**Actividad de Aprendizaje 11. La Pila y la Cola, implementación dinámica**

***Problema:***

Implemente la pila y la cola **basadas en listas ligadas**, ~~a través del uso de herencia a partir de la lista simplemente ligada~~. Y  aplíquelo a la actividad 04.

***Requerimientos:***

a)      El estilo de programación debe ser Orientado a Objetos

b)      Debe ser suficiente con el cambio de librería para que el resto del programa de forma idéntica.

Estructuras de Datos I

Christopher Ceballos JImenez

La actividad fue relativamente sencilla, lo único que tuve que realizar es cambiar la cola y lista de arreglos por la dinámica. Además de modificar un poco el main para que trabajara mejor que la actividad 4.

**Main**

#include <iostream>  
#include <string>  
  
#include "stack.h"  
#include "queue.h"  
  
**using** **namespace** std**;**  
  
int prioridadInfija**(**char**);**  
int prioridadPila**(**char**);**  
bool isOperator**(**char**);**  
  
int main **()** **{**  
  
     Queue **<**char**>** myCola**;**  
     Stack **<**char**>** myPila**;**  
  
     string myString**;**  
     char op**;**  
  
  
     **do{**  
          system**(**"cls"**);**  
          cout **<<** "Enter an infix expression without spaces: "**;**  
          fflush**(**stdin**);** getline**(**cin**,** myString**);** ///a/b^(c+d)-e\*f/g^h - ((((((a+b)\*c)-d)^e)/f)+g)\*h  
  
  
          **for(**int i**(**0**);** i **<** myString**.**length**();** i**++){**  
  
               **if** **((**myString**[**i**]** **>=** 'a' **&&** myString**[**i**]** **<=** 'z'**)** **||** **(**myString**[**i**]** **>=** 49 **&&** myString**[**i**]** **<=** 57**)){**  
                    myCola**.**enqueue**(**myString**[**i**]);**  
               **}**  
  
               **if(**isOperator**(**myString**[**i**])){**  
  
                    **if** **(**myPila**.**isEmpty**()){**  
                         myPila**.**push**(**myString**[**i**]);**  
                    **}else{**  
  
                         **if(**prioridadInfija**(**myString**[**i**])** **<** prioridadPila**(**myPila**.**getTop**())){**  
  
                              **while(!**myPila**.**isEmpty**()){**  
                                   myCola**.**enqueue**(**myPila**.**pop**());**  
                              **}**  
  
                              myPila**.**push**(**myString**[**i**]);**  
                         **}**  
                         **if** **(**prioridadInfija**(**myString**[**i**])** **==** prioridadPila**(**myPila**.**getTop**())){**  
  
                              myCola**.**enqueue**(**myPila**.**pop**());**  
                              myPila**.**push**(**myString**[**i**]);**  
                         **}**  
  
                         **if(**prioridadInfija**(**myString**[**i**])** **>** prioridadPila**(**myPila**.**getTop**())){**  
  
                              myPila**.**push**(**myString**[**i**]);**  
                         **}**  
                    **}**  
               **}**  
  
               **if(**myString**[**i**]** **==** '('**){**  
                    myPila**.**push**(**myString**[**i**]);**  
               **}**  
  
               **if(**myString**[**i**]** **==** ')'**){**  
                    **while(**myPila**.**getTop**()** **!=** '('**){**  
  
                         myCola**.**enqueue**(**myPila**.**pop**());**  
                    **}**  
                    myPila**.**pop**();**  
               **}**  
  
          **}**  
  
          **if(!**myPila**.**isEmpty**()){**  
               **while(!**myPila**.**isEmpty**()){**  
                    myCola**.**enqueue**(**myPila**.**pop**());**  
               **}**  
          **}**  
  
          cout **<<** endl **<<** endl**;**  
  
          **try{**  
               **if(!**myCola**.**isEmpty**()){**  
                    **while(!**myCola**.**isEmpty**()){**  
                         cout **<<** myCola**.**dequeue**();**  
                    **}**  
               **}**  
          **}catch(typename** QueueNode**<**char**>::**ExceptionQueueNode ex**){**  
               cout **<<** endl **<<** ex**.**what**()** **<<** endl**;**  
          **}**  
  
          cout **<<** "\n\n\tPress 1 to try again.\n\tPress 0 to exit.\n\t: "**;** cin **>>** op**;**  
     **}while(**op **!=** '0'**);**  
**}**  
  
  
int prioridadInfija**(**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** 4**;**  
     **}**  
  
