**Actividad de Aprendizaje 09. La Lista, implementación dinámica simplemente ligada**

***Problema:***

Tome el problema y requerimientos de la actividad 05 (sólo con búsqueda lineal), y cubra las necesidades utilizando una lista simplemente ligada en lugar de una lista estática.

***Requerimientos:***

a)      El estilo de programación debe ser Orientado a Objetos

b)      Considere de forma separada la clase *Nodo* respecto de la clase que servirá para instanciar los datos que almacena la lista

c)       La clase *Lista* y todas sus operaciones deberán alojarse en una librería, separándola del resto del programa

d)      El uso de la Lista debe hacerse exclusivamente a través de sus respectivos métodos

e)      Las operaciones a implementar, independientemente de que sean utilizadas o no en éste programa son: inicializa, vacía, llena, insertar, elimina, recupera, busca (lineal), primero, último, anterior, siguiente, y anula

Christopher Ceballos Jimenez

Estructuras de Datos I

Yo utilice la lista simplemente ligada lineal sin encabezado. Me resulto algo sencillo a excepción de los métodos para poder leer y escribir del disco, tuve que investigar un poco mas a fondo el funcionamiento de la librería istream y ofstream. Después de concluir con la lectura y escritura le di unos cuantos retoque al programa (estéticos) y eso fue todo.

Lista para lectura del disco.

Amnesia

Inspector

50

Aunque Que No Sea Conmigo

Enrique Bunbury

21

Believer

Imagine Dragons

7

Donna

The Lumineers

5

Firebird

Milky Chance

6

Holiday

Green Day

44

Ilussia

Mago De Oz

33

Koala

Oliver Heldens

22

La Pelotona

Cartel De Santa

10

Last Resort

Rapa Roach

9

Levels

Avicii

19

Mockingbird

Eminem

1

Nothing else matters

Metallica

12

Out Of Time Man

Mick Harvey

2

Pequeño Tratado De Un Adios

Panteon Rococo

37

Ride

Twenty One Pilots

27

Riptide

Vance Joy

8

Rucon

Aleman

100

Sensualidad

Bad Bunny

-100

Shepherd Of Fire

Avenged Sevenfold

15

Universe

Don Diablo

61

Best Friend

Foster The People

1141

First Of The Year

Skrillex

2021

Perfection

Twoloud

2953

The Anthem

Good Charlotte

1844

Sweet Sun

Milky Chance

1405

Break Ya Neck

INF1N1TE

1376

Take Me To Church

Hozier

2316

Lista para utilizar como inyección de texto

amnesia

inspector

1

aunque no sea conmigo

enrique bunbury

1

believer

imagine dragons

1

donna

the lumineers

1

firebird

milky chance

1

gimme the power

molotov

1

holiday

green day

1

ilussia

mago de oz

1

koala

oliver heldens

1

la pelotona

cartel de santa

1

last resort

papa roach

1

levels

avicii

1

mockingbird

eminem

1

nothing else matters

metallica

1

out of time man

mick harvey

1

peque¤o tratado de un adios

panteon rococo

1

ride

twenty one pilots

1

riptide

vance joy

1

rucon

aleman

1

sensualidad

bad bunny

1

shepherd of fire

avenged sevenfold

1

toxicity

system of a down

1

universe

don diablo

0

**MAIN**

#include <iostream>  
#include <string>  
#include <random>  
#include <chrono>  
#include <functional>  
#include <conio.h>  
  
#include "list.h"  
#include "song.h"  
  
**using** **namespace** std**;**  
  
void addSong**(**List **<**Song**>&);**  
void deleteSong**(**List **<**Song**>&);**  
void findSong**(**List **<**Song**>&);**  
  
void enterToContinue**(){**  
     cout **<<** "\nPresiona ENTER para continuar."**;** getch**();**  
**}**  
  
int main **()** **{**  
       
     char op**;**  
       
     List **<**Song**>** myLista**;**  
     string savedSongs**;**  
       
     **do{**  
          system**(**"cls"**);**  
          **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 cancion." **<<** endl**;**  
          cout **<<** "\t2- Buscar cancion." **<<** endl**;**  
          cout **<<** "\t3- Eliminar cancion." **<<** endl**;**  
          cout **<<** "\t4- Agregar una lista de canciones." **<<** endl**;**  
          cout **<<** "\t5- Guardar lista actual." **<<** endl**;**  
          cout **<<** "\t0- Salir" **<<** endl**;**  
          cout **<<** "\nElige una opcion: "**;** cin **>>** op**;**  
            
