**Actividad de Aprendizaje 05. Métodos de Búsqueda**

***Problema:***

Reutilice el resultado de la actividad 03, y adáptela a una nueva necesidad: almacenar toda la discografía de la radiodifusora que consta de cerca de 3,000 títulos más los que se vayan añadiendo. La lista servirá para las complacencias, por lo que debe contar también con el nombre del archivo MP3. Las complacencias se hacen a través del nombre de la canción o del nombre del cantante.

Haga un programa que realice la búsqueda en cualquiera de las dos formas y regrese el nombre del archivo MP3 correspondiente.

***Requerimientos:***

a)      El estilo de programación debe ser Orientado a Objetos

b)      Debe ofrecer dos opciones de búsqueda: lineal y binaria

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

Christopher Ceballos Jimenez

219750442

Estructura de datos I

El programa es capaz de coger un archivo con canciones anteriormente guardadas en el disco duro y además guardar canciones creando un archivo nuevo. Así como puede buscar la canción deseado por nombre o apellido con el método lineal, el método binario no trabaja de forma correcta aún. La actividad me resulto divertida y desarrolle nuevos conocimientos.

LISTA DE CANCIONES

amnesia

inspector

414

aunque no sea conmigo

enrique bunbury

1267

believer

imagine dragons

2413

donna

the lumineers

1299

firebird

milky chance

324

gimme the power

molotov

669

holiday

green day

1313

ilussia

mago de oz

706

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

peque¤o 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

universe

don diablo

218

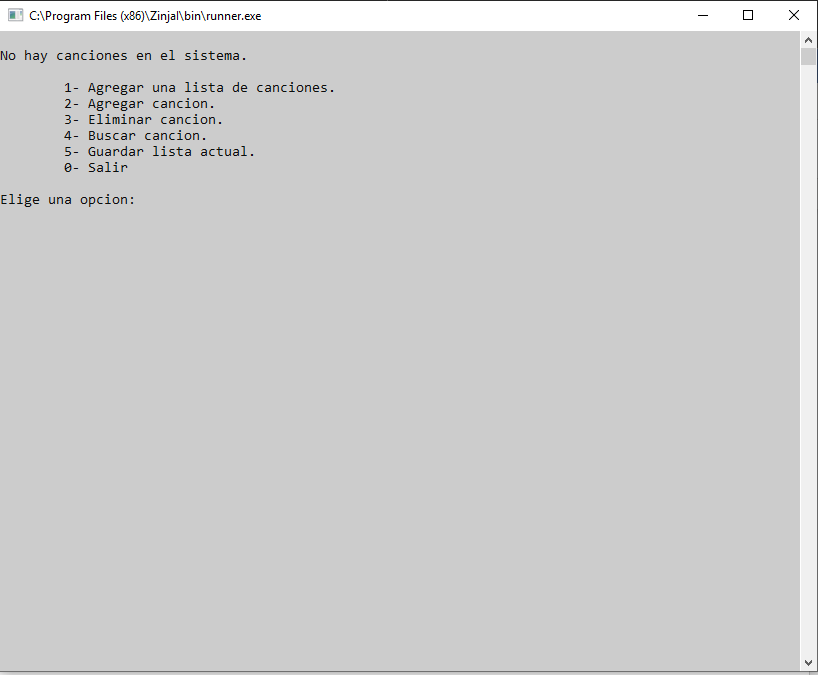
#include <iostream>  
#include <string>  
#include <random>  
#include <chrono>  
#include <functional>  
  
  
#include "list.h"  
#include "song.h"  
  
**using** **namespace** std**;**  
  
void addSong**();**  
void deleteSong**();**  
void findSong**();**  
  
int randNum**;**  
string myStr**;**  
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- Agregar cancion." **<<** endl**;**  
          cout **<<** "\t3- Eliminar cancion." **<<** endl**;**  
          cout **<<** "\t4- Buscar cancion." **<<** endl**;**  
          cout **<<** "\t5- Guardar lista actual." **<<** 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"**);**  
               addSong**();**  
                 
