**Actividad de Aprendizaje 04. Aplicación de Pila y Cola**

***Problema:***

Implemente un programa que reciba una cadena que contenga una expresión con notación infija y la convierta a su equivalente expresión con notación posfija, e imprima el resultado en pantalla.

***Requerimientos:***

a)      El estilo de programación debe ser Orientado a Objetos

b)      Las clases *Pila* y *Cola* deben tener una implementación estática (con arreglos).

c)       La cadena con la expresión infija se pasará a una Cola

d)      La conversión de infija a posfija debe hacerse con el método que utiliza una Pila

e)      La expresión resultante se pasará directamente a una Cola

f)       Los operadores a considerar son sólo binarios: suma (+), resta (-), multiplicación (\*), división (/), y potencia (^)

g)      Se incluye el manejo de paréntesis como agrupador

Christopher Ceballos Jiménez

219750442

Estructura de datos I

Fue algo complicado lograr entender a la perfección como funciona la notación infija a posfija. Pero una vez comprendida pude empezar a programar en base lo ejemplos en clase. Fue codificando de paso por paso pensando en las posibles situaciones, de esa forma logre completar el programa.

#include<iostream>  
  
#include "stack.h"  
#include "queue.h"  
  
  
**using** **namespace** std**;**  
  
int prioridadInfija**(**char**);**  
int prioridadPila**(**char**);**  
  
int main **()** **{**  
       
     Queue**<**char**>** myCola**;**  
     Stack**<**char**>** myPila**;**  
       
     char myChar**[**100**],** aux**;**  
     int op**;**  
       
       
     **do{**  
          system**(**"cls"**);**  
          cout **<<** "Enter an infix expression without spaces: "**;** cin **>>** myChar**;** ///a/b^(c+d)-e\*f/g^h  
            
          int i**(**0**);**  
            
          **while(** myChar**[**i**]** **!=** '\0'**){**  
                 
               **if** **((**myChar**[**i**]** **>=** 'a' **and** myChar**[**i**]** **<=** 'z'**)** **or** **(**myChar**[**i**]** **>=** 49 **and** myChar**[**i**]** **<=** 57**)){**  
                      
                    myCola**.**enqueue**(**myChar**[**i**]);**  
               **}**  
                 
               **if(**myChar**[**i**]** **==** '+' **or** myChar**[**i**]** **==** '-' **or** myChar**[**i**]** **==** '\*' **or** myChar**[**i**]** **==** '/' **or** myChar**[**i**]** **==** '^'**){**  
                      
                    **if** **(**myPila**.**isEmpty**()){**  
                           
                         myPila**.**push**(**myChar**[**i**]);**  
                           
                           
                    **}else{**  
                           
                         **if(!**myPila**.**isEmpty**()){**  
                                
                              **if((**prioridadInfija**(**myChar**[**i**])** **<** prioridadPila**(**myPila**.**getTop**()))** **or** **(**prioridadInfija**(**myChar**[**i**])** **==** prioridadPila**(**myPila**.**getTop**()))){**  
                                     
                                   aux **=** myPila**.**getTop**();**  
                                   myCola**.**enqueue**(**aux**);**  
                                   myPila**.**pop**();**  
                                   myPila**.**push**(**myChar**[**i**]);**  
                                     
                              **}else{**  
                                   myPila**.**push**(**myChar**[**i**]);**  
                              **}**  
                         **}**  
                    **}**  
               **}**  
                 
               **if(**myChar**[**i**]** **==** '('**){**  
                    myPila**.**push**(**myChar**[**i**]);**  
               **}**  
                 
               **if(**myChar**[**i**]** **==** ')'**){**  
                    **while(**myPila**.**getTop**()** **!=** '('**){**  
                           
                         aux **=** myPila**.**pop**();**  
                         myCola**.**enqueue**(**aux**);**  
                    **}**  
                    myPila**.**pop**();**  
               **}**  
                 
               i**++;**  
          **}**  
            
          cout **<<** endl **<<** endl**;**  
            
          **while(!**myCola**.**isEmpty**()){**  
               cout **<<** myCola**.**dequeue**();**  
          **}**  
            
          **while(!**myPila**.**isEmpty**()){**  
               cout **<<** myPila**.**getTop**();**  
               myPila**.**pop**();**  
                 
          **}**  
            
          cout **<<** "\n\n\tPress 1 to try again.\n\tPress 0 to exit.\n\n\t: "**;** cin **>>** op**;**  
     **}while(**op **!=** 0**);**  
**}**  
  
  
int prioridadInfija**(**char a**){**  
       
     **if(** a**==**'^'**){**  
          **return** 4**;**  
     **}**  
     **if(** a**==**'\*'**){**  
          **return** 2**;**  
     **}**  
     **if(** a**==**'/'**){**  
          **return** 2**;**  
     **}**  
     **if(** a**==**'+'**){**  
          **return** 1**;**  
     **}**  
     **if(** a**==**'-'**){**  
          **return** 1**;**  
     **}**  
     **if(**a**==**'('**){**  
          **return** 5**;**  
     **}**  
       