          **switch(**op**){**  
          **case** '4'**:**  
                 
               system**(**"cls"**);**  
                 
               cout **<<** "\nIngresa el nombre del archivo donde se encuentran las canciones: "**;**  
               fflush**(**stdin**);** getline**(**cin**,** savedSongs**);**  
                 
               myLista**.**readFromDisk**(**savedSongs**);**  
                 
               **break;**  
                 
          **case** '1'**:**  
               addSong**(**myLista**);**  
                 
               **break;**  
                 
          **case** '3'**:**  
               system**(**"cls"**);**  
               deleteSong**(**myLista**);**  
                 
               **break;**  
                 
          **case** '2'**:**  
               system**(**"cls"**);**  
               findSong**(**myLista**);**  
                 
               **break;**  
                 
          **case** '5'**:**  
                 
               system**(**"cls"**);**  
               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**(**List**<**Song**>&** myLista**)** **{**  
       
     default\_random\_engine generator**(**chrono**::**system\_clock**::**now**().**time\_since\_epoch**().**count**());**  
     uniform\_int\_distribution**<**int**>** distribution**(**1**,**3000**);**  
     **auto** dice **=** bind**(**distribution**,** generator**);**  
       
     string myStr**;**  
     Song mySong**;**  
     char op**;**  
       
     **do{**  
          system**(**"cls"**);**  
          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**);**  
            
          **try{**  
               myLista**.**insertData**(**myLista**.**getLastPos**(),** mySong**);**  
          **}catch** **(**List**<**Song**>::**Exception ex**){**  
               cout **<<** "Algo salio mal." **<<** ex**.**what**();**  
          **}**  
            
          cout **<<** "\nDeseas agregar otra cancion? (1=Si, 0= No): "**;** cin **>>** op**;**  
            
     **}while(**op **!=** '0'**);**  
       
**}**  
  
void deleteSong**(**List**<**Song**>&** myLista**){**  
       
     char op**;**  
     Song mySong**;**  
     string myStr**;**  
       
     cout **<<** "\nNombre o Autor de la cancion a eliminar: "**;**  
     fflush**(**stdin**);** getline**(**cin **,** myStr**);**  
     mySong**.**setSongName**(**myStr**);**  
     mySong**.**setAuthor**(**myStr**);**  
       
     **do{**  
          system**(**"cls"**);**  
          cout **<<** "\n\tEliminar:" **<<** myLista**.**retrieve**(**myLista**.**findData**(**mySong**)).**toString**()** **<<** "\n\t1- Eliminar.\n\t0- Cancelar.\n\t>>: "**;**  
          cin **>>** op**;**  
            
          **switch(**op**){**  
          **case** '0'**:**  
               **break;**  
                 
          **case** '1'**:**  
                 
               **try{**  
                    myLista**.**deleteData**(**myLista**.**findData**(**mySong**));**  
                    cout **<<** "\nSe ha eliminado correctamente." **<<** endl**;**  
                    enterToContinue**();**  
               **}catch(**List**<**Song**>::**Exception ex**){**  
                    cout **<<** "\nOcurrio algo malo." **<<** ex**.**what**();**  
               **}**  
                 
               **return;**  
               **break;**  
                 
          **default:**  
               cout **<<** "\nOpcion invalida. Intenta nuevamente." **<<** endl**;**  
          **}**  
     **}while(**op **!=** '0'**);**  
**}**  
  
void findSong**(**List**<**Song**>&** myLista**){**  
       
     Song mySong**;**  
     string myStr**;**  
     Node**<**Song**>\*** pos**;**  
       
     cout **<<** "\nCancion a buscar: "**;**  
     fflush**(**stdin**);** getline**(**cin **,** myStr**);**  
     mySong**.**setSongName**(**myStr**);**  
     mySong**.**setAuthor**(**myStr**);**  
       
     pos **=** myLista**.**findData**(**mySong**);**  
       
     **if(**pos **==** **nullptr){**  
          cout **<<** "\nLa cancion " **<<** myStr **<<** " no se encuentra en la lista." **<<** endl**;**

enterToContinue();  
     **}else{**  
          cout **<<** "\n\n\tCancion encontrada: " **<<** myLista**.**retrieve**(**pos**).**toString**()** **<<** endl**;**  
          enterToContinue**();**  
     **}**  
  