     **return** 0**;**  
**}**  
  
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**;**  
     **}**  
  
     **return** 0**;**  
**}**  
  
bool isOperator**(**char a**){**  
     **return** a **==** '+' **||** a **==** '-' **||** a **==** '\*' **||** a **==** '/' **||** a **==** '^'**;**  
**}**

**Stack**

#ifndef STACK\_H  
#define STACK\_H  
  
#include "stacknode.h"  
  
#include <iostream>  
#include <exception>  
  
**template** **<class** T**>**  
**class** Stack**{**  
**private:**  
       
     StackNode**<**T**>\*** anchor**;**  
       
     void copyAll**(const** Stack**<**T**>&);**  
       
     void deleteAll**();**  
       
**public:**  
       
     **class** ExceptionStack **:** **public** std**::**exception**{**  
     **private:**  
          std**::**string msg**;**  
            
     **public:**  
          **explicit** ExceptionStack**(const** char**\*** message**)** **:** msg**(**message**){}**  
            
          **explicit** ExceptionStack**(const** std**::**string**&** message**)** **:** msg**(**message**)** **{}**  
            
          **virtual** **~**ExceptionStack**()** **throw()** **{}**  
            
          **virtual** **const** char**\*** what**()** **const** **throw()** **{**  
               **return** msg**.**c\_str**();**  
          **}**  
            
     **};**  
       
     Stack**();**  
     Stack**(const** Stack**<**T**>&);**  
       
     **~**Stack**();**  
       
     bool isEmpty**()** **const;**  
       
     void push**(const** T**&);**  
       
     T pop**();**  
       
     T getTop**()** **const;**  
       
     Stack**<**T**>&** **operator** **=** **(const** Stack**<**T**>&);**       
**};**  
  
  
///Implementacion de la pila  
  
**template** **<class** T**>**  
void Stack**<**T**>::**copyAll**(const** Stack**<**T**>&** s**){**  
       
     StackNode**<**T**>\*** aux**(**s**.**anchor**);**  
     StackNode**<**T**>\*** last**(nullptr);**  
     StackNode**<**T**>\*** newNode**;**  
       
     **while(**aux **!=** **nullptr){**  
            
          **if((**newNode **=** **new** StackNode**<**T**>(**aux **->** getData**()))** **==** **nullptr){**  
               **throw** ExceptionStack**(**"Memoria no disponible, copyAll"**);**  
          **}**  
            
          **if(**last **==** **nullptr){**  
               anchor **=** newNode**;**  
          **}**  
          **else{**  
               last **->** setNext**(**newNode**);**  
          **}**  
            
          last **=** newNode**;**  
            
          aux **=** aux **->** getNext**();**  
     **}**  
**}**  
  
**template** **<class** T**>**  
void Stack**<**T**>::**deleteAll**(){**  
       
     StackNode**<**T**>\*** aux**;**  
       
     **while(**anchor **!=** **nullptr){**  
            
          aux **=** anchor**;**  
          anchor **=** anchor **->** getNext**();**  
            
          **delete** aux**;**  
     **}**  
**}**  
  
**template** **<class** T**>**  
Stack**<**T**>::**Stack**()** **:** anchor**(nullptr)** **{** **}**  
  
**template** **<class** T**>**  
Stack**<**T**>::**Stack**(const** Stack**<**T**>&** s**)** **:** anchor**(nullptr)** **{**  
     copyAll**(**s**);**  
**}**  
  
**template** **<class** T**>**  
Stack**<**T**>::~**Stack**(){**  
     deleteAll**();**  
**}**  
  
**template** **<class** T**>**  
bool Stack**<**T**>::**isEmpty**()** **const** **{**  
     **return** anchor **==** **nullptr;**  
**}**  
  
**template** **<class** T**>**  
void Stack**<**T**>::**push**(const** T**&** e**){**  
       
     StackNode**<**T**>\*** aux**(new** StackNode**<**T**>(**e**));**  
       
     **if(**aux **==** **nullptr){**  
          **throw** ExceptionStack**(**"Memoria no disponible, push"**);**  
     **}**  
       
     aux **->** setNext**(**anchor**);**  
     anchor **=** aux**;**  
**}**  
  
**template** **<class** T**>**  
T Stack**<**T**>::**pop**(){**  
       
     **if(**anchor **==** **nullptr){**  
          **throw** ExceptionStack**(**"insuficiencia de datos, pop"**);**  
     **}**  
       
