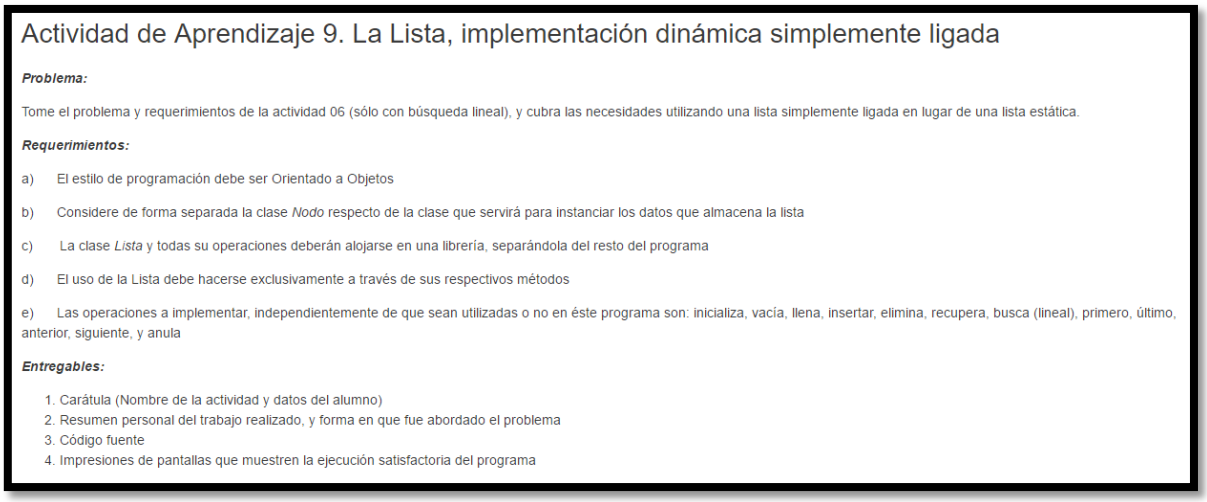
CENTRO UNIVERSITARIO DE CIENCIAS EXÁCTAS E INGENIERÍAS

Lista Ligada Simple

Tarea 09.1

Aldo Alexandro Vargas Meza 213495653

23/03/2017



Actividad de Aprendizaje 9. La Lista, implementación dinámica simplemente ligada

Problema:

Tome el problema y requerimientos de la actividad 06 (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 su 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

Entregables:

1. Carátula (Nombre de la actividad y datos del alumno)
2. Resumen personal del trabajo realizado, y forma en que fue abordado el problema
3. Código fuente
4. Impresiones de pantallas que muestren la ejecución satisfactoria del programa

En las tareas pasadas, se implementó una lista estática con un límite de datos, y con diferentes funciones. La actividad a continuación recopila las generalidades de los programas resultantes de tareas pasadas y crea un nuevo programa con diferentes cambios.
Los principales, son el funcionamiento a través de punteros y sin utilizar un registro.

En la última actividad, en los punteros, ya fue posible implementar la lista con todos sus métodos, de manera dinámica.

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 una dirección hacia otro elemento. 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.

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 simple ligada.

El menú principal:

```
"C:\Users\Aldo\Google Drive\CUCEI 8vo\Estructura de datos\Tareas\09.2 Listas Linkeadas\Linked\bin\Debug\Linked.exe"
-- Base de Datos de Canciones --

1. Ingresar Nueva Cancion
2. Borrar Cancion
3. Ordenar Lista
4. Visualizar lista
5. Busqueda Lineal de Archivo MP3
6. Salida
```

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.

```
"C:\Users\Aldo\Google Drive\CUCEI 8vo\Estructura de datos\Tareas\09.2 Listas Linkeadas\Linked\bin\Debug\Linked.exe"
Titulo:   Winds of Change
Autor:    Scorpions
Disquera: RockLo
Genero:

Ingrese el Genero:Rock
```

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.

```
Direccion: 0xa525f0 -> Siguiente: 0xa52540
Gorillaz: Stylo, Fin, Hip Hop |Stylo - Gorillaz.mp3
_____________________________
Direccion: 0xa52540 -> Siguiente: 0xa52490
Rebeca Black: Friday, Partay, Pop |Friday - Rebeca Black.mp3
_____________________________
Direccion: 0xa52490 -> Siguiente: 0x1e6ea0
Sublime: Santeria, Sublime, Punk |Santeria - Sublime.mp3
_____________________________
```

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

```
"C:\Users\Aldo\Google Drive\CUCEI 8vo\Estructura de datos\Tareas\09.2 Listas Linkeadas\Linked\bin\Debug\Linked.exe"
2. Borrar Cancion

Ingrese el nombre de la cancion:
Friday
Rebeca Black: Friday, Partay, Pop |Friday - Rebeca Black.mp3
Press any key to continue . . .
```

