**Actividad de Aprendizaje 06. Métodos de ordenamiento iterativos**

***Problema:***

Reutilice el resultado de la actividad 05. Se ha encontrado un problema en el programa: la búsqueda binaria no funciona muy bien, el jefe de programadores se ha dado cuenta que la lista casi nunca está ordenada y por eso falla, por lo que es necesario añadirle una opción para ordenarla.

Haga un programa que realice el ordenamiento de la lista, opcionalmente por nombre de la canción como por nombre del cantante. Para evaluar su funcionamiento la radiodifusora quiere probar con distintos tipos de ordenamiento.

***Requerimientos:***

a)      El estilo de programación debe ser Orientado a Objetos

b)      Debe ofrecer las opciones de ordenamiento: burbuja (mejorada), shell, inserción, y selección

c)       Los métodos de ordenamiento se implementarán como métodos de la clase Lista

Christopher Ceballos Jimenez

219750442

Estructuras de Datos I

Métodos de ordenamiento fue algo sencillo hasta que llegue al punto de querer ordenar por nombre de la canción o por autor, ya que me causo bastantes problemas. Mi meta era hacer lo menos repetitivo el programa con la menor cantidad de líneas posibles y utilizar los operadores de comparación para realizar lo ordenamientos. En base a eso elabore el siguiente código.

Lista de Canciones en desorden

Firebird

Milky Chance

15

Aunque No Sea Conmigo

Enrique Bunbury

1267

Believer

Imagine Dragons

2413

Donna

The Lumineers

1299

Gimme The Power

Molotov

669

Holiday

Green Day

1313

Koala

Oliver Heldens

1538

La Pelotona

Cartel De Santa

2786

Last Resort

Papa Roach

503

Levels

Avicii

2421

Mockingbird

Eminem

880

Nothing Else Matters

Metallica

889

Out Of Time Man

Mick Harvey

2325

Pequeño Tratado De Un Adios

Panteon Rococo

2804

Ride

Twenty One Pilots

2225

Riptide

Vance Joy

1691

Rucon

Aleman

2650

Sensualidad

Bad Bunny

793

Shepherd Of Fire

Avenged Sevenfold

2822

Toxicity

System Of A Down

812

Ilussia

Mago De Oz

706

Universe

Don Diablo

218

Amnesia

Inspector

414

Mmm Good

12th Planet

653

Electric Feel

MGMT

535

Tiburon

Proyecto Uno

1357

People Say

Don Diablo

1921

#include <iostream>  
#include <windows.h>  
#include <string>  
#include <random>  
#include <chrono>  
#include <functional>  
  
#include "list.h"  
#include "song.h"  
  
**using** **namespace** std**;**  
  
void addSong**();**  
void deleteSong**();**  
void findSong**();**  
void sortSong**();**  
  
int randNum**;**  
string myStr**;**  
List **<**Song**>** myLista**;**  
Song mySong**;**  
  
default\_random\_engine generator**(**chrono**::**system\_clock**::**now**().**time\_since\_epoch**().**count**());**  
uniform\_int\_distribution**<**int**>** distribution**(**1**,**3000**);**  
**auto** dice **=** bind**(**distribution**,** generator**);**  
  
int main **()** **{**  
       
     char op**;**  
       
     **do{**  
            
          **if** **(**myLista**.**isEmpty**()){**  
                 
               cout **<<** "\nNo hay canciones en el sistema." **<<** endl**;**  
          **}**  
            
          **if** **(!**myLista**.**isEmpty**()){**  
                 
               cout**<<** "\n\tTitulo - Artista - Ranking"**<<** endl **<<** endl**;**  
               cout **<<** myLista**.**toString**();**  
          **}**  
            
          cout **<<** "\n\t1- Agregar una lista de canciones." **<<** endl**;**  
          cout **<<** "\t2- Guardar lista actual." **<<** endl**;**  
          cout **<<** "\t3- Agregar cancion." **<<** endl**;**  
          cout **<<** "\t4- Eliminar cancion." **<<** endl**;**  
          cout **<<** "\t5- Buscar cancion." **<<** endl**;**  
          cout **<<** "\t6- Ordenar canciones." **<<** endl**;**  
          cout **<<** "\t0- Salir" **<<** endl**;**  
          cout **<<** "\nElige una opcion: "**;** cin **>>** op**;**  
            
          **switch(**op**){**  
          **case** '1'**:**  
               **{**       
               system**(**"cls"**);**  
                    string savedSongs**;**  
                      
                    cout **<<** "\nIngresa el nombre del archivo donde se encuentran las canciones: "**;**  
                    fflush**(**stdin**);** getline**(**cin**,** savedSongs**);**  
                      
                    myLista**.**readFromDisk**(**savedSongs**);**  
               **}**  
               **break;**  
          **case** '2'**:**  
               **{**       
               system**(**"cls"**);**  
               string savedSongs**;**  
               cout **<<** "\nIngresa el nombre que va a tener el archivo (extension .txt): "**;** fflush**(**stdin**);** getline**(**cin**,** savedSongs**);**  
                 
               myLista**.**writeToDisk**(**savedSongs**);**  
               **}**  
               **break;**  
          **case** '3'**:**  
               system**(**"cls"**);**  
               addSong**();**  
                 