               **break;**  
                 
          **case** '3'**:**  
               system**(**"cls"**);**  
               deleteSong**();**  
                 
               **break;**  
                 
          **case** '4'**:**  
               system**(**"cls"**);**  
               findSong**();**  
                 
               **break;**  
                 
          **case** '5'**:**  
               **{**       
                    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;**  
                 
          **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**);**  
          mySong**.**setSongName**(**myStr**);**  
            
          cout**<<** "Artista: "**;**  
          fflush**(**stdin**);**getline**(**cin**,** myStr**);**  
          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\t >>:"**;** cin **>>** op**;**  
       
     **switch(**op**){**  
     **case** '1'**:**  
          **{**  
               string myStr**;**  
               int pos**;**  
                 
               cout **<<** "\nCancion a buscar: "**;**  
               fflush**(**stdin**);** getline**(**cin **,** myStr**);**  
               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'**:**  
          **{**  
               myLista**.**sortDataBubble**();**  
                 
               string myStr**;**  
               int pos**;**  
                 
               cout **<<** "\nCancion a buscar: "**;**  
               fflush**(**stdin**);** getline**(**cin **,** myStr**);**  
               mySong**.**setSongName**(**myStr**);**  
               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;**  
            
     **default:**  
          cout **<<** "\nOpcion invalida, intentelo de nuevo." **<<** endl**;**  
          **break;**  
     **}**  
  
**}**

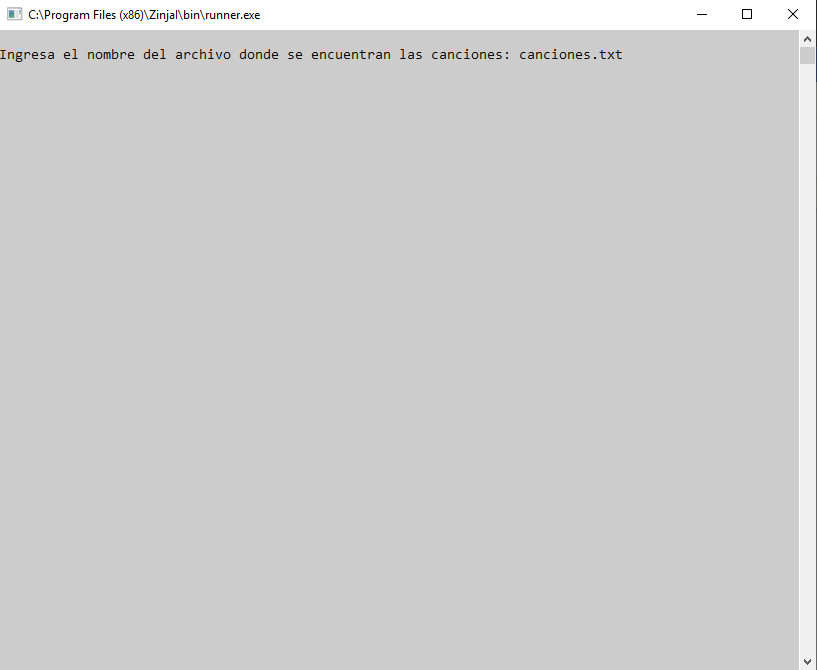
#ifndef LIST\_H\_INCLUDED  
#define LIST\_H\_INCLUDED  
  
#include <iostream>  
#include <string>  
#include <exception>  
#include <fstream>  
  
#include "song.h"  
  
  
**template** **<class** T**,** int ARRAYSIZE **=** 5000**>**  
**class** List **{**  
**private:**  
     T data**[**ARRAYSIZE**];**  
     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 sortDataBubble**();**  
     */\*void sortDataInsert();  
     void sortDataSelect(); // entregable 6  
     void sortDataShell();\*/*  
  
     T retrieve**(const** int**&);**  
  
     void sortData**();**  
  
     std**::**string toString**()** **const;**  
  
     void deleteAll**();**  
       
     void writeToDisk**(const** std**::**string**&);**  
     void readFromDisk**(const** std**::**string**&);**  
       
