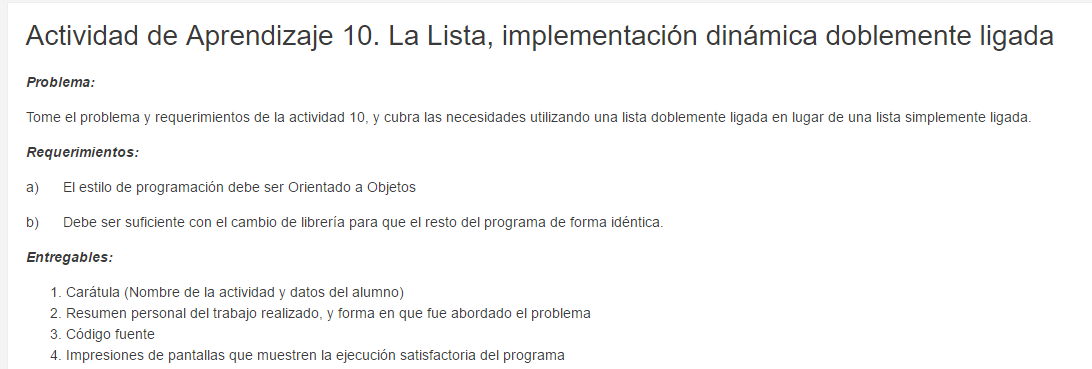
|  |
| --- |
| centro universitario de ciencias exáctas e ingenierías |
| Lista Ligada Doble |
| Tarea 09.2 |
|  |
| **Aldo Alexandro Vargas Meza 213495653** |
| **23/03/2017** |





En la última actividad, en los punteros, ya fue posible implementar la lista con todos sus métodos, de manera dinámica, ahora es cuestión de modificar los métodos para hacerla doblemente ligada.

Toda la estructura tiene pequeños cambios, pero funciona esencialmente igual.

**Tipo de dato Song.**

El tipo de dato Song es manejado como el dato principal a guardar en la lista, consta de 5 atributos: título, autor, disquera, género y mp3. Cada uno de ellos guarda una cadena de caracteres con información de la canción.

Los métodos que tiene la clase song, además de los getters y setters de cada uno de los atributos, son métodos de impresión y sobrecarga de operadores, así como dos constructores, uno con y otro sin parámetros.

**Clase Nodo**  
Para el manejo de los datos por medio de punteros, fue necesaria la entidad nodo, que consta del tipo de dato, y dos direcciones hacia otros elementos. Siguiendo esta lógica, se programó una clase Nodo, que contiene una dirección hacía enfrente en forma de puntero y un dato almacenado. Esta entidad conformará cada uno de los datos de la lista ligada.

Además de dos constructores, la entidad Nodo entre sus métodos contiene getters y setters para los atributos y una función especial de impresión.

Para la lista doblemente ligada, fue necesario implementar una entidad nodo con dos punteros, uno hacía el siguiente nodo y otro hacia el anterior.

**Clase Lista**

El cambio principal fue por la lista, la cual dejó de contener un dato en sí, sino un puntero llamado head, el cual al ser del tipo nodo contiene los atributos de la estructura.

La lista contiene una gran cantidad de funciones para el manejo de la misma, toda utilizando direcciones y apuntadores.

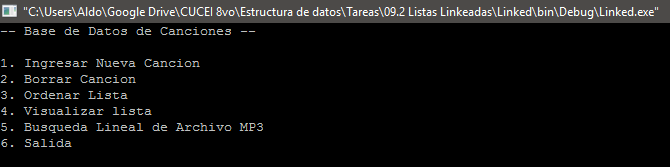
Dentro del archivo principal, se incluyeron funciones que manejan la lectura y escritura de datos en un archivo de texto, utilizado como base de datos.

**Manejo de Archivos**

Dentro del programa, hay funciones especializadas en escribir y leer de un archivo de texto, así como actualizarlo de ser necesario. Estas funciones se añadieron como una opción simple de tener una base de datos para pruebas del mismo programa.

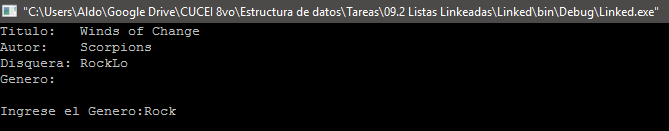
De esta forma, queda implementada en su totalidad, la lista doblemente ligada.

