

# Métodos de Búsqueda

---

## Tarea 05

Aldo Alexandro Vargas Meza 213495653  
23/02/2017



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

**Problema:**

Reutilice el resultado de la actividad 03, y adapte 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.

Para llevar a cabo el programa se reutilizó y modificó el programa de la actividad 3.

A la clase song se le modificó un nuevo atributo, el cual por medio del nombre del autor y la canción se genera un archivo mp3.

A la lista ranking, se le codificó la búsqueda lineal y binaria para por medio del autor o el título de la canción, la función regrese el nombre del archivo mp3.

Los archivos son los siguientes:

## **Clase Song**

### **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:

        void setTitle(const string&);
        void setAuthor(const string&);
        void setRecordL(const string&);
        void setGenre(const string&);
        void setMP3(const string&, const string&);

        string getMP3();
        string getTitle();
        string getAuthor();
        string getRecordL();
        string getGenre();

};

#endif // SONG_H_INCLUDED
```

Song.cpp

```
#include "song.h"
```

```
void Song::setTitle(const string& t) {  
    title = t;  
}  
  
void Song::setAuthor(const string& a) {  
    author = a;  
}  
  
void Song::setRecordL(const string& r1) {  
    recordL = r1;  
}  
  
void Song::setGenre(const string& g) {  
    genre = g;  
}  
  
void Song::setMP3(const string& a, const string& t){  
    mp3 = t + " - " + a + ".mp3";  
}  
  
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;  
}
```

La clase song está implementada con las funciones getters y setters generales. Como atributos tiene 5 cadenas. Esta clase va a proporcionar el tipo de dato a la lista del ranking.

## Clase Ranking

### Ranking.h

```
#ifndef RANKING_H_INCLUDED
#define RANKING_H_INCLUDED

#include <iomanip>
#include <iostream>
#include "string.h"

#include "song.h"
#include "rankingException.h"

class rankingList {
private:
    Song newSong[300];
    int last;

public:
    Song createSong(const string&, const string&, const string&, const
string&, const int&);

    void initialize();
    bool isEmpty();
    bool isFull();
    void deleteAll();

    void insertSong(const int&, const Song&);
    void deleteSong(const int&);

    int getFirstPos();
    int getLastPos();
    int prevPos(const int&);
    int nextPos(const int&);
    Song getSong(const int&);

    void printSong(const int&);

    string linealSearch(const string&);

};

#endif // RANKING_H_INCLUDED

Ranking.cpp

#include "ranking.h"

Song rankingList::createSong(const string& t, const string& a, const string& r1,
const string& g, const int& p ) {
    Song temp;
```

```

        temp.setTitle(t);
        temp.setAuthor(a);
        temp.setRecordL(r1);
        temp.setGenre(g);
        temp.setMP3(a,t);

        return temp;
    }

void rankingList::initialize() {
    last = -1;
}

bool rankingList::isEmpty() {
    return last == -1;
}

bool rankingList::isFull() {
    return last == 49;
}

void rankingList::deleteAll() {
    last = -1;
}

void rankingList::insertSong(const int& pos, const Song& s) {
    if(isFull()) {
        throw ListException("Desbordamiento de datos.");
    }
    else if(pos < -1 || (pos > last)) {
        throw ListException("Posicion invalida");
    }

    int i = last;

    while(i > pos) {
        newSong[i+1] = newSong[i];
        i++;
    }

    newSong[pos+1] = s;
    last++;
}

void rankingList::deleteSong(const int& pos) {
    if(isFull()) {
        throw ListException("Insuficiencia de datos.");
    }
    else if(pos < -1 || (pos > last)) {
        throw ListException("Posicion invalida");
    }

    int i = pos;

    while(i < last) {
        newSong[i] = newSong[i+1];
        i++;
    }
    last--;
}

int rankingList::getFirstPos() {
    if(isEmpty()) {
        return -1;
    }
    return 0;
}

```

```

    }

int rankingList::getLastPos() {
    return last;
}

int rankingList::prevPos(const int& pos) {
    if(isEmpty() || pos<1 || pos>last) {
        return -1;
    }
    return pos-1;
}

int rankingList::nextPos(const int& pos) {
    if(isEmpty() || pos<1 || pos>last) {
        return -1;
    }
    return pos+1;
}

