**Actividad de Aprendizaje 10. La Lista, implementación dinámica doblemente ligada**

***Problema:***

Tome el problema y requerimientos de la actividad 09, y cubra las necesidades utilizando una lista doblemente ligada en lugar de una lista simplemente ligada.

***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.

Christopher Ceballos Jiménez

Estructuras de Datos I

Como lo comentado lo único que se tuvo que realizar fue cambiar la lista simplemente ligada a una doblemente ligada circular con encabezado dummy, además modificar un poco el nodo adaptándolo a plantillas.

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

#ifndef LIST\_H  
#define LIST\_H  
  
#include <exception>  
#include <string>  
#include <iostream>  
#include <exception>  
#include <fstream>  
  
#include "node.h"  
  
**template** **<class** T**>**  
**class** List**{**  
**private:**  
       
     Node**<**T**>\*** header**;**  
       
     void copyAll**(const** List**<**T**>&);**  
       
     bool isValidPos**(**Node**<**T**>\*)** **const;**  
       
     void swapPtr**(**Node**<**T**>\*,** Node**<**T**>\*);**  
       
     void sortByName**(**Node**<**T**>\*,** Node**<**T**>\*);**  
     void sortByAuthor**(**Node**<**T**>\*,** Node**<**T**>\*);**  
       
**public:**  
       
     **typedef** Node**<**T**>\*** Position**;**  
       
     **class** Exception **:** **public** std**::**exception**{**  
     **private:**  
          std**::**string msg**;**  
            
     **public:**  
          **explicit** Exception**(const** char**\*** message**)** **:** msg**(**message**){}**  
            
          **explicit** Exception**(const** std**::**string**&** message**)** **:** msg**(**message**)** **{}**  
            
          **virtual** **~**Exception**()** **throw()** **{}**  
            
          **virtual** **const** char**\*** what**()** **const** **throw()** **{**  
               **return** msg**.**c\_str**();**  
          **}**  
     **};**  
       
     List**();**  
     List**(const** List**<**T**>&);**  
       
     **~**List**();**  
       
     bool isEmpty**()** **const;**  
       
     void insertData**(**Node**<**T**>\*,** **const** T**&);**  
       
     void deleteData**(**Node**<**T**>\*);**  
       
     Node**<**T**>\*** getFirstPos**()** **const;**  
     Node**<**T**>\*** getLastPos**()** **const;**  
     Node**<**T**>\*** getPrevPos**(**Node**<**T**>\*)** **const;**  
     Node**<**T**>\*** getNextPos**(**Node**<**T**>\*)** **const;**  
       
     Node**<**T**>\*** findData**(const** T**&)** **const;**  
       
     T retrieve**(**Node**<**T**>\*)** **const;**  
       
     std**::**string toString**()** **const;**  
       
     void deleteAll**();**  
       
     List**<**T**>&** **operator** **=** **(const** List**<**T**>&);**  
       
     void writeToDisk**(const** std**::**string**&);**  
     void readFromDisk**(const** std**::**string**&);**  
       
     void sortByName**();**  
     void sortByAuthor**();**  
**};**  
  
///Implementacion de la lista  
  
**template** **<class** T**>**  
void List**<**T**>::**copyAll **(const** List**<**T**>&** l**)** **{**  
       
     Node**<**T**>\*** aux**(**l**.**header **->** getNext**());**  
     Node**<**T**>\*** newNode**;**  
       
     **while(**aux **!=** l**.**header**){**  
            
          **try{**  
               **if((**newNode **=** **new** Node**<**T**>(**aux **->** getData**()))** **==** **nullptr){**  
                    **throw** Exception**(**"Memoria no disponible, copyAll."**);**  
               **}**  
          **}catch** **(typename** Node**<**T**>::**Exception ex**){**  
               **throw** Exception **(**ex**.**what**());**  
          **}**  
            
          newNode **->** setPrev**(**header **->** getPrev**());**  
          newNode **->** setNext**(**header**);**  
            
          header **->** getPrev**()** **->** setNext**(**newNode**);**  
          header **->** setPrev**(**newNode**);**  
            
          aux **=** aux **->** getNext**();**  
     **}**  
**}**  
  
**template** **<class** T**>**  
bool List**<**T**>::**isValidPos **(**Node**<**T**>\*** p**)** **const** **{**  
       
     Node**<**T**>\*** aux**(**header **->** getNext**());**  
            
     **while(**aux **!=** header**){**  
            
          **if(**aux **==** p**){**  
               **return** **true;**  
          **}**  
            
          aux **=** aux **->** getNext**();**  
     **}**  
       
     **return** **false;**  
**}**  
  
**template** **<class** T**>**  
List**<**T**>::**List **(** **)** **:** header**(new** Node**<**T**>)** **{**  
       
     **if(**header **==** **nullptr){**  
          **throw** Exception**(**"Memoria no disponible, inicializando lista."**);**  
     **}**  
       
     header **->** setPrev**(**header**);**  
     header **->** setNext**(**header**);**  
**}**  
  