               **break;**  
                 
          **case** '4'**:**  
               system**(**"cls"**);**  
               deleteSong**();**  
                 
               **break;**  
                 
          **case** '5'**:**  
               system**(**"cls"**);**  
               findSong**();**  
                 
               **break;**  
                 
          **case** '6'**:**  
               system**(**"cls"**);**  
               sortSong**();**  
               **break;**  
                 
          **default:**  
               cout **<<** "\nOpcion invalida, intentelo de nuevo." **<<** endl**;**  
               **break;**  
          **}**  
            
     **}while(**op **!=** '0'**);**  
       
**}**  
  
  
void addSong**()** **{**  
  
     randNum **=** dice**();**  
     char op**;**  
       
     **do{**  
          std**::**string s **=** ""**;**  
          s **=** std**::**to\_string**(**dice**());**  
            
          cout**<<** "\n\nNombre de la cancion: "**;**  
          fflush**(**stdin**);**getline**(**cin**,** myStr**);**  
          myStr**[**0**]** **=** toupper**(**myStr**[**0**]);**  
          mySong**.**setSongName**(**myStr**);**  
            
          cout**<<** "Artista: "**;**  
          fflush**(**stdin**);**getline**(**cin**,** myStr**);**  
          myStr**[**0**]** **=** toupper**(**myStr**[**0**]);**  
          mySong**.**setAuthor**(**myStr**);**  
            
            
          mySong**.**setRanking**(**s**);**  
            
          myLista**.**insertData**(**myLista**.**getLastPos**(),** mySong**);**  
            
          cout **<<** "\nDeseas agregar otra cancion? (1=Si, 0= No)"**;** cin **>>** op**;**  
            
     **}while(**op **!=** '0'**);**  
       
**}**  
  
void deleteSong**(){**  
       
     int pos**;**  
     string myStr**;**  
       
     cout **<<** "\nIngresa nombre de la cancion a eliminar: "**;**  
     fflush**(**stdin**);** getline**(**cin **,** myStr**);**  
     mySong**.**setSongName**(**myStr**);**  
       
     pos **=** myLista**.**findDataLineal**(**mySong**);**  
       
     myLista**.**deleteData**(**pos**);**  
       
     cout **<<** "\nSe ha eliminado correctamente." **<<** endl**;**  
       
**}**  
  
void findSong**(){**  
       
     char op**;**  
       
     cout **<<** "\n\n\tMetodo de busqueda:\n\n\t1- Lineal.\n\t2- Binaria.\n\n\t >>: "**;** cin **>>** op**;**  
       
     **switch(**op**){**  
     **case** '1'**:**  
          **{**  
               string myStr**;**  
               int pos**;**  
                 
               cout **<<** "\nNombre de la cancion a buscar: "**;**  
               fflush**(**stdin**);** getline**(**cin **,** myStr**);**  
               myStr**[**0**]** **=** toupper**(**myStr**[**0**]);**  
                 
               mySong**.**setSongName**(**myStr**);**  
               mySong**.**setAuthor**(**myStr**);**  
                 
                 
               pos **=** myLista**.**findDataLineal**(**mySong**);**  
                 
               **if(**pos **==** **-**1**){**  
                    cout **<<** "\nLa cancion " **<<** myStr **<<** " no se encuentra en la lista." **<<** endl**;**  
               **}else{**  
                    cout **<<** "\n\n\tCancion encontrada: " **<<** myLista**.**retrieve**(**pos**).**toString**()** **<<** endl**;**  
               **}**  
                 
          **}**  
          **break;**  
            
     **case** '2'**:**  
          **{**  
                 
               string myStr**;**  
               int pos**;**  
               char op2**;**  
                 
               cout **<<** "\n\t1- Buscar por Nombre \n\t2- Buscar por Autor\n\t>>: "**;** cin **>>** op2**;**  
                 
               **switch(**op2**){**  
               **case** '1'**:**  
                      
                    myLista**.**sortDataShellByName**();**  
                      
                    cout **<<** "\nNombre de la cancion: "**;**  
                    fflush**(**stdin**);** getline**(**cin **,** myStr**);**  
                    myStr**[**0**]** **=** toupper**(**myStr**[**0**]);**  
                      
                    mySong**.**setSongName**(**myStr**);**  
                      
                    pos **=** myLista**.**findDataBinaria**(**mySong**);**  
                      
                    **if(**pos **==** **-**1**){**  
                         cout **<<** "\nLa cancion " **<<** myStr **<<** " no se encuentra en la lista." **<<** endl**;**  
                    **}else{**  
                         cout **<<** "\n\n\tCancion encontrada: " **<<** myLista**.**retrieve**(**pos**).**toString**()** **<<** endl**;**  
                    **}**  
                      