El menú principal:

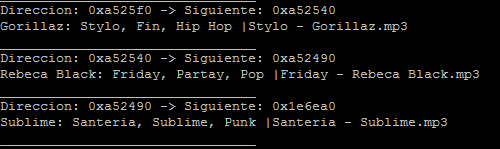


Al ingresar un nuevo dato, se modifica también un archivo de texto utilizado como base de datos.

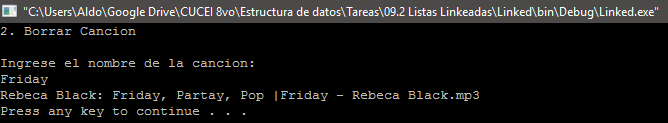
La asignación de nuevas direcciones en punteros se puede apreciar en la impresión.



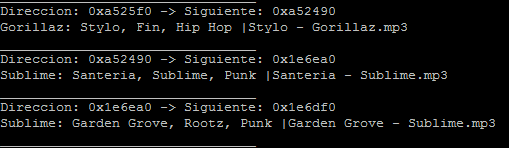
Para la eliminación de datos, es necesario ingresar el título de la canción. La función de manejo de archivo crea un archivo con una copia del mismo, omitiendo el dato a eliminar y luego borra el archivo original, para luego renombrar el archivo auxiliar.



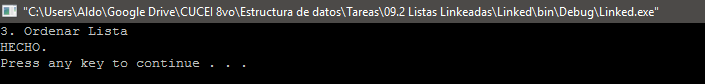
Al borrar el archivo Rebeca Black, Friday, la función localiza, encuentra el nodo a borrar y pasa como parámetro a la función borrar.



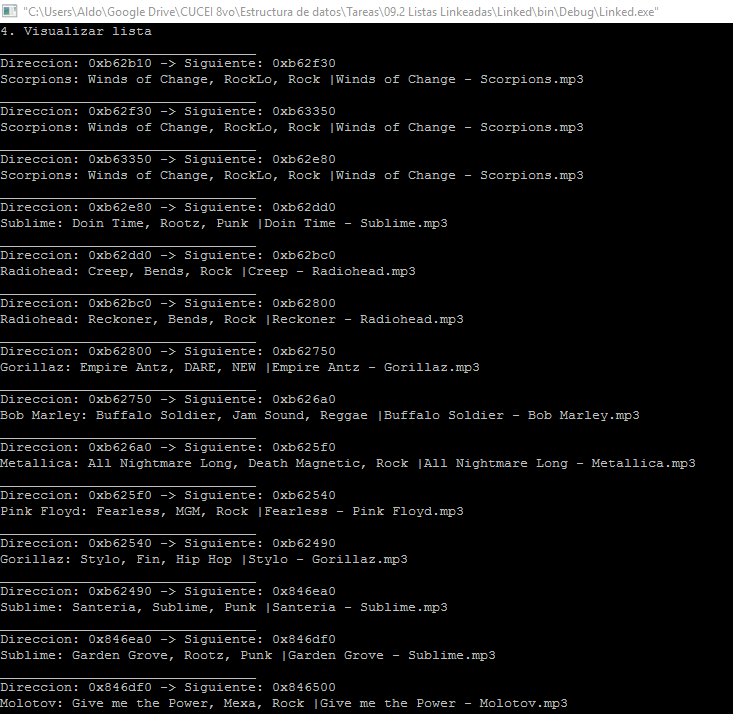
Podemos apreciar que las direcciones entre las canciones que están alrededor de la canción borrada, son manipuladas para seguir con el flujo de nodos.