**}**

**LIST.H**

#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**>\*** anchor**;**  
       
     void copyAll**(const** List**<**T**>&);**  
       
     bool isValidPos**(**Node**<**T**>\*)** **const;**  
       
**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**&);**  
       
**};**  
  
///Implementacion de la lista  
  
**template** **<class** T**>**  
void List**<**T**>::**copyAll **(const** List**<**T**>&** l**)** **{**  
       
     Node**<**T**>\*** aux**(**l**.**anchor**);**  
     Node**<**T**>\*** last**(nullptr);**  
     Node**<**T**>\*** newNode**;**  
       
     **while(**aux **!=** **nullptr){**  
            
          newNode **=** **new** Node**<**T**>(**aux**->**getData**());**  
            
          **if(**last **==** **nullptr){**  
               anchor **=** newNode**;**  
          **}**  
          **else{**  
               last**->**setNext**(**newNode**);**  
          **}**  
            
          last **=** newNode**;**  
            
          aux **=** aux**->**getNext**();**  
     **}**  
**}**  
  
**template** **<class** T**>**  
bool List**<**T**>::**isValidPos **(**Node**<**T**>\*** p**)** **const** **{**  
       
     Node**<**T**>\*** aux**(**anchor**);**  
       
     **while(**aux **!=** **nullptr){**  
            
          **if(**aux **==** p**){**  
               **return** **true;**  
          **}**  
            
          aux **=** aux **->** getNext**();**  
     **}**  
       
     **return** **false;**  
**}**  
  
**template** **<class** T**>**  
List**<**T**>::**List **(** **)** **:** anchor**(nullptr)** **{** **}**  
  
**template** **<class** T**>**  
List**<**T**>::**List **(const** List**<**T**>&** l**)** **:** anchor**(nullptr)** **{**  
     copyAll**(**l**);**  
**}**  
  
**template** **<class** T**>**  
List**<**T**>::~**List **(** **)** **{**  
     deleteAll**();**  
**}**  
  
**template** **<class** T**>**  
bool List**<**T**>::**isEmpty **(** **)** **const** **{**  
     **return** anchor **==** **nullptr;**  
**}**  
  
**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**(new** Node**<**T**>(**e**));**  
       
     **if(**aux **==** **nullptr){**  
          **throw** Exception**(**"Memoria no disponible, InsertData"**);**  
     **}**  
       
     **if(**p **==** **nullptr){** *//Insertamos al principio de la lista*          aux**->**setNext**(**anchor**);**  
          anchor **=** aux**;;**  
     **}**  
     **else{** *// Insert en cualquier otro lugar de la lista*          aux**->**setNext**(**p**->**getNext**());**  
          p**->**setNext**(**aux**);**  
     **}**  
       
**}**  
  
**template** **<class** T**>**  
void List**<**T**>::**deleteData **(**Node**<**T**>\*** p**)** **{**  
       
     **if(!**isValidPos**(**p**)){**  
          **throw** Exception**(**"Posicion invalida, DeleteData."**);**  
     **}**  
       
     **if(**p **==** anchor**){**  
          anchor **=** anchor **->** getNext**();**  
     **}**  
     **else{**  
          getPrevPos**(**p**)** **->** setNext**(**p **->** getNext**());**  
     **}**  
       
     **delete** p**;**  
**}**  
  
**template** **<class** T**>**  
Node**<**T**>\*** List**<**T**>::**getFirstPos **(** **)** **const** **{**  
     **return** anchor**;**  
**}**  
  
