* **StackAdt.h**

/\*

\* StackAdt.h

\*Created on: Oct 21, 2020

\* Author: Megha Sonavane

\*/

**#ifndef** STACKADT\_H\_

**#define** STACKADT\_H\_

**template**<**typename** **T**>

**struct** Node{

**T** symbol;

Node<**T**>\*next;

};

//-------------------------------------class declaration------------------------------

**template**<**class** **T**>

**class** StackAdt {

Node<**T**>\*top;

**public**:

**StackAdt**();

**bool** **isEmpty**();

**void** **push**(**T**);

**T** **pop**();

**T** **peep**();

**void** **display**();

**~StackAdt**();

};

**#endif** /\* STACKADT\_H\_ \*/

* **StackAdt.cpp**

/\*

\* StackAdt.cpp

\*

\* Created on: Oct 21, 2020

\* Author: Megha Sonavane

\*/

**#include**<iostream>

**#include**<cstdlib>

**#include** "StackAdt.h"

**using** **namespace** std;

**template**<**typename** **T**>

**StackAdt<T>::StackAdt**() {

top=NULL;

}

//--------------------------------------definition of isEmpty-------------------------------------------------

**template**<**typename** **T**>

**bool** **StackAdt<T>::isEmpty**(){

**if**(top==NULL)

**return** **true**;

**return** **false**;

}

//-----------------------------------definition of push method---------------------------------------------------

**template**<**class** **T**>

**void** **StackAdt<T>::push**(**T** symbol)

{

Node<**T**>\*ptr=**new** Node<**T**>;

ptr->symbol=symbol;

ptr->next=NULL;

//if it is first node

**if**(top==NULL)

{

top=ptr;

}

**else**{

ptr->next=top;

top=ptr;

}

}

//------------------------------------definition of pop method--------------------------------------------------------------

**template**<**class** **T**>

**T** **StackAdt<T>::pop**(){

**T** c=top->symbol;

top=top->next;

**return** c;

}

//-----------------------------------definition of peek method--------------------------------------------------------------

**template**<**class** **T**>

**T** **StackAdt<T>::peep**(){

**return** top->symbol;

}

//-----------------------------------definition of display---------------------------------------------------------------

**template**<**class** **T**>

**void** **StackAdt<T>::display**(){

Node<**T**>\*temp;

temp=top;

**while**(temp!=NULL){

cout<<temp->symbol;

temp=temp->next;

}

}

**template**<**class** **T**>

**StackAdt<T>::~StackAdt**() {

// **TODO** Auto-generated destructor stub

}

* **Ass2Template.cpp**

//============================================================================

// Name : Ass2Template.cpp

// Author : Megha Sonavane

// Description : Expression Conversion and evalution

//============================================================================

**#include** <iostream>

**#include**<cmath>

**#include** "StackAdt.h"

**#include** "StackAdt.cpp"

**using** **namespace** std;

**class** Convert{

**public**:

**bool** **isOperator**(**char**);

string **toPostfix**(string);

string **toPrefix**(string);

**double** **evaluate\_postfix**(string);

**double** **evaluate\_prefix**(string);

**double** **evaluate**(**double**,**double**,**char**);

**int** **precedence**(**char**);

};

//-----------------------------definition of isOperator method---------------------------

**bool** **Convert::isOperator**(**char** c){

**if**(c=='+'||c=='-'||c=='\*'||c=='/'||c=='^')

**return** **true**;

**return** **false**;

}

//----------------------------------definition to check precedence of operator----------------

**int** **Convert::precedence**(**char** c)

{

**if**(c=='^')

**return** 3;

**else** **if**(c=='\*'||c=='/')

**return** 2;

**else** **if**(c=='+'||c=='-')

**return** 1;

**return** -1;

}

//--------------------------------------definition of method to convert expression into postfix-------------------

string **Convert::toPostfix**(string infix){

StackAdt<**char**>s;

string postfix="";

**int** len=infix.length();

cout<<"------------------------------------------------------------------------------------------------"<<**endl**;

cout<<"\tConversion:"<<**endl**<<"Scan"<<"\t"<<"Stack"<<"\t"<<"Expression"<<**endl**;

**for**(**int** i=0;i<len;i++)

{

//-----------1.If it is operand----------------------

**if**(**isalpha**(infix[i]))

postfix+=infix[i];

