                                            Lab Experiment No.2

-----------------------------------------------------------------------------------------------------------------------------

Name: Prathamesh Sadashiv Gadekar

Roll No.:14

Class : SE-IT

-----------------------------------------------------------------------------------------------------------------------------

Problem Statement:

Implement stack as an abstract data type using singly linked list and use this ADT for conversion of infix expression to postfix, prefix and evaluation of postfix and prefix expression.

-----------------------------------------------------------------------------------------------------------------------------

#include<stdlib.h>

#include<iostream>

#include<string.h>

#define max 30

using namespace std;

struct node

{

    char data;

    struct node\* next;

};

class stack

{

    node\* top;

    char x;

public:

    stack()

    {

        top = NULL;

    }

    int empty()

    {

        if (top == NULL)

        {

            return(1);

        }

        else

        {

            return(0);

        }

    }

    void push(char x)

    {

        node\* p;

        p = new node;

        p->data = x;

        p->next = top;

        top = p;

    }

    char pop()

    {

        if (!empty())

        {

            node\* p;

            p = new node;

            p = top;

            top = top->next;

            x = p->data;

            delete p;

            return(x);

        }

        else

        {

            return(0);

        }

    }

    char pop1()

    {

        if (!empty())

        {

            node\* p;

            p = new node;

            p = top;

            x = p->data;

            return(x);

        }

        else

        {

            return(0);

        }

    }

};

int precedence(char x);

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

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

void eval\_prefix(char prefix[]);

void eval\_postfix(char postfix[]);

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

int main()

{

    char infix[30], prefix[30], postfix[30];

    int op;

    do {

        cout<<endl<<"\t\t"<<"MAIN MENU";

        cout<<endl;

        cout<<endl<<"\t"<<"1. Infix to Prefix with Evaluation        ";

        cout<<endl<<"\t"<<"2. Infix to Postfix with Evaluation       ";

        cout<<endl<<"\t"<<"3. Exit                                   ";

        cout<<endl<<"Enter your choice ";

        cin>>op;

        cout<<"........................................................."<<endl;

       switch (op)

        {

        case 1:

            cout << "\nEnter the infix expression::" << endl;

                cin >> infix;

            infix\_to\_prefix(infix, prefix);

            cout << "\nPrefix expression is " << prefix << endl;

            cout << "\n\nEvaluation of Prefix expression";

            eval\_prefix(prefix);

            break;

        case 2:

            cout << "\nEnter the infix expression::" << endl;

                cin >> infix;

            infix\_to\_postfix(infix, postfix);

            cout << "\nPostfix expression is " << postfix << endl;

            cout << "\n\nEvaluation of Postfix expression";

            eval\_postfix(postfix);

            break;

        }

    } while (op != 3);

    return 0;

}

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

{

    int i, j;

    char temp, in1[30];

    cout << "\n\nEntered infix is...";

    for (i = 0; i <= strlen(infix) - 1; i++)

    {

        cout << infix[i];

    }

    for (i = strlen(infix) - 1, j = 0; i >= 0; i--, j++)

        in1[j] = infix[i];

    in1[j] = '\0';

   for (i = 0; in1[i] != '\0'; i++)

    {

        if (in1[i] == '(')

            in1[i] = ')';

        else

            if (in1[i] == ')')

                in1[i] = '(';

    }

    infix\_to\_postfix(in1, prefix);

    for (i = 0, j = strlen(prefix) - 1; i < j; i++, j--)

    {

        temp = prefix[i];

        prefix[i] = prefix[j];

        prefix[j] = temp;

    }

}

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

{

    stack s;

    node\* top;

    char x;

    int i, j;

    char token;

    j = 0;

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

    {

        token = infix[i];

        if (isalnum(token))

        {

            postfix[j++] = token;

        }

        else if (token == '(')

        {

            s.push('(');

        }

        else if (token == ')')

        {

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

            {

                postfix[j++] = x;

            }

        }

        else

        {

            x = s.pop1();

            while (precedence(token) < precedence(x) && !s.empty())

            {

                x = s.pop();

                postfix[j++] = x;

            }

            s.push(token);

        }

    }

    while (!s.empty())

    {

        x = s.pop();

        postfix[j++] = x;

    }

    postfix[j] = '\0';

}

int precedence(char x)

{

    if (x == '(')

    {

        return(0);

    }

    if (x == '+' || x == '-')

    {

        return(1);

    }

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

    {

        return(2);

    }

    return(3);

}

void eval\_prefix(char prefix[])

{

    stack s;

    char x;

    int op1, op2, val, i;

    for (i = strlen(prefix) - 1; i >= 0; i--)

    {

        x = prefix[i];

        if (isalpha(x))

        {

            cout << "\nEnter the value of " << x << ":";

            cin >> val;

            s.push(val);

        }

        else

        {

            op1 = s.pop();

            op2 = s.pop();

            val = evaluate(x, op1, op2);

            s.push(val);

        }

    }

    val = s.pop();

    cout << "\nValue of expression is " << val;

}

void eval\_postfix(char postfix[])

{

    stack s;

    char x;

    int op1, op2, val, i;

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

    {

        x = postfix[i];

        if (isalpha(x))

        {

            cout << "\nEnter the value of " << x << ":";

            cin >> val;

            s.push(val);

        }

        else

        {

            op2 = s.pop();

            op1 = s.pop();

            val = evaluate(x, op1, op2);

            s.push(val);

        }

    }

    val = s.pop();

    cout << "\nValue of expression is " << val;

}

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

{

    if (x == '+')

        return(op1 + op2);

    if (x == '-')

        return(op1 - op2);

    if (x == '\*')

        return(op1 \* op2);

    if (x == '/')

        return(op1 / op2);

    if (x == '%')

        return(op1 % op2);

}

Value of expression is 96

        MAIN MENU

    1. Infix to Prefix with Evaluation

    2. Infix to Postfix with Evaluation

    3. Exit

Enter your choice 1

.........................................................

Enter the infix expression::

b\*c/(a-b)\*d

Entered infix is...b\*c/(a-b)\*d

Prefix expression is \*/\*bc-abd

Evaluation of Prefix expression

Enter the value of d:9

Enter the value of b:8

Enter the value of a:7

Enter the value of c:6

Enter the value of b:8

Value of expression is 80

        MAIN MENU

    1. Infix to Prefix with Evaluation

    2. Infix to Postfix with Evaluation

    3. Exit

Enter your choice 2

.........................................................

Enter the infix expression::

b\*a/(d-c)

Postfix expression is badc-/\*

Evaluation of Postfix expression

Enter the value of b:9

Enter the value of a:8

Enter the value of d:7

Enter the value of c:6

Value of expression is 72

        MAIN MENU

    1. Infix to Prefix with Evaluation

    2. Infix to Postfix with Evaluation

    3. Exit

Enter your choice 3

.........................................................

(base) rmdstic@rmdstic-OptiPlex-3020:~$