**template** **<class** T**>**  
List**<**T**>::**List **(const** List**<**T**>&** l**)** **:** List**()** **{**  
     copyAll**(**l**);**  
**}**  
  
**template** **<class** T**>**  
List**<**T**>::~**List **(** **)** **{**  
       
     deleteAll**();**  
       
     **delete** header**;**  
**}**  
  
**template** **<class** T**>**  
bool List**<**T**>::**isEmpty **(** **)** **const** **{**  
     **return** header **->** getNext**()** **==** header**;**  
**}**  
  
**template** **<class** T**>**  
void List**<**T**>::**insertData **(**Node**<**T**>\*** p**,** **const** T**&** e**)** **{**  
       
     **if(**p **!=** **nullptr** **and** **!**isValidPos**(**p**)){**  
          **throw** Exception**(**"Posicion invalida, insertData."**);**  
     **}**  
       
     Node**<**T**>\*** aux**;**  
       
     **try{**  
          aux **=** **new** Node**<**T**>(**e**);**  
     **}catch** **(typename** Node**<**T**>::**Exception ex**){**  
          **throw** Exception**(**ex**.**what**());**  
     **}**  
       
     **if(**aux **==** **nullptr){**  
          **throw** Exception**(**"Memoria no disponible, insertData."**);**  
     **}**  
       
     **if(**p **==** **nullptr){** *//Insertar al principio.*          p **=** header**;**  
     **}**  
       
     aux **->** setPrev**(**p**);**  
     aux **->** setNext**(**p **->** getNext**());**  
       
     p **->** getNext**()** **->** setPrev**(**aux**);**  
     p **->** setNext**(**aux**);**  
**}**  
  
**template** **<class** T**>**  
void List**<**T**>::**deleteData **(**Node**<**T**>\*** p**)** **{**  
       
     **if(!**isValidPos**(**p**)){**  
          **throw** Exception**(**"Posicion invalida, deleteData."**);**  
     **}**  
       
     p **->** getPrev**()** **->** setNext**(**p **->** getNext**());**  
     p **->** getNext**()** **->** setPrev**(**p **->** getPrev**());**  
       
     **delete** p**;**  
**}**  
  
**template** **<class** T**>**  
Node**<**T**>\*** List**<**T**>::**getFirstPos **(** **)** **const** **{**  
       
     **if(**isEmpty**()){**  
          **return** **nullptr;**  
     **}**  
       
     **return** header **->** getNext**();**  
**}**  
  
**template** **<class** T**>**  
Node**<**T**>\*** List**<**T**>::**getLastPos **(** **)** **const** **{**  
       
     **if(**isEmpty**()){**  
          **return** **nullptr;**  
     **}**  
       
     **return** header **->** getPrev**();**  
**}**  
  
**template** **<class** T**>**  
Node**<**T**>\*** List**<**T**>::**getPrevPos **(**Node**<**T**>\*** p**)** **const** **{**  
       
     **if(**p **==** header **->** getNext**()** **or** **!**isValidPos**(**p**)){**  
          **return** **nullptr;**  
     **}**  
       
     **return** p **->** getPrev**();**  
**}**  
  
**template** **<class** T**>**  
Node**<**T**>\*** List**<**T**>::**getNextPos **(**Node**<**T**>\*** p**)** **const** **{**  
       
     **if(**p **==** header **->** getPrev**()** **or** **!**isValidPos**(**p**)){**  
          **return** **nullptr;**  
     **}**  
       