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

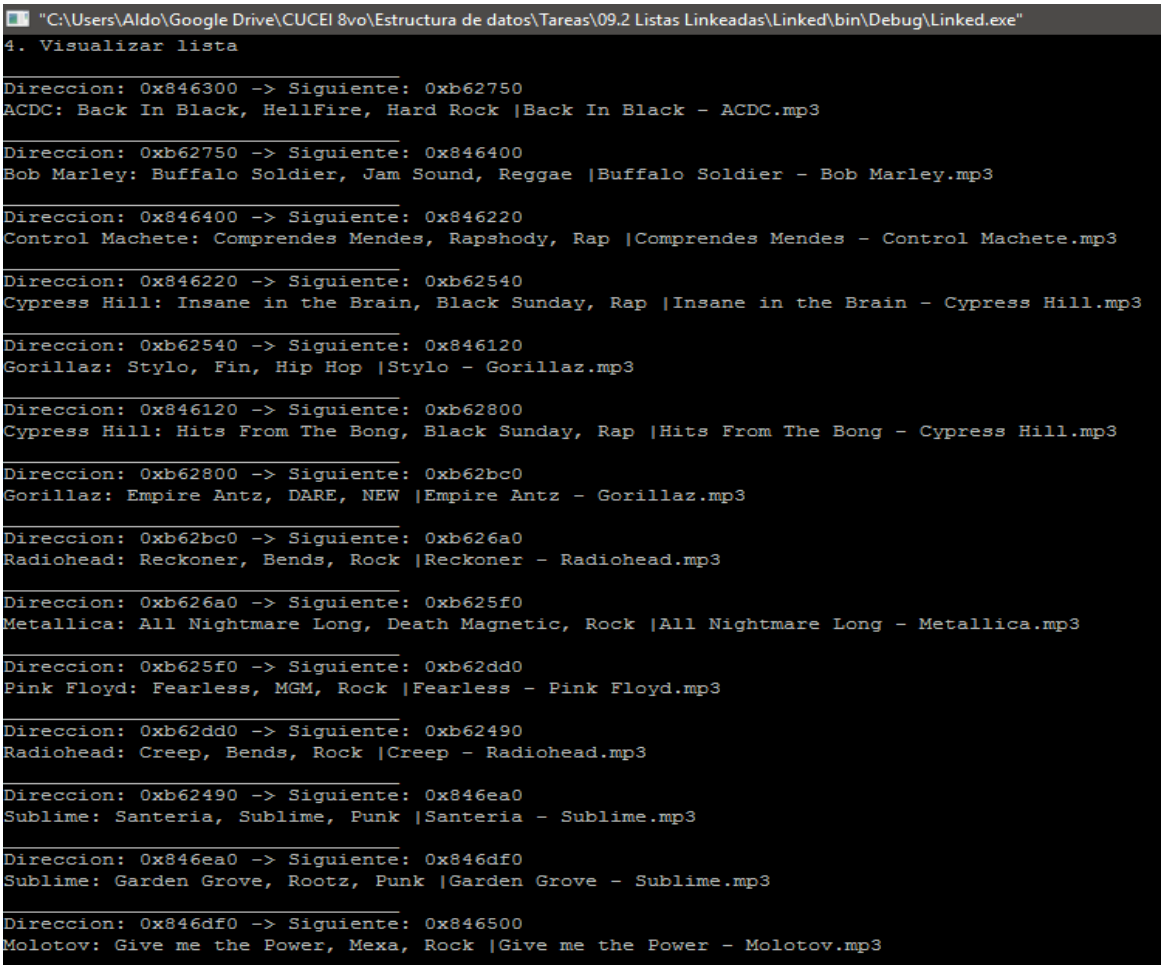
```
_____________________________
Direccion: 0xa525f0 -> Siguiente: 0xa52490
Gorillaz: Stylo, Fin, Hip Hop |Stylo - Gorillaz.mp3
_____________________________
Direccion: 0xa52490 -> Siguiente: 0x1e6ea0
Sublime: Santeria, Sublime, Punk |Santeria - Sublime.mp3
_____________________________
Direccion: 0x1e6ea0 -> Siguiente: 0x1e6df0
Sublime: Garden Grove, Rootz, Punk |Garden Grove - Sublime.mp3
_____________________________
```

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

```
"C:\Users\Aldo\Google Drive\CUCEI 8vo\Estructura de datos\Tareas\09.2 Listas Linkeadas\Linked\bin\Debug\Linked.exe"
3. Ordenar Lista
HECHO.
Press any key to continue . . .
```