                    **break;**  
                      
               **case** '2'**:**  
                      
                    myLista**.**sortDataShellByAuthor**();**  
                      
                    cout **<<** "\nNombre de la cancion: "**;**  
                    fflush**(**stdin**);** getline**(**cin **,** myStr**);**  
                    myStr**[**0**]** **=** toupper**(**myStr**[**0**]);**  
                      
                    mySong**.**setAuthor**(**myStr**);**  
                      
                    pos **=** myLista**.**findDataBinaria**(**mySong**);**  
                      
                    **if(**pos **==** **-**1**){**  
                         cout **<<** "\nLa cancion " **<<** myStr **<<** " no se encuentra en la lista." **<<** endl**;**  
                    **}else{**  
                         cout **<<** "\n\n\tCancion encontrada: " **<<** myLista**.**retrieve**(**pos**).**toString**()** **<<** endl**;**  
                    **}**  
                      
                    **break;**  
               **}**  
                 
                 
          **}**  
          **break;**  
            
     **default:**  
          cout **<<** "\nOpcion invalida, intentelo de nuevo." **<<** endl**;**  
          **break;**  
     **}**  
  
**}**  
       
void sortSong**(){**  
       
     char op**(**'\0'**),** op2**(**'\0'**);**  
       
     cout **<<** "\nSeleciona el tipo de ordenamiento a utilizar:" **<<** endl **<<** endl**;**  
     cout **<<** "\t1- Ordenamiento tipo Bubble.\n\t2- Ordenmiento tipo Insercion.\n\t3- Ordemiento tipo Seleccion.\n\t4- Ordenamiento tipo Shell." **<<** endl**;**  
     cout **<<** "\n\t>>: "**;** cin **>>** op**;**  
       
     **switch(**op**){**  
     **case** '1'**:**  
          cout **<<** "\n\t1- Ordenar por Nombre \n\t2- Ordenar por Autor\n\t>>: "**;** cin **>>** op2**;**  
            
          **if(**op2 **==** '1'**){**  
               myLista**.**sortDataBubbleByName**();**  
               cout **<<** "\nOrdenando"**;**  
               Sleep**(**200**);** cout **<<** " . "**;** Sleep**(**200**);** cout **<<** " . "**;** Sleep**(**200**);** cout **<<** " .\n"**;**  
          **}**  
          **if(**op2 **==** '2'**){**  
               myLista**.**sortDataBubbleByAuthor**();**  
               cout **<<** "\nOrdenando"**;**  
               Sleep**(**200**);** cout **<<** " . "**;** Sleep**(**200**);** cout **<<** " . "**;** Sleep**(**200**);** cout **<<** " .\n"**;**  
          **}**  
            
            
          **break;**  
            
     **case** '2'**:**  
            
          cout **<<** "\n\t1- Ordenar por Nombre \n\t2- Ordenar por Autor\n\t>>: "**;** cin **>>** op2**;**  
            
          **if(**op2 **==** '1'**){**  
               myLista**.**sortDataInsertByName**();**  
               cout **<<** "\nOrdenando"**;**  
               Sleep**(**200**);** cout **<<** " . "**;** Sleep**(**200**);** cout **<<** " . "**;** Sleep**(**200**);** cout **<<** " .\n"**;**  
          **}**  
          **if(**op2 **==** '2'**){**  
               myLista**.**sortDataInsertByAuthor**();**  
               cout **<<** "\nOrdenando"**;**  
               Sleep**(**200**);** cout **<<** " . "**;** Sleep**(**200**);** cout **<<** " . "**;** Sleep**(**200**);** cout **<<** " .\n"**;**  
          **}**  
            
          **break;**  
            
     **case** '3'**:**  
            
          cout **<<** "\n\t1- Ordenar por Nombre \n\t2- Ordenar por Autor\n\t>>: "**;** cin **>>** op2**;**  
            
          **if(**op2 **==** '1'**){**  
               myLista**.**sortDataSelectByName**();**  
               cout **<<** "\nOrdenando"**;**  
               Sleep**(**200**);** cout **<<** " . "**;** Sleep**(**200**);** cout **<<** " . "**;** Sleep**(**200**);** cout **<<** " .\n"**;**  
          **}**  
          **if(**op2 **==** '2'**){**  
               myLista**.**sortDataSelectByAuthor**();**  
               cout **<<** "\nOrdenando"**;**  
               Sleep**(**200**);** cout **<<** " . "**;** Sleep**(**200**);** cout **<<** " . "**;** Sleep**(**200**);** cout **<<** " .\n"**;**  
          **}**  
            
