#ifndef NODE\_H\_INCLUDED

#define NODE\_H\_INCLUDED

#include<iostream>

using namespace std ;

template<class T>

class Node

{

public:

Node(T);//set nextPtr to null and initialize the data with the value passed to constructor

T getData()const;//return the value stored in the data variable

Node<T>\* getNextPtr()const;//return the address stored in nextPtr variable

void setData(T);//set the passed value to the data variable

void setNextPtr(Node\*);//set the passed value to the nextPtr variable

private:

T data;

Node<T>\* nextPtr;

};

#endif

====================================

using namespace std;

#include<iostream>

#include "node.h"

template<class T>

Node<T> :: Node (T a)

{

data = a;

nextPtr = NULL;

}

template<class T>

void Node<T> :: setData(T a)

{

data = a;

}

template<class T>

void Node<T> :: setNextPtr (Node<T> \*p)

{

nextPtr = p;

}

template<class T>

T Node<T> :: getData() const

{

return data;

}

template<class T>

Node<T> \* Node<T> :: getNextPtr() const

{

return nextPtr;

}

Sack.h

#ifndef STACK\_H\_INCLUDED

#define STACK\_H\_INCLUDED

#include<iostream>

using namespace std;

#include "node.h"

template<class T>

class Stack

{

private:

Node<T> \*head;

public:

Stack();//set head to null

~Stack();// ???

void push(T);//add a new node at the start of stack with data provided as argument

T pop();//return the data stored in first node and delete the first node

T showTop();//return the data stored in first node

bool isEmpty();//return true if there is no node in stack

};

#endif

Stack.cpp

using namespace std;

#include<iostream>

#include "stack.h"

template<class T>

Stack<T> :: Stack()

{

head = NULL;

}

template<class T>

Stack<T> ::~Stack()

{

Node<T>\* current = head;

while(current!= NULL)

{

Node<T> \*temp = current;

current = current->getNextPtr();

delete temp;

}

}

template<class T>

void Stack<T> :: push(T a)

{

Node<T> \*temp = new Node<T>(a);

temp->setNextPtr(head);

head = temp;

}

template<class T>

T Stack<T> :: pop()

{

if(head != NULL)

{

T temp\_data = head->getData();

Node<T> \*temp = head;

head = head->getNextPtr();

delete temp;

return temp\_data;

}

}

template<class T>

T Stack<T> :: showTop()

{

if(head != NULL)

{

return head->getData();

}

}

template<class T>

bool Stack<T> :: isEmpty()

{

if(head != NULL)

{

return false;

}

else

{

return true;

}

}

Main.cpp

using namespace std;

#include<iostream>

#include <cstring>

#include <cmath>

#include "stack.h"

#include "stack.cpp"

#include "node.cpp"

#include "node.h"

bool prcd(char a, char b)

{

if(a == '(') return false;

if( (a != '(') && (b == ')') ) return true;

else return false;

if ( ( (a == '\*') || (a == '/') ) &&

( (b == '+') || (b == '-') ))

return true;

return false;

}

string infix\_to\_postfix(string eq)

{

Stack<char> s;

const char \*input = eq.c\_str();

int input\_length = eq.length();

string output = "";

for(int i=0; i<input\_length; i++)

{

if((input[i] != '+') &&

(input[i] != '-') &&

(input[i] != '\*') &&

(input[i] != '/') &&

(input[i] != '^') &&

(input[i] != '(') &&

(input[i] != ')'))

{

output += input[i];

}

else

{

while( !s.isEmpty() && prcd(s.showTop(), input[i]) )

{

output += s.pop();

}

if(s.isEmpty() || input[i] != ')')

s.push(input[i]);

else

s.pop();

}

}

while(!s.isEmpty())

{

output += s.pop();

}

return output;

}

double evaluate\_postfix(string eq)

{

Stack<double> s;

const char \*input = eq.c\_str();

int input\_length = eq.length();

for(int i=0; i<input\_length; i++)

{

if((input[i] != '+') &&

(input[i] != '-') &&

(input[i] != '\*') &&

(input[i] != '/') &&

(input[i] != '^'))

{

s.push(input[i]-48);

}

else

{

double b = s.pop();

double a = s.pop();

if(input[i] == '+')

{

s.push(a+b);

}

else if(input[i] == '-')

{

s.push(a-b);

}

else if(input[i] == '\*')

{

s.push(a\*b);

}

else if(input[i] == '/')

{

s.push(a/b);

}

else if(input[i] == '^')

{

s.push(pow(a,b));

}

}

}

return s.pop();

}

int main()

{

cout << evaluate\_postfix("623+-382/+\*2^3+") << endl;

cout << infix\_to\_postfix("((4+5)\*8)\*3");

return 0;

}

================

Conversion to Postfix

=======================

#include<iostream>

#include<stack>

using namespace std;

// defines the Boolean function for operator, operand, equalOrhigher precedence and the string conversion function.

bool IsOperator(char);

bool IsOperand(char);

bool eqlOrhigher(char, char);

string convert(string);

main()

{

string infix\_expression, postfix\_expression;

int ch;

do

{

cout << " Enter an infix expression: ";

cin >> infix\_expression;

postfix\_expression = convert(infix\_expression);

cout << "\n Your Infix expression is: " << infix\_expression;

cout << "\n Postfix expression is: " << postfix\_expression;

cout << "\n \t Do you want to enter infix expression (1/ 0)?";

cin >> ch;

//cin.ignore();

} while(ch == 1);

return 0;

}

// define the IsOperator() function to validate whether any symbol is operator.

/\* If the symbol is operator, it returns true, otherwise false. \*/

bool IsOperator(char c)

{

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

return true;

return false;

}

// IsOperand() function is used to validate whether the character is operand.

bool IsOperand(char c)

{

if( c >= 'A' && c <= 'Z') /\* Define the character in between A to Z. If not, it returns False.\*/

return true;

if (c >= 'a' && c <= 'z') // Define the character in between a to z. If not, it returns False. \*/

return true;

if(c >= '0' && c <= '9') // Define the character in between 0 to 9. If not, it returns False. \*/

return true;

return false;

}

// here, precedence() function is used to define the precedence to the operator.

int precedence(char op)

{

if(op == '+' || op == '-') /\* it defines the lowest precedence \*/

return 1;

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

return 2;

if(op == '^')

return 3;

return 0;

}

/\* The eqlOrhigher() function is used to check the higher or equal precedence of the two operators in infix expression. \*/

bool eqlOrhigher (char op1, char op2)

{

int p1 = precedence(op1);

int p2 = precedence(op2);

if (p1 == p2)

{

if (op1 == '^' )

return false;

return true;

}

return (p1>p2 ? true : false);

}

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

string convert(string infix)

{

stack <char> S;

string postfix ="";

char ch;

S.push( '(' );

infix += ')';

for(int i = 0; i<infix.length(); i++)

{

ch = infix[i];

if(ch == ' ')

continue;

else if(ch == '(')

S.push(ch);

else if(IsOperand(ch))

postfix += ch;

else if(IsOperator(ch))

{

while(!S.empty() && eqlOrhigher(S.top(), ch))

{

postfix += S.top();

S.pop();

}

S.push(ch);

}

else if(ch == ')')

{

while(!S.empty() && S.top() != '(')

{

postfix += S.top();

S.pop();

}

S.pop();

}

}

return postfix;

}