     **return** p **->** getNext**();**  
**}**  
  
**template** **<class** T**>**  
Node**<**T**>\*** List**<**T**>::**findData **(const** T**&** e**)** **const** **{**  
       
     Node**<**T**>\*** aux**(**header **->** getNext**());**  
       
     **while(**aux **!=** header**){**  
            
          **if(**aux **->** getData**()** **==** e**){**  
               **return** aux**;**  
          **}**  
            
          aux **=** aux **->** getNext**();**  
     **}**  
       
     **return** **nullptr;**  
**}**  
  
**template** **<class** T**>**  
T List**<**T**>::**retrieve **(**Node**<**T**>\*** p**)** **const** **{**  
       
     **if(!**isValidPos**(**p**)){**  
          **throw** Exception**(**"Posicion invalida, retrieve."**);**  
     **}**  
       
     **return** p **->** getData**();**  
**}**  
  
**template** **<class** T**>**  
std**::**string List**<**T**>::**toString **(** **)** **const** **{**  
       
     std**::**string result**;**  
     Node**<**T**>\*** aux**(**header **->** getNext**());**  
       
     **while(**aux **!=** header**){**  
            
          result **+=** aux **->** getData**().**toString**();**  
          aux **=** aux **->** getNext**();**  
            
     **}**  
       
     **return** result**;**  
**}**  
  
**template** **<class** T**>**  
void List**<**T**>::**deleteAll **(** **)** **{**  
       
     Node**<**T**>\*** aux**;**  
       
     **while(**header **->** getNext**()** **!=** header**){**  
            
          aux **=** header **->** getNext**();**  
          header **->** setNext**(**aux **->** getNext**());**  
            
          **delete** aux**;**  
            
     **}**  
       
     header **->** setPrev**(**header**);**  
**}**  
  
**template** **<class** T**>**  
List**<**T**>&** List**<**T**>::operator** **=** **(const** List**<**T**>&** l**)** **{**  
     deleteAll**();**  
       
     copyAll**(**l**);**  
       
     **return** **\*this;**  
**}**  
  
**template** **<class** T**>**  
void List**<**T**>::**writeToDisk**(const** std**::**string**&** fileName**){**  
       
     std**::**ofstream myFile**;**  
       
     myFile**.**open**(**fileName**,** myFile**.**trunc*/\*ios\_base::trunc\*/***);**  
       
     **if(!**myFile**.**is\_open**()){**  
            
          std**::**string message**;**  
          message **=** "No se pudo abrir el archivo "**;**  
          message **+=** fileName**;**  
          message **+=** " para escritura, writeToDisk."**;**  
            
          **throw** Exception**(**message**);**  
     **}**  
       
     Node**<**T**>\*** aux**(**header **->** getNext**());**  
       
     **while(**aux **!=** header**){**  
            
          myFile **<<** aux **->** getData**();**  
          aux **=** aux **->** getNext**();**  
     **}**  
       
     myFile**.**close**();**  
**}**  
  
**template** **<class** T**>**  
void List**<**T**>::**readFromDisk**(const** std**::**string**&** fileName**){**  
       
     std**::**ifstream myFile**;**  
       
     myFile**.**open**(**fileName**);**  
       
     **try{**  
          **if(!**myFile**.**is\_open**()){**  
               std**::**string message**;**  
               message **=** "No se pudo abrir el archivo "**;**  
               message **+=** fileName**;**  
               message **+=** " para lectura, readFromDisk."**;**  
          **}**  
            
     **}** **catch(**Exception ex**)** **{**  
          myFile**.**close**();**  
            
          std**::**string message**;**  
          message **+=**fileName**;**  
          message **+=** " : "**;**  
          message **+=** ex**.**what**();**  
          message **+=** ", readFromDisk"**;**  
            
          **throw** Exception **(**message**);**  
     **}**  
       
     deleteAll**();**  
       
     T myData**;**  
     Node**<**T**>\*** aux**;**  
       
     **try{**  
          **while(**myFile **>>** myData**){** *// myFile >> myData returna un falso o verdadero*                 
               **if((**aux **=** **new** Node**<**T**>(**myData**))** **==** **nullptr){**  
                    myFile**.**close**();**  
                    **throw** Exception**(**"Memoria no disponible. readFromDisk."**);**  
               **}**  
                 
               aux **->** setPrev**(**header **->** getPrev**());**  
               aux **->** setNext**(**header**);**  
                 
               header **->** getPrev**()** **->** setNext**(**aux**);**  
               header **->** setPrev**(**aux**);**  
          **}**  
     **}** **catch(**Exception ex**)** **{**  
          myFile**.**close**();**  
            
          std**::**string message**(**"Error durante la lectura del archivo."**);**  
            
          message **+=**fileName**;**  
          message **+=** " : "**;**  
          message **+=** ex**.**what**();**  
          message **+=** ", readFromDisk"**;**  
            