          **break;**  
            
     **case** '4'**:**  
            
          cout **<<** "\n\t1- Ordenar por Nombre \n\t2- Ordenar por Autor\n\t>>: "**;** cin **>>** op2**;**  
            
          **if(**op2 **==** '1'**){**  
               myLista**.**sortDataShellByName**();**  
               cout **<<** "\nOrdenando"**;**  
               Sleep**(**200**);** cout **<<** " . "**;** Sleep**(**200**);** cout **<<** " . "**;** Sleep**(**200**);** cout **<<** " .\n"**;**  
          **}**  
          **if(**op2 **==** '2'**){**  
               myLista**.**sortDataShellByAuthor**();**  
               cout **<<** "\nOrdenando"**;**  
               Sleep**(**200**);** cout **<<** " . "**;** Sleep**(**200**);** cout **<<** " . "**;** Sleep**(**200**);** cout **<<** " .\n"**;**  
          **}**  
            
          **break;**  
            
     **default:**  
          cout **<<** "\nOpcion invalida, intentelo de nuevo." **<<** endl**;**  
          **break;**  
     **}**  
**}**

#ifndef LIST\_H\_INCLUDED  
#define LIST\_H\_INCLUDED  
  
#include <iostream>  
#include <string>  
#include <exception>  
#include <fstream>  
  
#include "song.h"  
  
**template** **<class** T**,** int ARRAYSIZE **=** 5000**>**  
**class** List **{**  
**private:**  
     T data**[**ARRAYSIZE**];**  
     int last**;**  
  
     bool isValidPos**(const** int**&);**  
  
     void copyAll**(const** List**&);**  
       
     void swapData**(**T**&,** T**&);**  
**public:**  
  
     **class** ListException **:** **public** std**::**exception**{**  
     **private:**  
          std**::**string msg**;**  
     **public:**  
          **explicit** ListException**(const** char**\*** message**)** **:** msg**(**message**){}**  
  
          **explicit** ListException**(const** std**::**string**&** message**)** **:** msg**(**message**)** **{}**  
  
          **virtual** **~**ListException**()** **throw()** **{}**  
  
          **virtual** **const** char**\*** what**()** **const** **throw()** **{**  
               **return** msg**.**c\_str**();**  
          **}**  
     **};**  
  
     List**();**  
     List**(const** List**&);**  
       
     bool isEmpty**();**  
     bool isFull**();**  
  
     void insertData**(const** int**&,const** T**&);***//primera la pos y luego el elemento*  
     void deleteData**(const** int**&);**  
  
     int getFirstPos**()** **const;**  
     int getLastPos**()** **const;**  
     int getPrevPos**(const** int**&)** **const;**  
     int getNextPos**(const** int**&)** **const;**  
  
     int findDataLineal**(const** T**&);**  
     int findDataBinaria**(const** T**&);**  
       
     void sortDataBubbleByName**();**  
     void sortDataBubbleByAuthor**();**  
     void sortDataInsertByName**();**  
     void sortDataInsertByAuthor**();**  
     void sortDataSelectByName**();**  
     void sortDataSelectByAuthor**();**  
     void sortDataShellByName**();**  
     void sortDataShellByAuthor**();**  
  
     T retrieve**(const** int**&);**  
  
     std**::**string toString**()** **const;**  
  
     void deleteAll**();**  
       
     void writeToDisk**(const** std**::**string**&);**  
     void readFromDisk**(const** std**::**string**&);**  
       
     List**&** **operator** **=** **(const** List**&);**  
       
**};**  
  
*//implementacion*  
  
**template** **<class** T**,** int ARRAYSIZE**>**  
List**<**T**,**ARRAYSIZE**>::**List**()** **:** last**(-**1**)** **{}**  
  
**template** **<class** T**,** int ARRAYSIZE**>**  
List**<**T**,** ARRAYSIZE**>::**List**(const** List**&** s**)** **:** last**(**s**.**last**)** **{** **}**  
  
  
**template** **<class** T**,** int ARRAYSIZE**>**  
void List **<**T**,**ARRAYSIZE**>** **::** copyAll**(const** List**&** s**)** **{**  
       
     int i**(**0**);**  
       
     **while** **(**i **<=** s**.**last**)** **{**  
          **this** **->** data**[**i**]** **=** s**.**data**[**i**];**  
          i**++;**  
     **}**  
       
     **this->** last **=** s**.**last**;**  
**}**  
  
  
**template** **<class** T**,**int ARRAYSIZE**>**  
void List**<**T**,** ARRAYSIZE**>::**swapData**(**T**&** a**,** T**&** b**){** *// swapData*       
     T aux**(**a**);**  
     a **=** b**;**  
     b **=** aux**;**  
       