     T result**(**anchor **->** getData**());**  
       
     StackNode**<**T**>\*** aux**(**anchor**);**  
       
     anchor **=** anchor **->** getNext**();**  
       
     **delete** aux**;**  
       
     **return** result**;**  
**}**  
  
**template** **<class** T**>**  
T Stack**<**T**>::**getTop**()** **const{**  
       
     **if(**anchor **==** **nullptr){**  
          **throw** ExceptionStack**(**"Insuficiencia de datos, getTop"**);**  
     **}**  
       
     **return** anchor **->** getData**();**  
**}**  
  
**template** **<class** T**>**  
Stack**<**T**>&** Stack**<**T**>::operator** **=** **(const** Stack**<**T**>&** s**){**  
       
     deleteAll**();**  
     copyAll**(**s**);**  
       
     **return** **\*this;**  
**}**  
  
#endif

**StackNode**

#ifndef STACKNODE\_H  
#define STACKNODE\_H  
  
**template** **<class** T**>**  
**class** StackNode**{**  
**private:**  
       
     T data**;**  
     StackNode**\*** next**;**  
       
**public:**  
       
     StackNode**();**  
     StackNode**(const** T**&);**  
       
     T getData**()** **const;**  
     StackNode**\*** getNext**()** **const;**  
       
     void setData**(const** T**&);**  
     void setNext**(**StackNode**\*);**  
**};**  
  
///Implementacion del nodo  
  
**template** **<class** T**>**  
StackNode**<**T**>::**StackNode**()** **:** next**(nullptr)** **{** **}**  
  
**template** **<class** T**>**  
StackNode**<**T**>::**StackNode**(const** T**&** e**)** **:** data**(**e**),** next**(nullptr)** **{** **}**  
       
**template** **<class** T**>**  
T StackNode**<**T**>::**getData**()** **const** **{**  
     **return** data**;**  
**}**  
  
**template** **<class** T**>**  
StackNode**<**T**>\*** StackNode**<**T**>::**getNext**()** **const** **{**  
     **return** next**;**  
**}**  
  
**template** **<class** T**>**  
void StackNode**<**T**>::**setData**(const** T**&** e**)** **{**  
     data **=** e**;**  
**}**  
  
**template** **<class** T**>**  
void StackNode**<**T**>::**setNext**(**StackNode**\*** p**)** **{**  
     next **=** p**;**  
**}**  
  
#endif

**Queue**

#ifndef QUEUE\_H  
#define QUEUE\_H  
  
#include "queuenode.h"  
  
#include <iostream>  
#include <exception>  
  
**template** **<class** T**>**  
**class** Queue**{**  
**private:**  
       
     QueueNode**<**T**>\*** header**;**  
       
     void copyAll**(const** Queue**<**T**>&);**  
       
     void deleteAll**();**  
       
**public:**  
       
     **class** ExceptionQueue **:** **public** std**::**exception**{**  
     **private:**  
          std**::**string msg**;**  
            
     **public:**  
          **explicit** ExceptionQueue**(const** char**\*** message**)** **:** msg**(**message**){}**  
            
          **explicit** ExceptionQueue**(const** std**::**string**&** message**)** **:** msg**(**message**)** **{}**  
            
          **virtual** **~**ExceptionQueue**()** **throw()** **{}**  
            
          **virtual** **const** char**\*** what**()** **const** **throw()** **{**  
               **return** msg**.**c\_str**();**  
          **}**  
            
     **};**  
       
     Queue**();**  
     Queue**(const** Queue**<**T**>&);**  
       
     **~**Queue**();**  
       
     bool isEmpty**();**  
       
     void enqueue**(const** T**&);**  
       
     T dequeue**();**  
       
     T getFront**()** **const;**  
       
     Queue**<**T**>&** **operator** **=** **(const** Queue**<**T**>&);**  
       
**};**  
  
**template** **<class** T**>**  
void Queue**<**T**>::**copyAll **(const** Queue**<**T**>&** l**)** **{**  
       
     QueueNode**<**T**>\*** aux**(**l**.**header **->** getNext**());**  
     QueueNode**<**T**>\*** newNode**;**  
       