          **throw** Exception **(**message**);**  
     **}**  
       
     myFile**.**close**();**  
**}**  
  
**template** **<class** T**>**  
void List**<**T**>::**sortByName**(){**  
     sortByName**(**getFirstPos**(),** getLastPos**());**  
**}**  
  
**template** **<class** T**>**  
void List**<**T**>::**sortByName**(**Node**<**T**>\*** leftEdge**,** Node**<**T**>\*** rightEdge**){**  
       
     **if(**leftEdge **==** rightEdge**){**  
          **return;**  
     **}**  
       
     **if(**leftEdge **->** getNext**()** **==** rightEdge**){**  
          **if(**leftEdge **->** getData**().**getSongName**()** **>** rightEdge **->** getData**().**getSongName**()){**  
               swapPtr**(**leftEdge**,** rightEdge**);**  
          **}**  
            
          **return;**  
     **}**  
       
     *//separacion*     Node**<**T**>\*** i**(**leftEdge**);**  
     Node**<**T**>\*** j**(**rightEdge**);**  
       
     **while(**i **!=** j**){**  
          **while(**i **!=** j **and** i **->** getData**().**getSongName**()** **<=** rightEdge **->** getData**().**getSongName**()){**  
               i **=** i **->** getNext**();**  
          **}**  
          **while(**i **!=** j **and** j **->** getData**().**getSongName**()** **>=** rightEdge **->** getData**().**getSongName**()){**  
               j **=** j **->** getPrev**();**  
          **}**  
            
          swapPtr**(**i**,** j**);**  
     **}**  
       
     swapPtr**(**i**,** rightEdge**);**  
       
     *//divide y venceras*     **if(**i **!=** leftEdge**){**  
          sortByName**(**leftEdge**,** i **->** getPrev**());**  
     **}**  
       
     **if(**i **!=** rightEdge**){**  
          sortByName**(**i **->** getNext**(),** rightEdge**);**  
     **}**  
**}**  
  
**template** **<class** T**>**  
void List**<**T**>::**sortByAuthor**(){**  
     sortByAuthor**(**getFirstPos**(),** getLastPos**());**  
**}**  
  
**template** **<class** T**>**  
void List**<**T**>::**sortByAuthor**(**Node**<**T**>\*** leftEdge**,** Node**<**T**>\*** rightEdge**){**  
       
     **if(**leftEdge **==** rightEdge**){**  
          **return;**  
     **}**  
       
     **if(**leftEdge **->** getNext**()** **==** rightEdge**){**  
          **if(**leftEdge **->** getData**().**getaAuthor**()** **>** rightEdge **->** getData**().**getaAuthor**()){**  
               swapPtr**(**leftEdge**,** rightEdge**);**  
          **}**  
            
          **return;**  
     **}**  
       
     *//separacion*     Node**<**T**>\*** i**(**leftEdge**);**  
     Node**<**T**>\*** j**(**rightEdge**);**  
       
     **while(**i **!=** j**){**  
          **while(**i **!=** j **and** i **->** getData**().**getaAuthor**()** **<=** rightEdge **->** getData**().** getaAuthor**()){**  
               i **=** i **->** getNext**();**  
          **}**  
          **while(**i **!=** j **and** j **->** getData**().**getaAuthor**()** **>=** rightEdge **->** getData**().** getaAuthor**()){**  
               j **=** j **->** getPrev**();**  
          **}**  
            
          swapPtr**(**i**,** j**);**  
     **}**  
       
     swapPtr**(**i**,** rightEdge**);**  
       
     *//divide y venceras*     **if(**i **!=** leftEdge**){**  
          sortByAuthor**(**leftEdge**,** i **->** getPrev**());**  
     **}**  
       
     **if(**i **!=** rightEdge**){**  
          sortByAuthor**(**i **->** getNext**(),** rightEdge**);**  
     **}**  
**}**  
  
**template** **<class** T**>**  
void List**<**T**>::**swapPtr**(**Node**<**T**>\*** a**,** Node**<**T**>\*** b**){**  
       
     **if(**a **!=** b**){**  
          Song**\*** aux**(**a **->** getDataPtr**());**  
          a **->** setDataPtr**(**b **->** getDataPtr**());**  
          b **->** setDataPtr**(**aux**);**  
     **}**  
**}**  
       
#endif

Capturas de pantalla