//---------2.If it is (---------------------------

**else** **if**(infix[i]=='(')

s.push(infix[i]);

//--------3.If it is )------------------------------

**else** **if**(infix[i]==')')

{

**while**((s.peep()!='(')&&(!s.isEmpty()))

{

postfix+=s.pop();

}

**if**(s.peep()=='(')

s.pop();

}

//-----------4.If it is operator

**else** **if**(isOperator(infix[i]))

{

//----4.1.If stack is empty or contains ( at top---

**if**((s.isEmpty())||(s.peep()=='('))

s.push(infix[i]);

//---4.2.If precedence of operator in expression is greater than that of operator in stack---

**else** **if**(precedence(infix[i])>precedence(s.peep()))

s.push(infix[i]);

//---4.3. If the precedence of operator in expression is smaller than that of operator in stack---

**else**{

**while**((!s.isEmpty())&&( precedence(infix[i])<=precedence(s.peep())))

{

postfix+=s.pop();

}

s.push(infix[i]);

}

}

//------------else the expression is invalid------------------------

/\*else{

cout<<"\t\*\*\*Invalid expression\*\*\*"<<endl;

}\*/

//-----------------display symbol scanned, current status of stack and expression------------------------

cout<<infix[i]<<"\t";

s.display();

cout<<"\t"<<postfix<<**endl**;

}

**while**(!s.isEmpty())

postfix+=s.pop();

cout<<"------------------------------------------------------------------------------------------------"<<**endl**;

**return** postfix;

}

//-------------------------------------------------definition of method to convert into prefix------------------------------

string **Convert::toPrefix**(string infix)

{

StackAdt<**char**>s;

string reverse="";

string prefix="";

**int** len=infix.length();

**for**(**int** i=len-1;i>=0;i--)

{

**if**(infix[i]=='(')

infix[i]=')';

**else** **if**(infix[i]==')')

infix[i]='(';

reverse+=infix[i];

}

reverse=toPostfix(reverse);

**for**(**int** i=reverse.length()-1;i>=0;i--)

prefix+=reverse[i];

**return** prefix;

}

//-----------------------------------------------definition of evaluate method--------------------------------------------------

**double** **Convert::evaluate**(**double** a, **double** b,**char** op){

**switch**(op){

**case** '+':

**return** (a+b);

**break**;

**case** '-':

**return** (a-b);

**break**;

**case** '\*':

**return** (a\*b);

**break**;

**case** '/':

**return** (a/b);

**break**;

**case** '^':

**return** (**pow**(a,b));

**break**;

**default**:

cout<<"\*\*\*\*Invalid values\*\*\*";

**return** 0;

}

}

//-----------------------------------------------definition of postfix expression evaluation----------------------------------------------

**double** **Convert::evaluate\_postfix**(string exp)

{

**double** result;

StackAdt<**double**>s;

**double** op1,op2,val;

**int** len=exp.length();

**for**(**int** i=0;i<len;i++)

{

**if**(**isalpha**(exp[i]))

{

cout<<"Enter value of "<<exp[i]<<":";

cin>>val;

s.push(val);

}

**else**{

op2=s.pop();

op1=s.pop();

result=evaluate(op1,op2,exp[i]);

s.push(result);

}

}

**return** result;

}

//-----------------------------definition of prefix expression evaluation----------------------------------------------------------

**double** **Convert::evaluate\_prefix**(string exp){

**double** result;

StackAdt<**double**>s;

**double** op1,op2,val;

**for**(**int** i=exp.length()-1;i>=0;i--){

**if**(**isalpha**(exp[i])){

cout<<"Enter value of "<<exp[i]<<":";

cin>>val;

s.push(val);

}

**else**{

op1=s.pop();

op2=s.pop();

result=evaluate(op1,op2,exp[i]);

s.push(result);

}

}

**return** result;

}

**int** **main**() {

Convert c;

string infix,postfix,prefix;

**double** result;

**int** choice;

cout<<"\tEnter infix expression:";

cin>>infix;

**do**{

cout<<"----------------------------------------------------------------------------------------------------------"<<**endl**;

cout<<"\t1:To prefix"<<**endl**<<"\t2:To postfix"<<**endl**<<"\t3:Evaluate postfix"<<**endl**<<"\t4:Evaluate prefix"<<**endl**<<"\t5:New expression"<<**endl**<<"\t6:Exit"<<**endl**;

cout<<"\tEnter choice:";

cin>>choice;