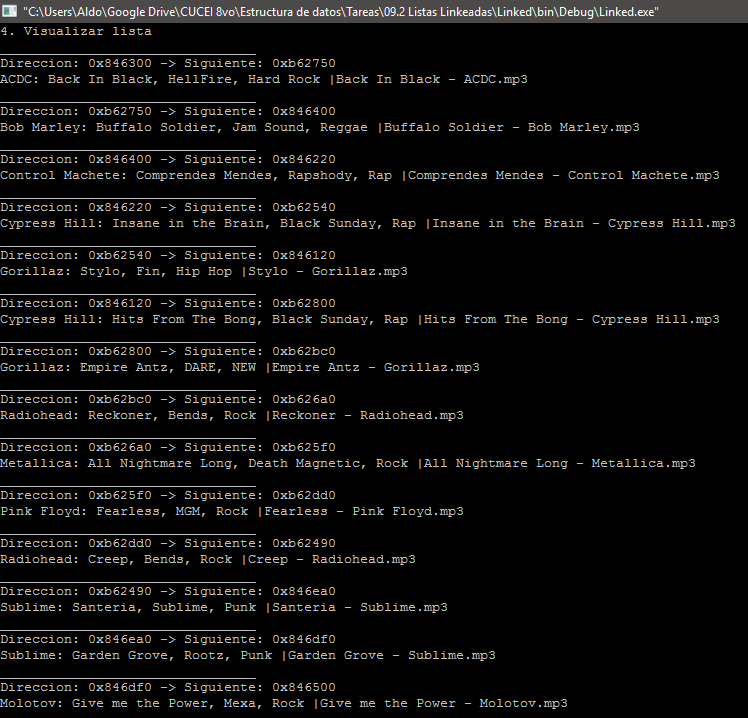


La opción de ordenamiento ejecuta un algoritmo quickSort para acomodar alfabéticamente por nombre de autor y título las canciones.



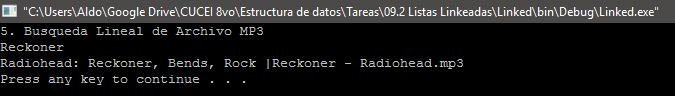
La impresión sin ordenar y ordenada:





La impresión de la lista se lleva a cabo con el método de print, que se mueve a través de los punteros e imprime todos los nodos ligados.

La búsqueda lineal, encuentra con la función localiza, por medio del nombre del título de la canción la información de la canción contenida en el nodo.



En lo personal, la lista doblemente ligada está a un paso mas de ser mejor, ya que con un puntero hacía el anterior, el manejo de los métodos es mas fácil.

**Codigos**

//main.cpp

#include <iostream>

#include <iomanip>

#include <fstream>

#include "linkedList.h"

using namespace std;

void FileHandlerRecover(LinkedList& l) {

ifstream lectura;

lectura.open("canciones.txt",ios::out | ios::in);

string indice;

string t, a, rl, g;

if(lectura.is\_open()) {

int i=-1;

std::getline(lectura, indice);

while(!lectura.eof()) {

std::getline(lectura, t);

std::getline(lectura, a);

std::getline(lectura, rl);

std::getline(lectura, g);

Song temp;

temp.setTitle(t);

temp.setAuthor(a);

temp.setRecordL(rl);

temp.setGenre(g);

temp.setMP3(a,t);

l.insertNode(nullptr, temp);

std::getline(lectura, indice);

}

}

}

void FileHandlerInsert(char t[100] = {}, char a[100]= {}, char rl[100]= {}, char g[100]= {}) {

ofstream escritura;

escritura.open("canciones.txt",ios::out | ios::app);

char i = '-';

if(escritura.is\_open()) {

escritura << i<<endl;

escritura << t << endl;

escritura << a << endl;

escritura << rl << endl;

escritura << g << endl;

}

escritura.close();

}

void FileHandlerDelete(string data) {

ofstream aux;

ifstream lectura;

string indice,t, a, rl, g;

aux.open("auxiliar.txt",ios::out);

lectura.open("canciones.txt",ios::in);

if(aux.is\_open() && lectura.is\_open()) {

int i=0;

while(!lectura.eof()) {

std::getline(lectura, indice);

std::getline(lectura, t);

std::getline(lectura, a);

std::getline(lectura, rl);

std::getline(lectura, g);

if(data != t ) {

aux << i <<endl;

aux << t << endl;

aux << a << endl;

aux << rl << endl;

aux << g << endl;

i++;

}

}

}

else {

cout<<"Error de Borrado."<<endl;

system("pause");

}

aux.close();

lectura.close();

remove("canciones.txt");

rename("auxiliar.txt","canciones.txt");

}

int main() {

LinkedList l;

FileHandlerRecover(l);

int estado = 0;

while(estado != 6) {

switch(estado) {

case 0: {

system("cls");

int opc;

cout<<"-- Base de Datos de Canciones --\n\n";

cout<<"1. Ingresar Nueva Cancion \n";

cout<<"2. Borrar Cancion (PENDIENTE)\n";

cout<<"3. Ordenar Lista \n";

cout<<"4. Visualizar lista \n";

cout<<"5. Busqueda Lineal de Archivo MP3 \n";

cout<<"6. Salida "<<endl;

cin >> estado;