**template** **<class** T**>**  
Node**<**T**>\*** List**<**T**>::**getLastPos **(** **)** **const** **{**  
       
     **if(**isEmpty**()){**  
          **return** **nullptr;**  
     **}**  
       
     Node**<**T**>\*** aux**(**anchor**);**  
       
     **while(**aux**->**getNext**()** **!=** **nullptr){**  
          aux **=** aux**->**getNext**();**  
     **}**  
       
     **return** aux**;**  
**}**  
  
**template** **<class** T**>**  
Node**<**T**>\*** List**<**T**>::**getPrevPos **(**Node**<**T**>\*** p**)** **const** **{**  
       
     **if(**p **==** anchor**){**  
          **return** **nullptr;**  
     **}**  
       
     Node**<**T**>\*** aux**(**anchor**);**  
       
     **while(**aux **!=** **nullptr** **and** aux **->**getNext**()** **!=** p**){**  
          aux **=** aux**->**getNext**();**  
     **}**  
       
     **return** aux**;**  
**}**  
  
**template** **<class** T**>**  
Node**<**T**>\*** List**<**T**>::**getNextPos **(**Node**<**T**>\*** p**)** **const** **{**  
       
     **if(!**isValidPos**(**p**)){**  
          **return** **nullptr;**  
     **}**  
       
     **return** p**->**getNext**();**  
**}**  
  
**template** **<class** T**>**  
Node**<**T**>\*** List**<**T**>::**findData **(const** T**&** e**)** **const** **{**  
       
     Node**<**T**>\*** aux**(**anchor**);**  
       
     **while(**aux **!=** **nullptr** **and** aux**->** getData**()** **!=** e**){**  
          aux **=** aux **->** getNext**();**  
     **}**  
       
     **return** aux**;**  
**}**  
  
**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** **{**  
       
     Node**<**T**>\*** aux**(**anchor**);**  
     std**::**string result**;**  
       
     **while(**aux **!=** **nullptr){**  
          result **+=** aux**->**getData**().**toString**();**  
            
          aux **=** aux**->**getNext**();**  
     **}**  
       
     **return** result**;**  
**}**  
  
**template** **<class** T**>**  
void List**<**T**>::**deleteAll **(** **)** **{**  
       
     Node**<**T**>\*** aux**;**  
       
     **while(**anchor **!=** **nullptr){**  
            
          aux **=** anchor**;**  
            
          anchor **=** anchor**->**getNext**();**  
            
          **delete** aux**;**  
     **}**  
**}**  
  
**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**;**  
       
     **while(**anchor **!=** **nullptr){**  
          aux **=** anchor**;**  
          myFile **<<** aux **->** getData**().**getSong**();**  
          anchor **=** anchor **->** getNext**();**  
     **}**  
       
     myFile**.**close**();**  
**}**  
  
**template** **<class** T**>**  
void List**<**T**>::**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(**Exception ex**)** **{**  
          myFile**.**close**();**  
            
          std**::**string message**;**  
          message **+=**fileName**;**  
          message **+=** " : "**;**  
          message **+=** ex**.**what**();**  
          message **+=** ", readFromDisk"**;**  
            
          **throw** Exception **(**message**);**  
     **}**  
       
     deleteAll**();**  
       
       
     **try{**  
          **while(**myFile **>>** myData**){** *// myFile >> myData returna un falso o verdadero*               insertData**(**getLastPos**(),** myData**);**  
          **}**  
     **}** **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**();**  
**}**  
#endif