**}**  
  
**template** **<class** T**,**int ARRAYSIZE**>**  
bool List**<**T**,** ARRAYSIZE**>::**isValidPos**(const** int**&** p**){**  
     **return** p **>=** 0 **and** p**<=** last**;**  
**}**  
  
  
**template** **<class** T**,**int ARRAYSIZE**>**  
bool List**<**T**,** ARRAYSIZE**>::**isEmpty **(** **)** **{**  
     **return** last **==** **-**1**;**  
  
**}**  
  
**template** **<class** T**,**int ARRAYSIZE**>**  
bool List**<**T**,** ARRAYSIZE**>::**isFull **(** **)** **{**  
     **return** last **==** ARRAYSIZE **-** 1**;**  
  
**}**  
  
**template** **<class** T**,**int ARRAYSIZE**>**  
void List**<**T**,** ARRAYSIZE**>::**insertData **(const** int**&** p**,** **const** T**&** e**)** **{**  
  
     **if(**isFull**()){**  
          **throw** ListException**(**"Desbordamiento de datos, insertData"**);**  
     **}**  
  
     **if(**p **!=** **-**1 **and** **!**isValidPos**(**p**)){**  
          **throw** ListException**(**"Posicion invalida, insertData"**);**  
     **}**  
       
     int i**(**last**);**  
  
     **while(**i **>** p**){**  
          data**[**i**+**1**]** **=** data**[**i**];**  
            
          i**--;**  
     **}**  
  
     data**[**p**+**1**]** **=** e**;**  
  
     last**++;**  
**}**  
  
**template** **<class** T**,**int ARRAYSIZE**>**  
void List**<**T**,** ARRAYSIZE**>::**deleteData**(const** int**&** p**){**  
       
     **if(!**isValidPos**(**p**)){**  
          **throw** ListException**(**"Posicion invalida, deleteData"**);**  
     **}**  
  
     int i**(**p**);**  
  
     **while(**i **<** last**){**  
            
          data**[**i**]=** data**[**i**+**1**];**  
          i**++;**  
     **}**  
  
     last**--;**  
**}**  
  
**template** **<class** T**,**int ARRAYSIZE**>**  
int List**<**T**,** ARRAYSIZE**>::**getFirstPos**()** **const{**  
     **if** **(**isEmpty**()){**  
          **return** **-**1**;**  
     **}**  
  
     **return** 0**;**  
**}**  
  
**template** **<class** T**,**int ARRAYSIZE**>**  
int List**<**T**,** ARRAYSIZE**>::**getLastPos **(** **)** **const{**  
     **return** last**;**  
  
**}**  
  
**template** **<class** T**,**int ARRAYSIZE**>**  
int List**<**T**,** ARRAYSIZE**>::**getPrevPos **(const** int**&** p**)** **const{**  
     **if(**p**==** 0 **or** isValidPos**(**p**)){**  
          **return** **-**1**;**  
     **}**  
  
     **return** p**-**1**;**  
  
  
**}**  
  
**template** **<class** T**,**int ARRAYSIZE**>**  
int List**<**T**,** ARRAYSIZE**>::**getNextPos**(const** int**&** p**)** **const{**  
     **if(**p **==** 0 **or** **!**isValidPos**(**p**)){**  
          *//problema*     **}**  
  
     **return** 0**+**1**;**  
**}**  
  
  
**template** **<class** T**,**int ARRAYSIZE**>**  
int List**<**T**,** ARRAYSIZE**>::**findDataLineal**(const** T**&** e**){**                          ///findDataLineal  
     int i**(**0**);**  
       
     **while(**i **<=** last**){**  
          **if(** data**[**i**]** **==** e**){**  
               **return** i**;**  
          **}**  
            
          i**++;**  
     **}**  
       
     **return** **-**1**;**  
**}**  
  
**template** **<class** T**,**int ARRAYSIZE**>**  
int List**<**T**,** ARRAYSIZE**>::**findDataBinaria**(const** T**&** e**){**                     ///findDataBinaria  
     int i**(**0**),** j**(**last**),** m**;**  
       
     **while(**i **<=** j**){**  
          m **=** **(**i**+**j**)/**2**;**  
            
          **if** **(**data**[**m**]** **==** e**){**  
               **return** m**;**  
          **}**  
            
          **if** **(**e **<** data**[**m**]){**  
               j **=** m**-**1**;**  
          **}else{**  
               i **=** m**+**1**;**  
          **}**  
     **}**  
       