Song rankingList::getSong(const int& pos) {
    if(isFull()) {
        throw ListException("Insuficiencia de datos.");
    }
    else if(pos< -1 || (pos>last)) {
        throw ListException("Posicion invalida");
    }
    return newSong[pos];
}

void rankingList::printSong(const int& pos) {
    cout <<setw(3)<<"["<<pos+1<<"]" << setw(25)<<newSong[pos].getTitle()
        << setw(25) <<newSong[pos].getAuthor()
        << setw(25) <<newSong[pos].getRecordL()
        << setw(25) <<newSong[pos].getGenre()
        << setw(25) <<newSong[pos].getMP3()
        <<endl;
}

string rankingList::linealSearch(const string& s){
    int i = 0;

    while(i<=last){
        while( (s == newSong[i].getAuthor()) || (s == newSong[i].getTitle()) ){
            return newSong[i].getMP3();
        }
        i++;
    }
    return "No Encontrado Lineal";
}

```

Para la clase lista, las funciones de lista necesarias fueron implementadas eficientemente.

La lista además de sus funciones estándar, cuenta con una función propia hecha para crear el objeto del tipo song antes de insertarlo a la lista.

La lista maneja excepciones para el desbordamiento de datos y las posiciones invalidas posibles.

```
listException.h

#ifndef RANKINGEXCEPTION_H_INCLUDED
#define RANKINGEXCEPTION_H_INCLUDED

#include <exception>
#include <string>

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();
    }
};

#endif // RANKINGEXCEPTION_H_INCLUDED
```

Por ultimo en el archivo main tenemos la implementación del menú, en donde un ciclo while con una máquina de estados hace la función de menú cíclico.

En varias partes de la impresión del menú fue necesario imprimir con una función, los valores actuales de la tabla.

```
#include <iostream>
#include <iomanip>
#include <fstream>
#include "ranking.h"

using namespace std;

int main() {
    rankingList l;
```

```

1.initialize();
int estado = 0;
while(estado != 5) {
    switch(estado) {
        case 0: {
            system("cls");
            int opc;

            cout<<"Discografia Completa\n\n";

            cout<<"1. Ingresar Nueva Cancion \n";
            cout<<"2. Borrar Cancion \n";
            cout<<"3. Visualizar lista \n";
            cout<<"4. Busqueda Lineal de Archivo MP3 \n";
            cout<<"5. Salida \n";
            cin >>opc;

            switch(opc) {
                case 1:
                    estado = 1;
                    break;
                case 2:
                    estado = 2;
                    break;
                case 3:
                    estado = 3;
                    break;
                case 4:
                    estado = 4;
                    break;
                case 5:
                    estado = 5;
                    break;
                case 6:
                    estado = 5;
                    break;

                default:
                    estado = 0;
                    break;
            }
        }
        break;
    case 1: {
        system("cls");
        char t[100] = {},a[100]= {},r[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: "<<r<<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(r1,100);
            break;
        case 3:
            cout<<"Ingrese el Genero:";
            cin.getline(g,100);
            break;
    }
}

l.insertSong(l.getLastPos(),
l.createSong(t,a,r1,g,l.getLastPos()));
estado = 0;
}

break;

case 2: {
    int opc;
    system("cls");

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

    for(int i=0; i<l.getLastPos()+1; i++) {
        l.printSong(i);
    }

    cout<<"\nIngrese posicion a borrar."<<endl;
    cin >> opc;

    l.deleteSong(opc-1);

    estado = 0;
}
break;

case 3: {
    system("cls");

    cout<<"3. Visualizar lista \n";

    cout <<setw(3)<<"Num"
        << setw(25) <<"Titulo"
        << setw(25) <<"Autor"
        << setw(25) <<"Disquera"
        << setw(25) <<"Genero"
        << setw(25) <<"File"
        <<endl;

    for(int i=0; i<l.getLastPos()+1; i++) {
        l.printSong(i);
    }

    estado = 0;
    system("pause");
}
break;

case 4:{
    system("cls");
    cout<<"4. Busqueda Lineal de Archivo MP3 \n";
    string temp;

    if(l.getLastPos()==-1) {

```

```

        cout<<"Lista Vacía."<<endl;
        system("pause");
        estado = 0;
    }
    else {
        cout<<"Ingrese el Autor para obtener el nombre del archivo.";

        cin >>temp;
        cout<<"Nombre del Archivo: "<<l.linea1search(temp)<<endl;
        system("pause");
        estado = 0;
    }
}
break;