     List**&** **operator** **=** **(const** List**&);**  
       
     bool **operator** **==** **(const** List**&)** **const;**  
     bool **operator** **!=** **(const** List**&)** **const;**  
     bool **operator** **<** **(const** List**&)** **const;**  
     bool **operator** **>** **(const** List**&)** **const;**  
     bool **operator** **<=** **(const** List**&)** **const;**  
     bool **operator** **>=** **(const** List**&)** **const;**  
       
**};**  
  
*//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**>::**sortDataBubble **(** **)** **{**  
       
     int i**(**last**)** **,** j**;**  
       
     bool flag**;**  
       
     **do{**  
          flag **=** **false;**  
          j **=** 0**;**  
            
          **while(**j **<** i**){**  
               **if(**data**[**j**]** **>** data**[**j**+**1**]){**  
                    swapData**(**data**[**j**],** data**[**j **+** 1**]);**  
                      
                    flag **=** **true;**  
               **}**  
               j**++;**  
          **}**  
          i**--;**  
     **}while(**flag**);**  
**}**  
  
  
**template** **<class** T**,**int ARRAYSIZE**>**  
T List**<**T**,** ARRAYSIZE**>::**retrieve**(const** int**&** p**)** **{**  
       
     **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;**  
**}**  
  
  
**template** **<class** T**,**int ARRAYSIZE**>**  
bool List**<**T**,** ARRAYSIZE**>::operator** **==** **(const** List**&** l**)** **const** **{**  
       
     int i**(**0**);**  
     **while(**i **<=** last**){**  
          **return** data**[**i**]** **==** l**.**data**[**i**];**  
     **}**  
**}**  
  
  
**template** **<class** T**,**int ARRAYSIZE**>**  
bool List**<**T**,** ARRAYSIZE**>::operator** **!=** **(const** List**&** **)** **const** **{**  
       
**}**  
  
  
**template** **<class** T**,**int ARRAYSIZE**>**  
bool List**<**T**,** ARRAYSIZE**>::operator** **<** **(const** List**&** **)** **const** **{**  
       
**}**  
  
  
**template** **<class** T**,**int ARRAYSIZE**>**  
bool List**<**T**,** ARRAYSIZE**>::operator** **>** **(const** List**&** l**)** **const** **{**  
  
**}**  
  
  
**template** **<class** T**,**int ARRAYSIZE**>**  
bool List**<**T**,** ARRAYSIZE**>::operator** **<=** **(const** List**&** **)** **const** **{**  
       
**}**  
  
  
**template** **<class** T**,**int ARRAYSIZE**>**  
bool List**<**T**,** ARRAYSIZE**>::operator** **>=** **(const** List**&** **)** **const** **{**  
       
**}**  
  
#endif

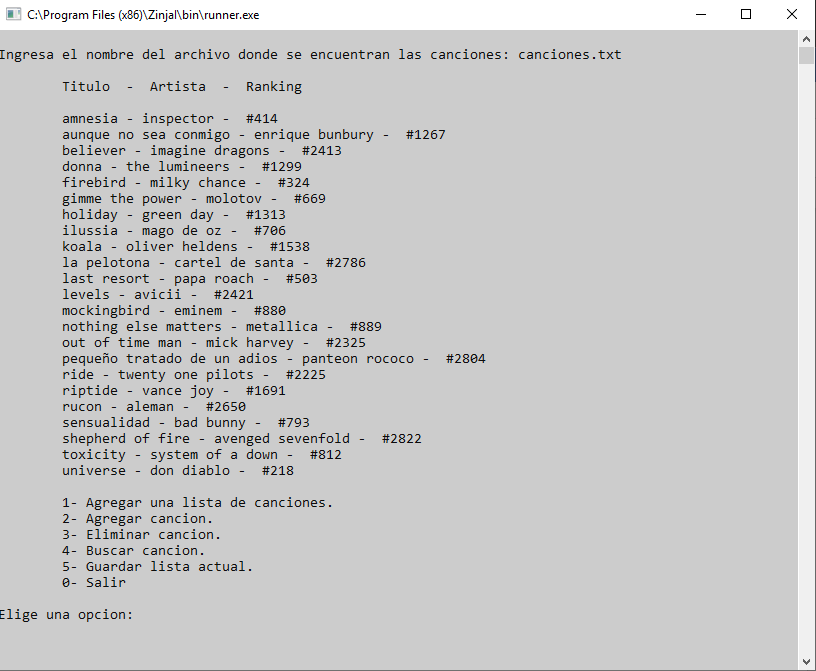
#ifndef SONG\_H  
#define SONG\_H  
  
#include <string>  
#include <iostream>  
  
**class** Song**{**  
**private:**  
       
     std**::**string songName**;**  
     std**::**string author**;**  
     std**::**string ranking**;**  
       