**NODE.H**

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

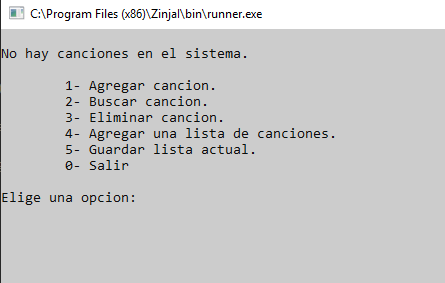
**SONG.H**

#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 getSong**();**  
       
     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

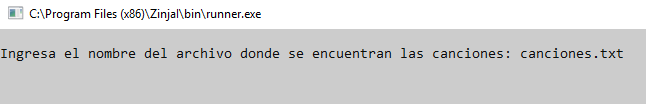
**SONG.CPP**

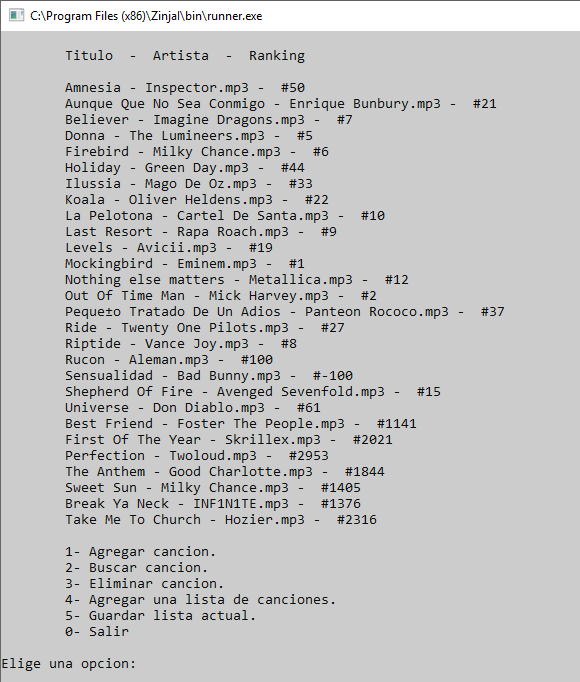
#include "song.h"  
  
**using** **namespace** std**;**  
  
Song**::**Song **(** **)** **{** **}**  
  
Song**::**Song **(const** Song**&** s**)** **:** songName**(**s**.**songName**),** author**(**s**.**author**),** ranking**(**s**.**ranking**)** **{** **}**  
  
Song**&** Song**::operator** **=**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

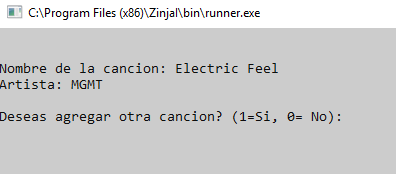
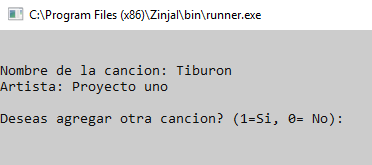
****

