tok))

{

cout <<"Enter "<< tok<<": ";

cin >> x;

s.push(char(x));

}

else

{

op2 = s.pop();

op1 = s.pop();

result = evaluate(op1, op2, tok);

s.push(char(result));

}

}

result = s.pop();

cout <<endl<<"Simplified value of postfix expression is : "<<result<<endl;

}

void evaluate\_prefix(char prefix[20])

{

Stack s;

int i, op1, op2, result;

char tok;

int x;

for (i = 0; prefix[i] != '\0'; i++)

{

}

i--;

for (; i >= 0; i--)

{

tok = prefix[i];

if (isalnum(tok))

{

cout << "Enter " << tok<<" : ";

cin >> x;

s.push(char(x));

}

else

{

op1 = s.pop();

op2 = s.pop();

result = evaluate(op1, op2, tok);

s.push(char(result));

}

}

result = s.pop();

cout << "Simplified value of prefix epression is : " << result<<endl;

}