     **return** **-**1**;**  
**}**  
  
**template** **<class** T**,**int ARRAYSIZE**>**  
void List**<**T**,** ARRAYSIZE**>::**sortDataBubbleByName **(** **)** **{**                                   ///Ordenarmiento tipo bubble  
       
     int i**(**last**)** **,** j**;**  
       
     bool flag**;**  
       
     **do{**  
          flag **=** **false;**  
          j **=** 0**;**  
            
          **while(**j **<** i**){**  
               **if(**data**[**j**].**getSongName**()** **>** data**[**j**+**1**].**getSongName**()){**  
                    swapData**(**data**[**j**],** data**[**j **+** 1**]);**  
                      
                    flag **=** **true;**  
               **}**  
               j**++;**  
          **}**  
          i**--;**  
     **}while(**flag**);**  
**}**  
  
**template** **<class** T**,**int ARRAYSIZE**>**  
void List**<**T**,** ARRAYSIZE**>::**sortDataBubbleByAuthor **(** **)** **{**                                   ///Ordenarmiento tipo bubble  
       
     int i**(**last**)** **,** j**;**  
       
     bool flag**;**  
       
     **do{**  
          flag **=** **false;**  
          j **=** 0**;**  
            
          **while(**j **<** i**){**  
               **if(**data**[**j**].**getAuthor**()** **>** data**[**j**+**1**].**getAuthor**()){**  
                    swapData**(**data**[**j**],** data**[**j **+** 1**]);**  
                      
                    flag **=** **true;**  
               **}**  
               j**++;**  
          **}**  
          i**--;**  
     **}while(**flag**);**  
**}**  
  
**template** **<class** T**,**int ARRAYSIZE**>**  
void List**<**T**,** ARRAYSIZE**>::**sortDataInsertByName **(** **)** **{**                                   ///Ordenamiento tipo Insercion  
       
     int i**(**1**),** j**;**  
       
     T aux**;**  
       
     **while(**i **<=** last**){**  
          aux **=** data**[**i**];**  
          j **=** i**;**  
            
          **while(**j **>** 0 **and** aux**.**getSongName**()** **<** data**[** j **-** 1**].**getSongName**()){**  
               data**[**j**]** **=** data**[**j **-** 1**];**  
                 
               j**--;**  
          **}**  
            
          **if(** i **!=** j**){**  
               data**[**j**]** **=** aux**;**  
          **}**  
            
          i**++;**  
     **}**  
**}**  
  
**template** **<class** T**,**int ARRAYSIZE**>**  
void List**<**T**,** ARRAYSIZE**>::**sortDataInsertByAuthor **(** **)** **{**                                   ///Ordenamiento tipo Insercion  
       
     int i**(**1**),** j**;**  
       
     T aux**;**  
       
     **while(**i **<=** last**){**  
          aux **=** data**[**i**];**  
          j **=** i**;**  
            
          **while(**j **>** 0 **and** aux**.**getAuthor**()** **<** data**[** j **-** 1**].**getAuthor**()){**  
               data**[**j**]** **=** data**[**j **-** 1**];**  
                 
               j**--;**  
          **}**  
            
          **if(** i **!=** j**){**  
               data**[**j**]** **=** aux**;**  
          **}**  
            
          i**++;**  
     **}**  
**}**  
  
**template** **<class** T**,**int ARRAYSIZE**>**  
void List**<**T**,** ARRAYSIZE**>::**sortDataSelectByName **(** **)** **{**                                   ///Ordenamiento tipo Seleccion  
       
     int i**(**0**),** j**,** m**;**  
       
     **while(**i **<** last**){**  
            
          m **=** i**;**  
          j**=** i **+** 1**;**  
            
          **while(**j **<=** last**){**  
               **if(**data**[**j**].**getSongName**()** **<** data**[**m**].**getSongName**()){**  
                    m **=** j**;**  
               **}**  
               j**++;**  
          **}**  
            
          **if(**i **!=** m**){**  
               swapData**(**data**[**i**],** data**[**m**]);**  
          **}**  
            
          i**++;**  
     **}**  
**}**  
  
**template** **<class** T**,**int ARRAYSIZE**>**  
void List**<**T**,** ARRAYSIZE**>::**sortDataSelectByAuthor **(** **)** **{**                                   ///Ordenamiento tipo Seleccion  
       
     int i**(**0**),** j**,** m**;**  
       
     **while(**i **<** last**){**  
            
          m **=** i**;**  
          j**=** i **+** 1**;**  
            
          **while(**j **<=** last**){**  
               **if(**data**[**j**].**getAuthor**()** **<** data**[**m**].**getAuthor**()){**  
                    m **=** j**;**  
               **}**  
               j**++;**  
          **}**  
            
          **if(**i **!=** m**){**  
               swapData**(**data**[**i**],** data**[**m**]);**  
          **}**  
            
          i**++;**  
     **}**  
**}**  
  
**template** **<class** T**,**int ARRAYSIZE**>**  
void List**<**T**,** ARRAYSIZE**>::**sortDataShellByName **(** **)** **{**                                        ///Ordenamiento tipo Shell  
       
     float factor**(**1.0**/**2.0**);**  
     int dif**((**last **+** 1**)\*** factor**),** i**,**j**;**  
       