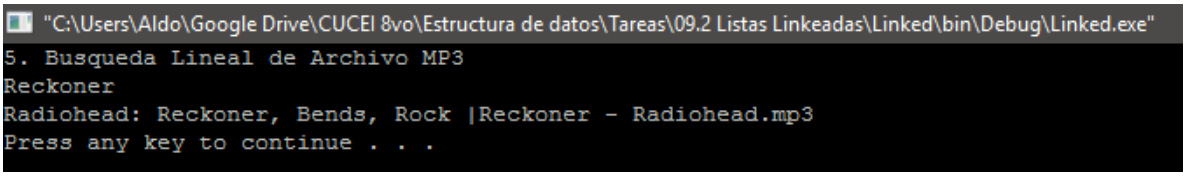
La impresión sin ordenar y ordenada:

```
"C:\Users\Aldo\Google Drive\CUCEI 8vo\Estructura de datos\Tareas\09.2 Listas Linkeadas\Linked\bin\Debug\Linked.exe"
4. Visualizar lista
_____________________________
Direccion: 0xb62b10 -> Siguiente: 0xb62f30
Scorpions: Winds of Change, RockLo, Rock |Winds of Change - Scorpions.mp3
_____________________________
Direccion: 0xb62f30 -> Siguiente: 0xb63350
Scorpions: Winds of Change, RockLo, Rock |Winds of Change - Scorpions.mp3
_____________________________
Direccion: 0xb63350 -> Siguiente: 0xb62e80
Scorpions: Winds of Change, RockLo, Rock |Winds of Change - Scorpions.mp3
_____________________________
Direccion: 0xb62e80 -> Siguiente: 0xb62dd0
Sublime: Doin Time, Rootz, Punk |Doin Time - Sublime.mp3
_____________________________
Direccion: 0xb62dd0 -> Siguiente: 0xb62bc0
Radiohead: Creep, Bends, Rock |Creep - Radiohead.mp3
_____________________________
Direccion: 0xb62bc0 -> Siguiente: 0xb62800
Radiohead: Reckoner, Bends, Rock |Reckoner - Radiohead.mp3
_____________________________
Direccion: 0xb62800 -> Siguiente: 0xb62750
Gorillaz: Empire Antz, DARE, NEW |Empire Antz - Gorillaz.mp3
_____________________________
Direccion: 0xb62750 -> Siguiente: 0xb626a0
Bob Marley: Buffalo Soldier, Jam Sound, Reggae |Buffalo Soldier - Bob Marley.mp3
_____________________________
Direccion: 0xb626a0 -> Siguiente: 0xb625f0
Metallica: All Nightmare Long, Death Magnetic, Rock |All Nightmare Long - Metallica.mp3
_____________________________
Direccion: 0xb625f0 -> Siguiente: 0xb62540
Pink Floyd: Fearless, MGM, Rock |Fearless - Pink Floyd.mp3
_____________________________
Direccion: 0xb62540 -> Siguiente: 0xb62490
Gorillaz: Stylo, Fin, Hip Hop |Stylo - Gorillaz.mp3
_____________________________
Direccion: 0xb62490 -> Siguiente: 0x846ea0
Sublime: Santeria, Sublime, Punk |Santeria - Sublime.mp3
_____________________________
Direccion: 0x846ea0 -> Siguiente: 0x846df0
Sublime: Garden Grove, Rootz, Punk |Garden Grove - Sublime.mp3
_____________________________
Direccion: 0x846df0 -> Siguiente: 0x846500
Molotov: Give me the Power, Mexa, Rock |Give me the Power - Molotov.mp3
```



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.



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 \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->setNext(head);
        head = aux;
        }
    else {
        aux->setNext(pos->getNext());
        pos->setNext(aux);
        }
```

```
    }

void LinkedList::deleteNode(Node* pos) {
    if(isEmpty() || pos == nullptr) {
        return;
        }
    if(pos == head) {
        head = head->getNext();
        }
    else {
        getPrev(pos)->setNext(pos->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);
    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 = 0;

    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;
        }
    Node *aux = head;

    while(aux != nullptr && aux->getNext() != pos) {
        aux = aux->getNext();
        }
    return aux;
    }

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;

    public:

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

        void setData(Song &e);
        void setNext(Node*);
        void setPrev(Node*);

        Song&  getData();
        Node *getNext();

        void printNode();
    };


#endif // NODE_H_INCLUDED
```

```cpp
//node.cpp

#include "node.h"

/*  Constructores  */
Node::Node() {
    nextAddr = nullptr;
    }

Node::Node(Song data, Node* next, Node* prev ) {
    this->data      = data;
    this->nextAddr  = next;
    }


/*  Setters  */
void Node::setData(Song &data) {
    this->data = data;
    }

void Node::setNext(Node *next) {
    this->nextAddr = next;
    }

/*  Getters  */
Song& Node::getData() {
    return data;

    }

Node* Node::getNext() {
    return nextAddr;
    }

/*  Impresiones  */
void Node::printNode() {
    cout<< "_____________________________"<<endl;
    cout<< "Direccion: " << this << " -> Siguiente: " << nextAddr << endl;
    data.printSong();

    }
```

```
// song.h

//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();
    }
```