**}**  
       
       
int prioridadPila**(**char a**){**  
       
     **if(** a**==**'^'**){**  
          **return** 3**;**  
     **}**  
     **if(** a**==**'\*'**){**  
          **return** 2**;**  
     **}**  
     **if(** a**==**'/'**){**  
          **return** 2**;**  
     **}**  
     **if(** a**==**'+'**){**  
          **return** 1**;**  
     **}**  
     **if(** a**==**'-'**){**  
          **return** 1**;**  
     **}**  
     **if(** a**==**'('**){**  
          **return** 0**;**  
     **}**  
       
**}**

#ifndef STACK\_H  
#define STACK\_H  
  
#include <string>  
  
**template** **<class** T**,** int ARRAYSIZE **=** 1000**>** ///Definicion  
**class** Stack**{**  
**private:**  
       
     T data**[**ARRAYSIZE**];**  
     int top**;**  
       
     void copyAll**(const** Stack**&);**  
       
**public:**  
     Stack**();**  
     Stack**(const** Stack**&);**  
       
     bool isEmpty**();**  
     bool isFull**();**  
       
     void push**(const** T**&);**  
       
     T pop**();**  
       
     T getTop**()** **const;**  
       
       
     Stack**&** **operator** **=** **(const** Stack**&);**  
       
     bool **operator** **==** **(const** Stack**&)** **const;**  
     bool **operator** **!=** **(const** Stack**&)** **const;**  
     bool **operator** **<** **(const** Stack**&)** **const;**  
     bool **operator** **<=** **(const** Stack**&)** **const;**  
     bool **operator** **>** **(const** Stack**&)** **const;**  
     bool **operator** **>=** **(const** Stack**&)** **const;**  
**};**  
  
  
  
**template** **<class** T**,** int ARRAYSIZE**>**  
void Stack**<**T**,** ARRAYSIZE**>::**copyAll **(const** Stack**&** s**)** **{** ///Implementacion  
       
     int i**(**0**);**  
       
     **while** **(**i**<=** s**.**top**){**  
          **this->**data**[**i**]** **=** s**.**data**[**i**];**  
          i**++;**  
     **}**  
       
     **this** **->** top **=** s**.**top**;**  
       
     **return** **\*this;**  
**}**  
  
  
**template** **<class** T**,** int ARRAYSIZE**>**  
Stack**<**T**,** ARRAYSIZE**>::**Stack **(** **)** **:** top**(-**1**){** **}**  
  
  
**template** **<class** T**,** int ARRAYSIZE**>**  
Stack**<**T**,** ARRAYSIZE**>::**Stack **(const** Stack**&** s**)** **{**  
       
     copyAll**(**s**);**  
**}**  
  
  
**template** **<class** T**,** int ARRAYSIZE**>**  
bool Stack**<**T**,** ARRAYSIZE**>::**isEmpty **(** **)** **{**  
       
     **return** top **==** **-**1**;**  
**}**  
  
  
**template** **<class** T**,** int ARRAYSIZE**>**  
bool Stack**<**T**,** ARRAYSIZE**>::**isFull **(** **)** **{**  
       
     **return** top **==** ARRAYSIZE **-**1**;**  
**}**  
  
  
**template** **<class** T**,** int ARRAYSIZE**>**  
void Stack**<**T**,** ARRAYSIZE**>::**push **(const** T**&** e**)** **{**  
       
     data**[++**top**]** **=** e**;**  
**}**  
  
  
**template** **<class** T**,** int ARRAYSIZE**>**  
T Stack**<**T**,** ARRAYSIZE**>::**pop **(** **){**  
       
     **return** data**[**top**--];**  
**}**  
  
  
**template** **<class** T**,** int ARRAYSIZE**>**  
T Stack**<**T**,** ARRAYSIZE**>::**getTop **(** **)** **const** **{**  
       
     **return** data**[**top**];**  
**}**  
  
  
**template** **<class** T**,** int ARRAYSIZE**>**  
Stack**<**T**,** ARRAYSIZE**>&** Stack**<**T**,** ARRAYSIZE**>::operator** **=** **(const** Stack**&** s**)** **{**  
       
     copyAll**(**s**);**  
       
     **return** **\*this;**  
**}**  
  
  
**template** **<class** T**,** int ARRAYSIZE**>**  
bool Stack**<**T**,** ARRAYSIZE**>::operator** **!=** **(const** Stack**&** s**)** **const** **{**  
     **return** top **!=** s**.**top**;**  
**}**  
  
  
**template** **<class** T**,** int ARRAYSIZE**>**  
bool Stack**<**T**,** ARRAYSIZE**>::operator** **==** **(const** Stack**&** s**)** **const** **{**  
     **return** top **==** s**.**top**;**  
**}**  
  