}

break;

case 1: {

system("cls");

char t[100] = {},a[100]= {},rl[100]= {},g[100]= {},m[100]= {};

cout<<"1. Ingresar Nueva Cancion \n";

for(int i=0; i<5; i++) {

system("cls");

cout<<"Titulo: "<<t<<endl;

cout<<"Autor: "<<a<<endl;

cout<<"Disquera: "<<rl<<endl;

cout<<"Genero: "<<g<<endl<<endl;

switch(i) {

case 0:

cout<<"Ingrese el Titulo:";

cin.ignore();

cin.getline(t,100);

break;

case 1:

cout<<"Ingrese el Autor:";

cin.getline(a,100);

break;

case 2:

cout<<"Ingrese la Disquera:";

cin.getline(rl,100);

break;

case 3:

cout<<"Ingrese el Genero:";

cin.getline(g,100);

break;

}

}

Song newSong = l.createSong(t,a,rl,g);

l.insertNode(l.getLast(), newSong);

FileHandlerInsert(t,a,rl,g);

estado = 0;

}

break;

case 2: {

system("cls");

char buscar[100];

cout<<"2. Borrar Cancion \n\n";

cout<<"Ingrese el nombre de la cancion: "<<endl;

cin.ignore();

cin.getline(buscar,100);

Node\* aux;

cout<< l.locateByTitle(buscar)->getData().getSongInfo();

aux = l.locateByTitle(buscar);

l.deleteNode(aux);

FileHandlerDelete(buscar);

system("pause");

estado = 0;

}

break;

case 3: {

system("cls");

cout<<"3. Ordenar Lista \n";

l.sortData();

cout<<"HECHO.\n";

system("pause");

estado = 0;

}

break;

case 4: {

system("cls");

cout<<"4. Visualizar lista \n";

l.printList();

system("pause");

estado = 0;

}

break;

case 5: {

system("cls");

string printed;

char b[100] = {};

cout<<"5. Busqueda Lineal de Archivo MP3 \n";

cin.ignore();

cin.getline(b,100);

Node\* aux;

printed = l.locateByTitle(b)->getData().getSongInfo();

cout<< printed;

system("pause");

estado = 0;

}

break;

default:

estado = 0;

break;

}

}

return 0;

}

//NodeException.h

#ifndef NODEEXCEPTION\_H\_INCLUDED

#define NODEEXCEPTION\_H\_INCLUDED

#include <string>

class NodeException : public std::exception {

protected:

std::string msg;

public:

explicit NodeException(const char\* message) : msg(message) { }

explicit NodeException(const std::string& message) : msg(message) { }

virtual ~NodeException() throw () { }

virtual const char\* what() const throw () {

return msg.c\_str();

}

};

#endif // NODEEXCEPTION\_H\_INCLUDED

//linkedList.h

#ifndef LINKEDLIST\_H\_INCLUDED

#define LINKEDLIST\_H\_INCLUDED

#include "node.h"

class LinkedList {

private:

Node \*head;

public:

LinkedList(Node \*n);

LinkedList();

~LinkedList();

void insertNode(Node\* pos, Song data);

void deleteNode(Node\* pos);

void sortData();

void quickSort(Node \*\*arreglo, int izq, int der);

void nullData();

Node\* getFirst();

Node\* getLast();

Node\* getPrev(Node \*pos);

Node\* getNext(Node \*pos);

Node\* locateNode(Song ing);

Node\* locateByTitle(string);

bool isEmpty();

void printList();

Song createSong(const string&, const string&, const string&, const string&);

Song getData(Node \*pos);

};

#endif // LINKEDLIST\_H\_INCLUDED

//linkedList.cpp

#include "linkedList.h"

/\* Constructores \*/

LinkedList::LinkedList() {

head = nullptr;

}

LinkedList::LinkedList(Node \*n) {

head = n;

}

LinkedList::~LinkedList() {

nullData();

}

/\* Manejo de datos \*/

void LinkedList::insertNode(Node\* pos, Song data) {

Node \*aux = new Node();

aux->setData(data);

if(pos == nullptr) {

aux->setPrev(nullptr);

aux->setNext(head);

if(head != nullptr) {

head->setPrev(aux);