**switch**(choice){

**case** 1:

prefix=c.toPrefix(infix);

cout<<"\tPrefix expression:"<<prefix<<**endl**;

**break**;

**case** 2:

postfix=c.toPostfix(infix);

cout<<"\tPostfix expression:"<<postfix<<**endl**;

**break**;

**case** 3:

postfix=c.toPostfix(infix);

result=c.evaluate\_postfix(postfix);

cout<<"\tResult:"<<result<<**endl**;

**break**;

**case** 4:

prefix=c.toPrefix(infix);

result=c.evaluate\_prefix(prefix);

cout<<"\tResult:"<<result<<**endl**;

**break**;

**case** 5:

cout<<"\tEnter infix expression:";

cin>>infix;

**break**;

**case** 6:

cout<<"\tThank you..";

**break**;

**default**:

cout<<"\tEnter valid choice..."<<**endl**;

**break**;

}

}**while**(choice!=6);

**return** 0;

}

* **Output:**

Enter infix expression:(a+b)\*(c+d)

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

1:To prefix

2:To postfix

3:Evaluate postfix

4:Evaluate prefix

5:New expression

6:Exit

Enter choice:1

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

Conversion:

Scan Stack Expression

( (

d ( d

+ +( d

c +( dc

) dc+

\* \* dc+

( (\* dc+

b (\* dc+b

+ +(\* dc+b

a +(\* dc+ba

) \* dc+ba+

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

Prefix expression:\*+ab+cd

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

1:To prefix

2:To postfix

3:Evaluate postfix

4:Evaluate prefix

5:New expression

6:Exit

Enter choice:2

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

Conversion:

Scan Stack Expression

( (

a ( a

+ +( a

b +( ab

) ab+

\* \* ab+

( (\* ab+

c (\* ab+c

+ +(\* ab+c

d +(\* ab+cd

) \* ab+cd+

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

Postfix expression:ab+cd+\*

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

1:To prefix

2:To postfix

3:Evaluate postfix

4:Evaluate prefix

5:New expression

6:Exit

Enter choice:1

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

Conversion:

Scan Stack Expression

( (

d ( d

+ +( d

c +( dc

) dc+

\* \* dc+

( (\* dc+

b (\* dc+b

+ +(\* dc+b

a +(\* dc+ba

) \* dc+ba+

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

Prefix expression:\*+ab+cd

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

1:To prefix

2:To postfix

3:Evaluate postfix

4:Evaluate prefix

5:New expression

6:Exit

Enter choice:12

Enter valid choice...

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

1:To prefix

2:To postfix

3:Evaluate postfix

4:Evaluate prefix

5:New expression

6:Exit

Enter choice:2

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

Conversion:

Scan Stack Expression

( (

a ( a

+ +( a

b +( ab

) ab+

\* \* ab+

( (\* ab+

c (\* ab+c

+ +(\* ab+c

d +(\* ab+cd

) \* ab+cd+

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

Postfix expression:ab+cd+\*

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

1:To prefix

2:To postfix

3:Evaluate postfix

4:Evaluate prefix

5:New expression

6:Exit

Enter choice:3

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

Conversion:

Scan Stack Expression

( (

a ( a

+ +( a

b +( ab

) ab+

\* \* ab+

( (\* ab+

c (\* ab+c

+ +(\* ab+c

d +(\* ab+cd

) \* ab+cd+

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

Enter value of a:12

Enter value of b:3

Enter value of c:10

Enter value of d:5

Result:225

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

1:To prefix

2:To postfix

3:Evaluate postfix

4:Evaluate prefix

5:New expression

6:Exit

Enter choice:4

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

Conversion:

Scan Stack Expression

( (

d ( d

+ +( d

c +( dc

) dc+

\* \* dc+

( (\* dc+

b (\* dc+b

+ +(\* dc+b

a +(\* dc+ba

) \* dc+ba+

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

Enter value of d:5

Enter value of c:10

Enter value of b:3

Enter value of a:12

Result:225

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

1:To prefix

2:To postfix

3:Evaluate postfix

4:Evaluate prefix

5:New expression

6:Exit

Enter choice:6

Thank you..