  
**template** **<class** T**,** int ARRAYSIZE**>**  
bool Stack**<**T**,** ARRAYSIZE**>::operator** **<** **(const** Stack**&** s**)** **const** **{**  
     **return** top **<** s**.**top**;**  
**}**  
  
  
**template** **<class** T**,** int ARRAYSIZE**>**  
bool Stack**<**T**,** ARRAYSIZE**>::operator** **<=** **(const** Stack**&** s**)** **const** **{**  
     **return** top **<=** s**.**top**;**  
**}**  
  
  
**template** **<class** T**,** int ARRAYSIZE**>**  
bool Stack**<**T**,** ARRAYSIZE**>::operator** **>** **(const** Stack**&** s**)** **const** **{**  
     **return** top **>** s**.**top**;**  
**}**  
  
  
**template** **<class** T**,** int ARRAYSIZE**>**  
bool Stack**<**T**,** ARRAYSIZE**>::operator** **>=** **(const** Stack**&** s**)** **const** **{**  
     **return** top **>=** s**.**top**;**  
**}**  
  
  
#endif

#ifndef QUEUE\_H  
#define QUEUE\_H  
  
#include <string>  
  
**template** **<class** T**,** int ARRAYSIZE **=** 1000**>** ///Definicion  
**class** Queue**{**  
       
**private:**  
       
     T data**[**ARRAYSIZE**];**  
       
     int frontPos**;**  
     int endPos**;**  
       
     void copyAll**(const** Queue**&);**  
       
**public:**  
       
     Queue**();**  
     Queue**(const** Queue**&);**  
       
     bool isEmpty**();**  
     bool isFull**();**  
       
     void enqueue**(const** T**&);**  
       
     T dequeue**();**  
       
     T getFront**()** **const;**  
       
     Queue**&** **operator** **=** **(const** Queue**&);**  
       
**};**  
  
  
  
**template** **<class** T**,** int ARRAYSIZE**>** ///Implementacion  
void Queue**<**T**,** ARRAYSIZE**>::**copyAll **(const** Queue**&** q**)** **{**  
       
     int i**(**0**);**  
       
     **while** **(**i **<=** q**.**endPos**){**  
          **this->**data**[**i**]** **=** q**.**data**[**i**];**  
          i**++;**  
     **}**  
       
     **this** **->** endPos **=** q**.**endPos**;**  
**}**  
  
  
**template** **<class** T**,** int ARRAYSIZE**>**  
Queue**<**T**,** ARRAYSIZE**>::**Queue **(** **)** **:** frontPos**(**0**),** endPos**(**ARRAYSIZE **-** 1**){** **}**  
  
  
**template** **<class** T**,** int ARRAYSIZE**>**  
Queue**<**T**,** ARRAYSIZE**>::**Queue **(const** Queue**&** q**)** **{**  
       
     copyAll**(**q**);**  
**}**  
  
  
**template** **<class** T**,** int ARRAYSIZE**>**  
bool Queue**<**T**,** ARRAYSIZE**>::**isEmpty **(** **)** **{**  
       
     **return** frontPos **==** endPos **+** 1 **or** **(**frontPos **==** 0 **and** endPos **==** ARRAYSIZE **-** 1**);**  
**}**  
  
  
**template** **<class** T**,** int ARRAYSIZE**>**  
bool Queue**<**T**,** ARRAYSIZE**>::**isFull **(** **)** **{**  
       
     **return** frontPos **==** endPos **+** 2  
          **or** **(**frontPos **==** 0 **and** endPos **==** ARRAYSIZE **-** 2**)**  
          **or** **(**frontPos **==** 1 **and** endPos **==** ARRAYSIZE **-**1**);**  
**}**  
  
  
**template** **<class** T**,** int ARRAYSIZE**>**  
void Queue**<**T**,** ARRAYSIZE**>::**enqueue **(const** T**&** e**)** **{**  
       
     data**[**endPos **=** **++**endPos **==** ARRAYSIZE **?** 0 **:** endPos**]** **=** e**;**  
**}**  
  
  
**template** **<class** T**,** int ARRAYSIZE**>**  
T Queue**<**T**,** ARRAYSIZE**>::**dequeue **(** **)** **{**  
       
     T result**(**data**[**frontPos**]);**  
       
     **if** **(++**frontPos **==** ARRAYSIZE**){**  
          frontPos **=** 0**;**  
     **}**  
       
     **return** result**;**  
**}**  
  
  
**template** **<class** T**,** int ARRAYSIZE**>**  
T Queue**<**T**,** ARRAYSIZE**>::**getFront **(** **)** **const** **{**  
       
     **return** data**[**frontPos**];**  
**}**  
  
  
**template** **<class** T**,** int ARRAYSIZE**>**  
Queue**<**T**,** ARRAYSIZE**>&** Queue**<**T**,** ARRAYSIZE**>::operator** **=** **(const** Queue**&** q**)** **{**  
       
     copyAll**(**q**);**  
       
     **return** **\*this;**  
**}**  
  
#endif

ScreenShots