     **while(**dif **>** 0**){**  
            
          i **=** dif**;**  
          **while(**i **<=** last**){**  
                 
               j **=** i**;**  
               **while(**j **>=** dif **and** data**[**j **-** dif**].**getSongName**()** **>** data**[**j**].**getSongName**()){**  
                      
                    swapData**(**data**[**j **-** dif**],** data**[**j**]);**  
                      
                    j **-=** dif**;**  
               **}**  
                 
               i**++;**  
          **}**  
            
          dif**\*=** factor**;**  
     **}**  
**}**  
  
**template** **<class** T**,**int ARRAYSIZE**>**  
void List**<**T**,** ARRAYSIZE**>::**sortDataShellByAuthor **(** **)** **{**                                        ///Ordenamiento tipo Shell  
       
     float factor**(**1.0**/**2.0**);**  
     int dif**((**last **+** 1**)\*** factor**),** i**,**j**;**  
       
     **while(**dif **>** 0**){**  
            
          i **=** dif**;**  
          **while(**i **<=** last**){**  
                 
               j **=** i**;**  
               **while(**j **>=** dif **and** data**[**j **-** dif**].**getAuthor**()** **>** data**[**j**].**getAuthor**()){**  
                      
                    swapData**(**data**[**j **-** dif**],** data**[**j**]);**  
                      
                    j **-=** dif**;**  
               **}**  
                 
               i**++;**  
          **}**  
            
          dif**\*=** factor**;**  
     **}**  
**}**  
  
**template** **<class** T**,**int ARRAYSIZE**>**  
T List**<**T**,** ARRAYSIZE**>::**retrieve**(const** int**&** p**)** **{**                                   ///recuperar  
       
     **if(!**isValidPos**(**p**)){**  
          **throw** ListException**(**"Posicion invalida, retrieve"**);**  
     **}**  
       
     **return** data**[**p**];**  
**}**  
  
  
**template** **<class** T**,**int ARRAYSIZE**>**  
std**::**string List**<**T**,** ARRAYSIZE**>::**toString **(** **)** **const{**  
  
     std**::**string result**;**  
     int i**(**0**);**  
  
     **while(**i **<=** last**){**  
            
          result **+=** data**[**i**].**toString**();**  
            
          i**++;**  
     **}**  
       
     **return** result**;**  
       
**}**  
  
  
**template** **<class** T**,**int ARRAYSIZE**>**  
void List**<**T**,** ARRAYSIZE**>::**deleteAll **(** **)** **{**  
     last **=** **-**1**;**  
**}**  
  
  
**template** **<class** T**,**int ARRAYSIZE**>**  
void List**<**T**,** ARRAYSIZE**>::**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** ListException**(**message**);**  
     **}**  
       
     int i**(**0**);**  
       
     **while(**i **<=** last**){**  
          myFile **<<** data**[**i**++];**  
     **}**  
       
     myFile**.**close**();**  
**}**  
  
**template** **<class** T**,**int ARRAYSIZE**>**  
void List**<**T**,** ARRAYSIZE**>::**readFromDisk**(const** std**::**string**&** fileName**){**  
       
     std**::**ifstream myFile**;**  
       
     T myData**;**  
       
     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(**ListException ex**)** **{**  
          myFile**.**close**();**  
            
          std**::**string message**;**  
          message **+=**fileName**;**  
          message **+=** " : "**;**  
          message **+=** ex**.**what**();**  
          message **+=** ", readFromDisk"**;**  
            
          **throw** ListException **(**message**);**  
     **}**  
       
     int i**(**0**);**  
       
     **while(**i **<=** last**){**  
          myFile **>>** data**[**i**];**  
          i**++;**  
     **}**  
       
     deleteAll**();**  
       
       
     **try{**  
          **while(**myFile **>>** myData**){** *// myFile >> myData returna un falso o verdadero*               insertData**(**getLastPos**(),** myData**);**  
          **}**  
     **}** **catch(**ListException ex**)** **{**  
          myFile**.**close**();**  
            
          std**::**string message**(**"Error durante la lectura del archivo."**);**  
            
          message **+=**fileName**;**  
          message **+=** " : "**;**  
          message **+=** ex**.**what**();**  
          message **+=** ", readFromDisk"**;**  
            
          **throw** ListException **(**message**);**  
     **}**  
       
     myFile**.**close**();**  
**}**  
  
  
**template** **<class** T**,**int ARRAYSIZE**>**  
List**<**T**,** ARRAYSIZE**>&** List**<**T**,** ARRAYSIZE**>::operator** **=** **(const** List**&** l**)** **{**  
       
     copyAll**(**l**);**  
       
     **return** **\*this;**  
**}**  
  
#endif

#ifndef SONG\_H  
#define SONG\_H  
  
#include <string>  
#include <iostream>  
  
**class** Song**{**  
**private:**  
       
     std**::**string songName**;**  
     std**::**string author**;**  
     std**::**string ranking**;**  
     bool flag**;**  
       