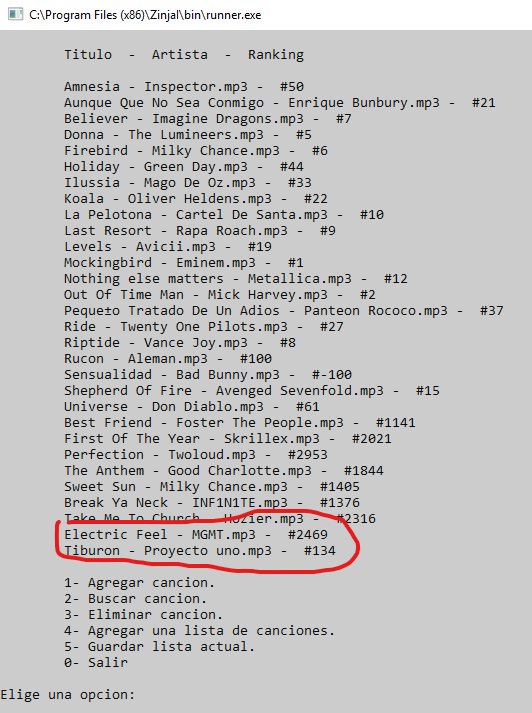
**Inyección de texto.**

****

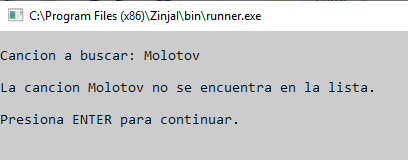
****

**Agregar canciones.**

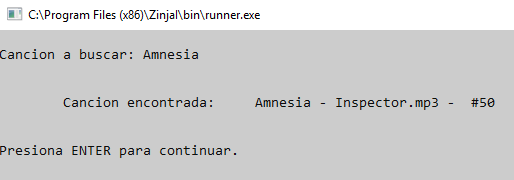
** **

****

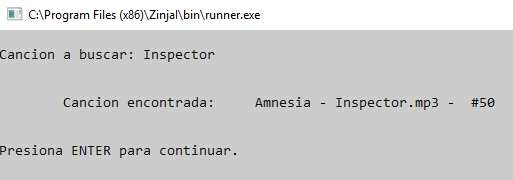
**Buscar canciones.**

****

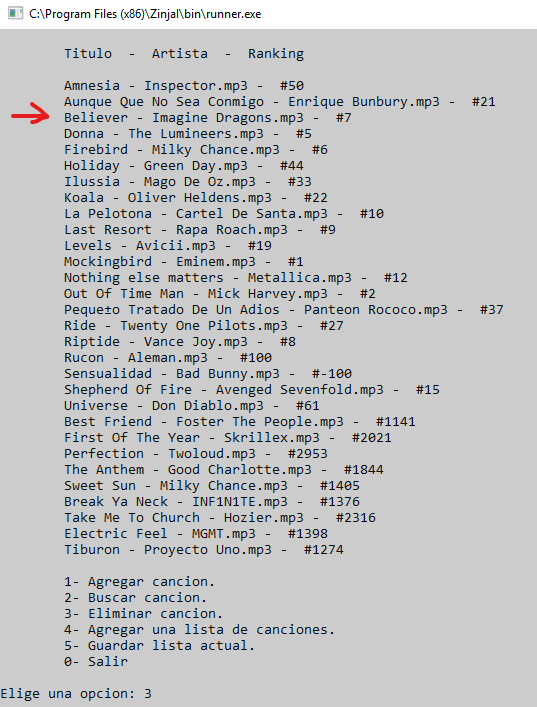
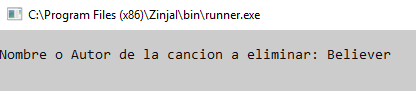
**Por nombre de la canción**

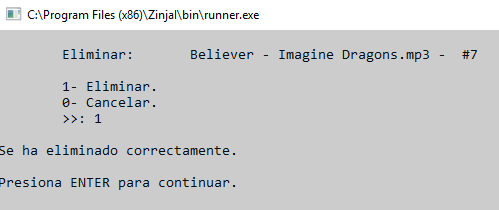
****

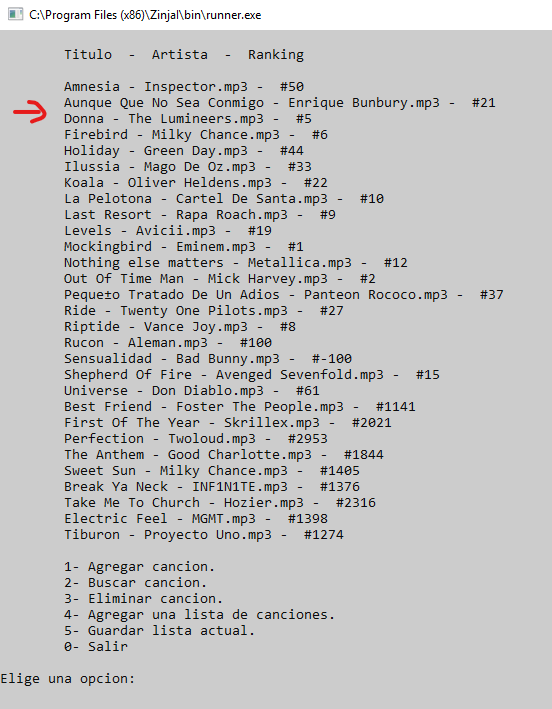
**Por autor de la canción**

****

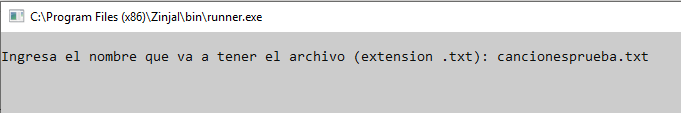
**Eliminar canciones.**

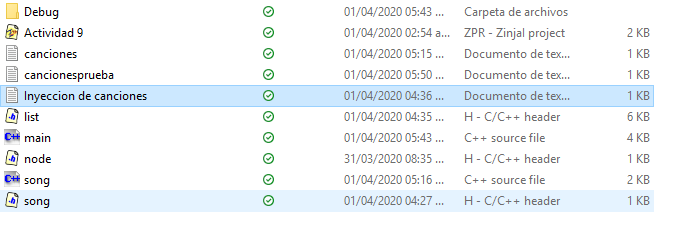
** **

****

****

**Leer al disco.**

****

****