     **while(**aux **!=** l**.**header**){**  
            
          **try{**  
               **if((**newNode **=** **new** QueueNode**<**T**>(**aux **->** getData**()))** **==** **nullptr){**  
                    **throw** ExceptionQueue**(**"Memoria no disponible, copyAll."**);**  
               **}**  
          **}catch** **(typename** QueueNode**<**T**>::**ExceptionQueueNode ex**){**  
               **throw** ExceptionQueue **(**ex**.**what**());**  
          **}**  
            
          newNode **->** setPrev**(**header **->** getPrev**());**  
          newNode **->** setNext**(**header**);**  
            
          header **->** getPrev**()** **->** setNext**(**newNode**);**  
          header **->** setPrev**(**newNode**);**  
            
          aux **=** aux **->** getNext**();**  
     **}**  
**}**  
  
**template** **<class** T**>**  
void Queue**<**T**>::**deleteAll **(** **)** **{**  
       
     QueueNode**<**T**>\*** aux**;**  
       
     **while(**header **->** getNext**()** **!=** header**){**  
            
          aux **=** header **->** getNext**();**  
          header **->** setNext**(**aux **->** getNext**());**  
            
          **delete** aux**;**  
     **}**  
       
     header **->** setPrev**(**header**);**  
**}**  
  
**template** **<class** T**>**  
Queue**<**T**>::**Queue **(** **)** **:** header**(new** QueueNode**<**T**>){**  
       
     **if(**header **==** **nullptr){**  
          **throw** ExceptionQueue**(**"Memoria no disponible, inicializando queue."**);**  
     **}**  
       
     header **->** setPrev**(**header**);**  
     header **->** setNext**(**header**);**  
**}**  
  
**template** **<class** T**>**  
Queue**<**T**>::**Queue **(const** Queue**<**T**>&** q**)** **:** Queue**()** **{**  
     copyAll**(**q**);**  
**}**  
  
**template** **<class** T**>**  
Queue**<**T**>::~**Queue **(** **)** **{**  
       
     deleteAll**();**  
     **delete** header**;**  
**}**  
  
**template** **<class** T**>**  
bool Queue**<**T**>::**isEmpty **(** **)** **{**  
     **return** header **->** getNext**()** **==** header**;**  
**}**  
  
**template** **<class** T**>**  
void Queue**<**T**>::**enqueue **(const** T**&** e**)** **{**  
       
     QueueNode**<**T**>\*** aux**;**  
       
     **try{**  
          **if((**aux **=** **new** QueueNode**<**T**>(**e**))** **==** **nullptr){**  
               **throw** ExceptionQueue**(**"Memoria no disponible, enqueue."**);**  
          **}**  
     **}catch** **(typename** QueueNode**<**T**>::**ExceptionQueueNode ex**){**  
          **throw** ExceptionQueue**(**ex**.**what**());**  
     **}**  
       
     aux **->** setPrev**(**header **->** getPrev**());**  
     aux **->** setNext**(**header**);**  
       
     header **->** getPrev**()** **->** setNext**(**aux**);**  
     header **->** setPrev**(**aux**);**  
**}**  
  
**template** **<class** T**>**  
T Queue**<**T**>::**dequeue **(** **)** **{**  
       
     **if(**isEmpty**()){**  
          **throw** ExceptionQueue**(**"insuficiencia de datos, dequeue."**);**  
     **}**  
       
     T result**(**header **->** getNext**()** **->** getData**());**  
       
     QueueNode**<**T**>\*** aux**(**header **->** getNext**());**  
       
     aux **->** getPrev**()** **->** setNext**(**aux **->** getNext**());**  
     aux **->** getNext**()** **->** setPrev**(**aux **->** getPrev**());**  
       
     **delete** aux**;**  
       
     **return** result**;**  
**}**  
  
**template** **<class** T**>**  
T Queue**<**T**>::**getFront **(** **)** **const** **{**  
       
     **if(**isEmpty**()){**  
          **throw** ExceptionQueue**(**"insuficiencia de datos, getFront."**);**  
     **}**  
       
     **return** header **->** getNext**()** **->** getData**();**  
**}**  
  
**template** **<class** T**>**  
Queue**<**T**>&** Queue**<**T**>::operator** **=** **(const** Queue**<**T**>&** q**)** **{**  
       
     deleteAll**();**  
     copyAll**(**q**);**  
       
     **return** **\*this;**  
**}**  
  
  
#endif

**QueueNode**

#ifndef QUEUENODE\_H  
#define QUEUENODE\_H  
  
**template** **<class** T**>**  
**class** QueueNode**{**  
**private:**  
       
     T**\*** dataPtr**;**  
     QueueNode**\*** prev**;**  
     QueueNode**\*** next**;**  
       