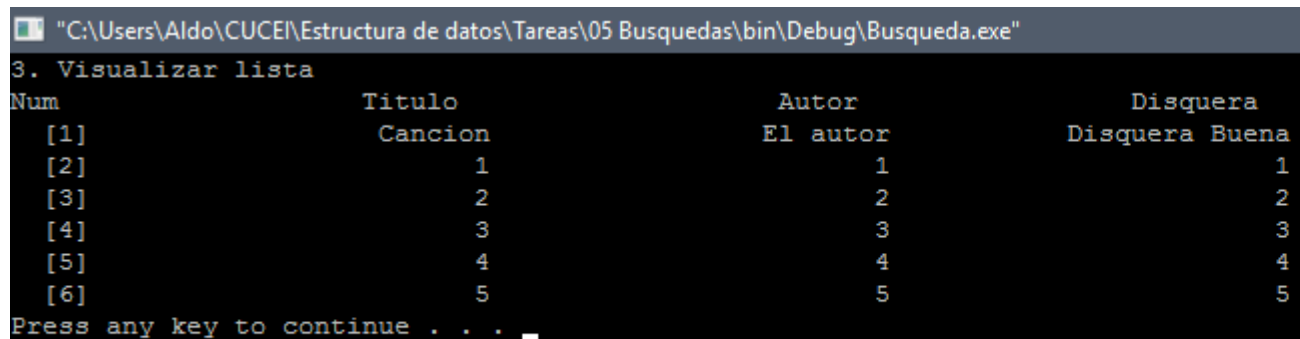
}

}

return 0;
}

```

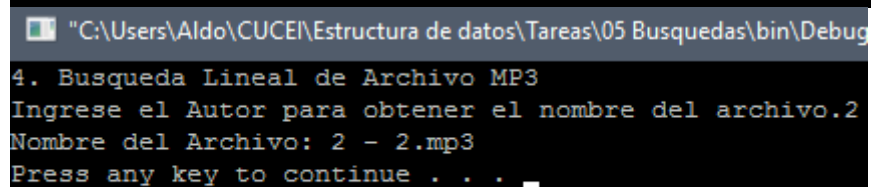
La ejecución del programa da como resultado las siguientes impresiones:



```

C:\Users\Aldo\CUCEI\Estructura de datos\Tareas\05 Busquedas\bin\Debug\Busqueda.exe
3. Visualizar lista
Num          Titulo          Autor          Disquera
[1]          Cancion          El autor      Disquera Buena
[2]              1              1              1
[3]              2              2              2
[4]              3              3              3
[5]              4              4              4
[6]              5              5              5
Press any key to continue . . .

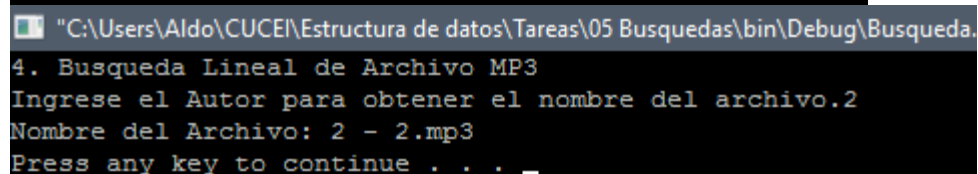
```



```

C:\Users\Aldo\CUCEI\Estructura de datos\Tareas\05 Busquedas\bin\Debug\Busqueda.exe
4. Busqueda Lineal de Archivo MP3
Ingrese el Autor para obtener el nombre del archivo.2
Nombre del Archivo: 2 - 2.mp3
Press any key to continue . . .

```



```

C:\Users\Aldo\CUCEI\Estructura de datos\Tareas\05 Busquedas\bin\Debug\Busqueda.exe
4. Busqueda Lineal de Archivo MP3
Ingrese el Autor para obtener el nombre del archivo.2
Nombre del Archivo: 2 - 2.mp3
Press any key to continue . . .

```

"C:\Users\Aldo\CUCE\Estructura de datos\Tareas\05 Busquedas\bin\Debug\Bus

Titulo: 6

Autor: 6

Disquera: 6

Genero:

Ingrese el Genero:

"C:\Users\Aldo\CUCE\Estructura de datos\Tareas\05 Busquedas\bin\Debug\Bus

4. Busqueda Lineal de Archivo MP3

Ingrese el Autor para obtener el nombre del archivo. \_

"C:\Users\Aldo\CUCE\Estructura de datos\Tareas\05 Busquedas\bin\

Discografia Completa

1. Ingresar Nueva Cancion

2. Borrar Cancion

3. Visualizar lista

4. Busqueda Lineal de Archivo MP3

5. Salida

2. Borrar Cancion

[1]	1	1	1
-----	---	---	---

1 - 1.mp3			
-----------	--	--	--

[2]	2	2	2
-----	---	---	---

2 - 2.mp3			
-----------	--	--	--

[3]	3	3	3
-----	---	---	---

3 - 3.mp3			
-----------	--	--	--

[4]	4	4	4
-----	---	---	---

4 - 4.mp3			
-----------	--	--	--

[5]	5	5	5
-----	---	---	---

5 - 5.mp3			
-----------	--	--	--

[6]	6	6	6
-----	---	---	---

6 - 6.mp3			
-----------	--	--	--

Ingrese posicion a borrar.

2 \_