}

head = aux;

}

else {

aux->setPrev(pos);

aux->setNext(getNext(pos));

if(pos->getNext() != nullptr) {

pos->getNext()->setPrev(aux);

}

pos->setNext(aux);

}

}

void LinkedList::deleteNode(Node\* pos) {

if(isEmpty() || pos == nullptr) {

return;

}

if(pos->getPrev() != nullptr) {

pos->getPrev()->setNext(pos->getNext());

}

if(pos->getNext() != nullptr) {

pos->getNext()->setPrev(pos->getPrev());

}

if(pos == head) {

head = head->getNext();

}

delete pos;

}

void LinkedList::sortData() {

Node \*limit = nullptr;

Node \*aux = head;

int contador,i = 0;

while(aux != limit) {

contador++;

aux = aux->getNext();

}

Node \*\*arreglo;

arreglo = new Node \*[contador];

aux = head;

while(aux != limit) {

arreglo[i] = aux;

aux = aux->getNext();

i++;

}

quickSort(arreglo,0,contador-1);

i = 0;

while(i < contador-1) {

arreglo[i]->setNext(arreglo[i+1]);

i++;

}

arreglo[i]->setNext(limit);

arreglo[0]->setPrev(limit);

i = 1;

while(i < contador-1) {

arreglo[i]->setPrev(arreglo[i-1]);

i++;

}

head = arreglo[0];

delete arreglo;

}

void LinkedList::quickSort(Node \*\*arreglo, int izq, int der) {

int i, j, centro;

Node \*temp;

Node \*pivote;

i = izq;

j = der;

centro = (izq + der)/2;

pivote = arreglo[centro];

temp = nullptr;

while(i <= j) {

while(arreglo[i]->getData() < pivote->getData()) i++;

while(arreglo[j]->getData() > pivote->getData()) j--;

if(i <= j) {

temp = arreglo[i];

arreglo[i] = arreglo[j];

arreglo[j] = temp;

i++;

j--;

}

}

if(izq < j) {

quickSort(arreglo,izq,j);

}

else if(i < der) {

quickSort(arreglo, i, der);

}

}

void LinkedList::nullData() {

Node \*aux;

while(head != NULL) {

aux = head;

head = head->getNext();

delete aux;

}

}

Song LinkedList::getData(Node\* pos) {

if(isEmpty() || pos == nullptr) {

throw NodeException("Insuficiencia de datos.");

}

else {

return pos->getData();

}

}

Node\* LinkedList::getFirst() {

return head;

}

Node\* LinkedList::getLast() {

if(isEmpty()) {

return nullptr;

}

Node \*aux = head;

while(aux->getNext() != nullptr) {

aux = aux->getNext();

}

return aux;

}

Node\* LinkedList::getPrev(Node\* pos) {

if(isEmpty() || pos == nullptr) {

return nullptr;

}

return pos->getPrev();

}

Node\* LinkedList::getNext(Node\* pos) {

if(isEmpty() || pos == nullptr) {

return nullptr;

}

return pos->getNext();

}

Node\* LinkedList::locateNode(Song sng) {

Node \*aux = head;

while(aux != nullptr && aux->getData() != sng) {

aux = aux->getNext();

}

return aux;

}

bool LinkedList::isEmpty() {

return (head==nullptr);

}

void LinkedList::printList() {

Node \*aux = head;

while(aux != nullptr) {

aux->printNode();

aux = aux->getNext();

}

}

Song LinkedList::createSong(const string& t, const string& a, const string& rl, const string& g ) {

Song temp;

temp.setTitle(t);

temp.setAuthor(a);

temp.setRecordL(rl);

temp.setGenre(g);

temp.setMP3(a,t);

return temp;

}

Node\* LinkedList::locateByTitle(string s) {

Node \*aux = head;

while(aux != nullptr && aux->getData().getTitle() != s) {

aux = aux->getNext();

}

return aux;

}

//Node.h

#ifndef NODE\_H\_INCLUDED

#define NODE\_H\_INCLUDED

#include "nodeException.h"

#include "song.h"

/\* TIPO SONG \*/

class Node {

private:

Song data;

Node \*nextAddr;

Node \*prevAddr;

public:

Node();

Node(Song, Node\*, Node\*);

void setData(Song &e);

void setNext(Node\*);

void setPrev(Node\*);

Song& getData();

Node \*getNext();

Node \*getPrev();

void printNode();

};

#endif // NODE\_H\_INCLUDED

//Node.cpp

#include "node.h"

/\* Constructores \*/

Node::Node() {

nextAddr = nullptr;

prevAddr = nullptr;

}

Node::Node(Song data, Node\* next, Node\* prev ) {

this->data = data;

this->nextAddr = next;

this->prevAddr = prev;

}

/\* Setters \*/

void Node::setData(Song &data) {

this->data = data;

}

void Node::setNext(Node \*next) {

this->nextAddr = next;

}

void Node::setPrev(Node \*prev) {

this->prevAddr = prev;

}

/\* Getters \*/

Song& Node::getData() {

return data;

}

Node\* Node::getNext() {

return nextAddr;

}

Node \*Node::getPrev() {

return prevAddr;

}

/\* Impresiones \*/

void Node::printNode() {

cout<< "\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_"<<endl;

cout<< "Direccion: " << this << " -> Siguiente: " << nextAddr << endl;

data.printSong();

}

//Song.h

#ifndef SONG\_H\_INCLUDED

#define SONG\_H\_INCLUDED

#include <iostream>

#include <string>

using namespace std;

class Song {

private:

string title;

string author;

string recordL;

string genre;

string mp3;

public:

Song();

Song(const string&,const string&,const string&,const string&);

string getMP3();

string getTitle();

string getAuthor();

string getRecordL();

string getGenre();

string getSongInfo();

void setTitle(const string&);

void setAuthor(const string&);

void setRecordL(const string&);

void setGenre(const string&);

void setMP3(const string&, const string&);

void printSong();

bool operator == (Song&);

bool operator != (Song&);

bool operator < (Song&);

bool operator > (Song&);

bool operator = (Song&);

};

#endif // SONG\_H\_INCLUDED

//song.cpp

#include "song.h"

/\* Constructores \*/

Song::Song() {

title = "Def Title";

author = "Def Author";

recordL = "Def RecordL";

genre = "Def Genre";

mp3 = "Def mp3";

}

Song::Song(const string &t, const string &a, const string &rl, const string &g) {

title = t;

author = a;

recordL = rl;

genre = g;

setMP3(a,t);

}

/\* Getters \*/

string Song::getTitle() {

return title;

}

string Song::getAuthor() {

return author;

}

string Song::getRecordL() {

return recordL;

}

string Song::getGenre() {

return genre;

}

string Song::getMP3() {

return mp3;

}

string Song::getSongInfo () {

return author + ": " + title + ", " + recordL + ", " + genre + " |" + mp3 + "\n";

}

/\* Setters \*/

void Song::setTitle(const string& t) {

title = t;

}

void Song::setAuthor(const string& a) {

author = a;

}

void Song::setRecordL(const string& rl) {

recordL = rl;

}

void Song::setGenre(const string& g) {

genre = g;

}

void Song::setMP3(const string& a, const string& t) {

mp3 = t +" - "+ a + ".mp3";

}

void Song::printSong() {

cout<< getSongInfo();

}

/\* Operadores \*/

bool Song::operator == (Song &s) {

if(getTitle() == s.getTitle() && getAuthor() == s.getAuthor() ) {

return true;

}

return false;

}

bool Song::operator != (Song &s) {

if( (getTitle() != s.getTitle()) || (getAuthor() != s.getAuthor()) || (getMP3() != s.getMP3()) || (getGenre() != s.getGenre()) ) {

return true;

}

else {

return false;

}

}

bool Song::operator < (Song &s) {

if((getAuthor() == s.getAuthor() ) && (getTitle() < s.getTitle() ) ) {

return true;

}

if (getAuthor() < s.getAuthor() ) {

return true;

}

return false;

}

bool Song::operator > (Song &s) {

if( (getAuthor() == s.getAuthor()) && (getTitle() > s.getTitle()) ) {

return true;

}

if (getAuthor() > s.getAuthor() ) {

return true;

}

return false;

}

bool Song::operator = (Song &s) {

title = s.getTitle();

author = s.getAuthor();

recordL = s.getRecordL();

genre = s.getGenre();

mp3 = s.getMP3();

}