**public:**  
       
     **class** ExceptionQueueNode **:** **public** std**::**exception**{**  
     **private:**  
          std**::**string msg**;**  
            
     **public:**  
          **explicit** ExceptionQueueNode**(const** char**\*** message**)** **:** msg**(**message**){}**  
            
          **explicit** ExceptionQueueNode**(const** std**::**string**&** message**)** **:** msg**(**message**)** **{}**  
            
          **virtual** **~**ExceptionQueueNode**()** **throw()** **{}**  
            
          **virtual** **const** char**\*** what**()** **const** **throw()** **{**  
               **return** msg**.**c\_str**();**  
          **}**  
            
     **};**  
       
     QueueNode**();**  
     QueueNode**(const** T**&);**  
       
     **~**QueueNode**();**  
       
     T**\*** getDataPtr**()** **const;**  
     T getData**()** **const;**  
     QueueNode**\*** getPrev**()** **const;**  
     QueueNode**\*** getNext**()** **const;**  
       
     void setDataPtr**(**T**\*);**  
     void setData**(const** T**&);**  
     void setPrev**(**QueueNode**\*);**  
     void setNext**(**QueueNode**\*);**  
       
**};**  
  
///Implementacion del nodo  
  
**template** **<class** T**>**  
QueueNode**<**T**>::**QueueNode **(** **)** **:** dataPtr**(nullptr),** prev**(nullptr),** next**(nullptr)** **{** **}**  
  
**template** **<class** T**>**  
QueueNode**<**T**>::**QueueNode **(const** T**&** e**)** **:** dataPtr**(new** T**(**e**)),** prev**(nullptr),** next**(nullptr)** **{**  
       
     **if(**dataPtr **==** **nullptr){**  
          **throw** ExceptionQueueNode**(**"Memoria insuficiente. creando nodo."**);**  
     **}**  
**}**  
  
**template** **<class** T**>**  
QueueNode**<**T**>::~**QueueNode **(){**  
     **delete** dataPtr**;**  
**}**  
       
**template** **<class** T**>**  
T**\*** QueueNode**<**T**>::**getDataPtr **(** **)** **const** **{**  
     **return** dataPtr**;**  
**}**  
       
**template** **<class** T**>**            
T QueueNode**<**T**>::**getData **(** **)** **const** **{**  
       
     **if(**dataPtr **==** **nullptr){**  
          **throw** ExceptionQueueNode**(**"Dato inexistente, getData."**);**  
     **}**  
       
     **return** **\***dataPtr**;**  
**}**  
       
**template** **<class** T**>**  
QueueNode**<**T**>\*** QueueNode**<**T**>::**getPrev **(** **)** **const** **{**  
     **return** prev**;**  
**}**  
       
**template** **<class** T**>**  
QueueNode**<**T**>\*** QueueNode**<**T**>::**getNext **(** **)** **const** **{**  
     **return** next**;**  
**}**  
       
**template** **<class** T**>**  
void QueueNode**<**T**>::**setDataPtr **(**T**\*** e**)** **{**  
     dataPtr **=** e**;**  
**}**  
       
**template** **<class** T**>**  
void QueueNode**<**T**>::**setData **(const** T**&** e**)** **{**  
       
     **if(**dataPtr **==** **nullptr){**  
          **if((**dataPtr **=** **new** T**(**e**))** **==** **nullptr){**  
               **throw** ExceptionQueueNode **(**"Memoria no disponible, setData."**);**  
          **}**  
     **}**  
     **else{**  
          **\***dataPtr **=** e**;**  
     **}**  
**}**  
  
**template** **<class** T**>**  
void QueueNode**<**T**>::**setPrev **(**QueueNode**\*** p**)** **{**  
     prev **=** p**;**  
**}**  
       
**template** **<class** T**>**  
void QueueNode**<**T**>::**setNext **(**QueueNode**\*** p**)** **{**  
     next **=** p**;**  
**}**  
  
#endif

**Capturas de pantalla**