**public:**  
       
     Song**();**  
     Song**(const** Song**&);**  
       
     Song**&** **operator** **=** **(const** Song**&);**  
       
     std**::**string getSongName**()** **;**  
     std**::**string getAuthor**()** **;**  
     std**::**string getRanking**()** **;**  
       
     std**::**string toString**()** **const;**  
       
     void setSongName**(const** std**::**string**&** **);**  
     void setAuthor**(const** std**::**string**&** **);**  
     void setRanking**(const** std**::**string**&** **);**  
       
     bool **operator** **==** **(const** Song**&)** **const;**  
     bool **operator** **!=** **(const** Song**&)** **const;**  
     bool **operator** **<** **(const** Song**&)** **const;**  
     bool **operator** **>** **(const** Song**&)** **const;**  
     bool **operator** **<=** **(const** Song**&)** **const;**  
     bool **operator** **>=** **(const** Song**&)** **const;**  
       
       
     **friend** std**::**ostream**&** **operator** **<<** **(**std**::**ostream**&,** Song**&);**  
     **friend** std**::**istream**&** **operator** **>>** **(**std**::**istream**&,** Song**&);**  
       
**};**  
  
#endif

#include "song.h"  
  
**using** **namespace** std**;**  
  
Song**::**Song **(** **)** **{** **}**  
  
Song**::**Song **(const** Song**&** s**)** **:** songName**(**s**.**songName**),** author**(**s**.**author**),** ranking**(**s**.**ranking**){**  
       
**}**  
  
Song**&** Song**::operator** **=** **(const** Song**&** s**)** **{**  
       
     songName **=** s**.**songName**;**  
     author **=** s**.**author**;**  
     ranking **=** s**.**ranking**;**  
       
     **return** **\*this;**  
       
**}**  
  
  
string Song**::**toString **(** **)** **const{**  
       
     string result**;**  
       
       
     result **=** "\t"**;**  
     result **+=** songName**;**  
     result **+=** " - "**;**  
     result **+=** author**;**  
     result **+=** ".mp3"**;**  
     result **+=** " - "**;**  
     result **+=** " #"**;**  
     result **+=** ranking**;**  
     result **+=** "\n"**;**  
  
       
     **return** result**;**  
       
**}**  
  
  
  
string Song**::**getSongName **(** **)** **{**  
     **return** songName**;**  
**}**  
  
string Song**::**getAuthor **(** **)** **{**  
     **return** author**;**  
**}**  
  
string Song**::**getRanking **(** **)** **{**  
     **return** ranking**;**  
**}**  
  
void Song**::**setSongName **(const** string**&** n**)** **{**  
     songName **=** n**;**  
**}**  
  
void Song**::**setAuthor **(const** string**&** a**)** **{**  
     author **=** a**;**  
**}**  
  
void Song**::**setRanking **(const** string**&** r**)** **{**  
     ranking **=** r**;**  
**}**  
  
  
bool Song**::operator** **==** **(const** Song**&** s**)** **const** **{**  
     **return** songName **==** s**.**songName **or** author **==** s**.**author **or** ranking **==** s**.**ranking**;;**  
**}**  
  
bool Song**::operator** **!=** **(const** Song**&** s**)** **const** **{**  
     **return** songName **!=** s**.**songName **or** author **!=** s**.**author**;**  
**}**  
  
bool Song**::operator** **<** **(const** Song**&** s**)** **const** **{**  
     **return** songName **<** s**.**songName**;**  
**}**  
  
bool Song**::operator** **>** **(const** Song**&** s**)** **const** **{**  
     **return** songName **>** s**.**songName**;**  
**}**  
  
bool Song**::operator** **<=** **(const** Song**&** s**)** **const** **{**  
     **return** songName **<=** s**.**songName **or** author **<=** s**.**author**;**  
**}**  
  
bool Song**::operator** **>=** **(const** Song**&** s**)** **const** **{**  
     **return** songName **>=** s**.**songName **or** author **>=** s**.**author**;**  
**}**  
  
ostream**&** **operator** **<<** **(**ostream**&** os**,** Song**&** s**){**  
       
     os **<<** s**.**songName **<<** endl**;**  
     os **<<** s**.**author **<<** endl**;**  
     os **<<** s**.**ranking **<<** endl**;**  
       
     **return** os**;**  
**}**  
  
istream**&** **operator** **>>** **(**istream**&** is**,** Song**&** s**){**  
       
     getline**(**is**,** s**.**songName**);**  
     getline**(**is**,** s**.**author**);**  
     getline**(**is**,** s**.**ranking**);**  
       
     **return** is**;**  
**}**