**public:**  
       
     Song**();**  
     Song**(const** Song**&);**  
       
     Song**&** **operator** **=** **(const** Song**&);**  
       
     std**::**string getSongName**()** **;**  
     std**::**string getaAuthor**()** **;**  
     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

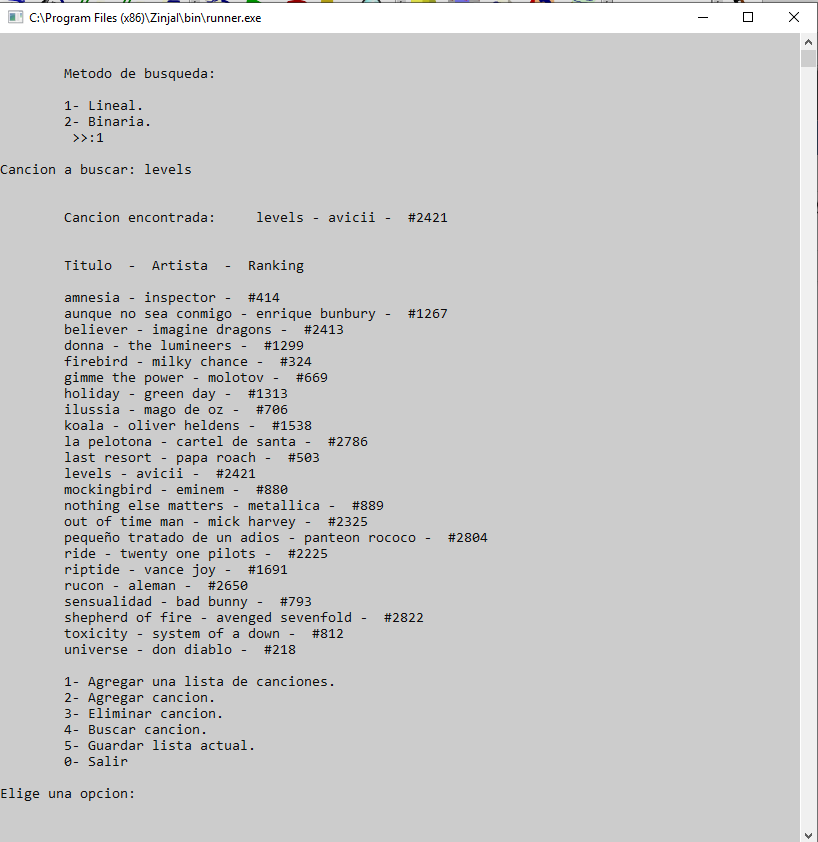
#include "song.h"  
  
**using** **namespace** std**;**  
  
Song**::**Song **(** **)** **{** **}**  
  
Song**::**Song **(const** Song**&** s**)** **:** songName**(**s**.**songName**),** author**(**s**.**author**),** ranking**(**s**.**ranking**){**  
       
**}**  
  
Song**&** Song**::operator** **=** **(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 **+=** " - "**;**  
     result **+=** " #"**;**  
     result **+=** ranking**;**  
     result **+=** "\n"**;**  
  
       
     **return** result**;**  
       
**}**  
  
  
  
string Song**::**getSongName **(** **)** **{**  
     **return** songName**;**  
**}**  
  
string Song**::**getaAuthor **(** **)** **{**  
     **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**;**  
**}**  
  
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**;**  
**}**  
  
bool Song**::operator** **>** **(const** Song**&** s**)** **const** **{**  
     **return** songName **>** s**.**songName **or** 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**;**  
**}**

CAPTURAS DE PANTALLA









Búsqueda por autor

