Centro Universitario de Ciencias Exactas e Ingenierías (CUCEI)

# Actividad de aprendizaje 3:
# Lista, implementación estática

Víctor Agustín Díaz Méndez
Ingeniería en Informática

Estructura de Datos I (Sección D12)
Profesor: Dr. Gutierrez Hernandez Alfredo

# Problema planteado

Una radiodifusora necesita publicar su lista de éxitos de la 50 canciones más escuchadas, y le pasa los datos a su webmaster, aún no saben en qué orden se publicará la lista, si en orden alfabético por nombre del autor, o del intérprete, por nombre de la canción, o posición en el ranking, por lo que será necesario utilizar una estructura de datos que permita un manejo aleatorio de los datos, es decir una lista.

Haga un programa que cubra el problema. Deberá mostrar en pantalla la lista todo el tiempo, permitiendo añadir nuevos elementos y así como eliminarlos, y mostrar los cambios al momento.

## Requerimientos:

a) El estilo de programación debe ser Orientado a Objetos.
b) Utilizará un arreglo para almacenar los datos.
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, primero, último, anterior, siguiente, y anula.

# Resumen

El problema a abordar es relativamente sencillo teniendo en cuenta que se entendió y realizo correctamente la demostración en clase sobre listas. Para crear la colección de canciones del ranking, solo fue necesario partir de la lista mostrada en clase haciendo las debidas modificaciones:

- El tipo de dato de la lista debería de ser del registro “Song”. Mas adelante hablare sobre los atributos de este registro.
- Tomando en cuenta que el tipo de dato esta basado en un registro, el método buscar debería de funcionar de forma diferente. Debe comparar el atributo “name” con un string que recibiría como parámetro. La búsqueda seria por le nombre de la canción.
- Editar el método “printData” para mostrar los atributos de las canciones.

Respecto a al registro “Song” tiene una estructura muy sencilla para guardar su posición en el ranking, nombre, autor e interprete:

```
struct Song {

public:

    int ranking;

    string name;

    string author;

    string player;

};
```

La clase “Menu” tiene una función muy sencilla, ser la interfaz con el usuario. En ella se pueden insertar, eliminar canciones, y también visualizarlas. En todas las vistas se visualizan las canciones del top antes que los menús y formularios. De esta manera siempre se puede visualizar el top. Para mostrar ranking utilicé el método “printData” de la clase “songsList”.

Los metodos de esta clase son:

```
    void showMenu();

    void addSong();

    void deleteSong();

    void showSongs();
```

Aunque “showSongs” no fue requerido dentro del problema ni los requerimientos, decidí agregarlo como una opción extra ya que era muy sencillo de programar.

La función del método “showMenu” Es muy sencilla: mostrar el menu y no terminar el programa hasta que el usuario lo desee.

El método “addSongs” muestra un formulario con validación para que el usuario no ingrese valores que no sean validos, como por ejemplo una posición en el ranking fuera del 1 al 50 o nombres en blanco. Para almacenar la información en la lista utiliza los métodos de “songsList” para obtener la ultima posición(getLastPos) y

guardar la información(insertData). “insertData” recibe como parámetro la ultima posición de la lista y la canción a guardar. También vigila el caso de que se presente una excepción y no se guarde la información.

El método “deleteSong” permite eliminar una canción de la lista basándose en su nombre. Para esto le pregunta al nombre del usuario el nombre de la canción que desea eliminar, despues la busca su posición en la lista con el método “findData” del objeto “songsList” que nos devuelve como valor su posición o menos uno si no existe, este valor se pasa como parámetro al método “deleteData” de la misma clase que arrojara una excepción si el registro no se puede eliminar sea por insuficiencia de datos o una posición no valida.

## Main.cpp

```
#include "menu.h"


int main() {
	Menu menu;
	menu.showMenu();
	return 0;
}
```

## Menu.h

```
#ifndef MENU_H
#define MENU_H

#include "songsList.h"

class Menu {
public:
	void showMenu();
protected:
	SongsList songsList;
	void addSong();
	void deleteSong();
	void showSongs();
```

```
private:

};


#endif // MENU_H
```

## Menu.cpp

```
#include "menu.h"

#include <iostream>

#include <cstdlib>

#include "song.h"


#ifdef WIN32

#define CLEAR "cls"

#else

#define CLEAR "clear"

#endif


using namespace std;


void Menu::showMenu() {

    int option;

    songsList.initialize();

    do {
```

```
songsList.printData();

cout << "\tRANKING MUSICAL" << endl

        << "Menu:" << endl

        << "1.-Añadir cancion" << endl

        << "2.-Eliminar cancion" << endl

        << "3.-Mostrar ranking" << endl

        << "0.-Salir" << endl;

cin >> option;

cin.ignore();


switch(option) {

case 1:

      addSong();

      break;

case 2:

      deleteSong();

      break;

case 3:

      showSongs();

      break;

case 0:

      break;
```

```
            default:
                system(CLEAR);
                cout << "ERROR, opcion invalida" << endl;
                break;
            }
        } while(option!=0);
}

void Menu::addSong() {
        bool answerError;
        Song song;

        system(CLEAR);

        songsList.printData();
        cout << "\tAñadir cancion" << endl;

        do {
            cout << "Nombre: ";
            getline(cin, song.name);

            if(song.name != "") {
```

```
            answerError=false;
        } else {
            cout << "ERROR, ingresa el nombre de la canción" << endl;
            answerError=true;
        }
    } while(answerError);

    do {
        cout << "Autor: ";
        getline(cin, song.author);

        if(song.author != "") {
            answerError=false;
        } else {
            cout << "ERROR, ingresa el nombre del autor" << endl;
            answerError=true;
        }
    } while(answerError);

    do {
        cout << "Interprete: ";
        getline(cin, song.player);
```

```
        if(song.player != "") {
            answerError=false;
        } else {
            cout << "ERROR, ingresa el nombre del interprete" << endl;
            answerError=true;
        }
    } while(answerError);

    do {
        cout << "Puesto en el ranking: " << endl;
        cin >> song.ranking;
        if(song.ranking > 0 and song.ranking <= 50) {
            answerError=false;
        } else {
            cout << "ERROR, el ranking no es valido (1-50)" << endl;
            answerError=true;
        }
    } while(answerError);

    try {
        songsList.insertData(songsList.getLastPos(), song);
```

```
        } catch(SongsListException ex) {
            cout << "ERROR, no se pudo guardar" << endl;
        }

        system(CLEAR);
        cout << "La cancion se guardo exitosamente" << endl << endl;
}

void Menu::deleteSong() {
        string name;

        system(CLEAR);
        songsList.printData();

        cout << "Nombre de la cancion a eliminar" << endl;
        getline(cin, name);

        system(CLEAR);
        try {
            songsList.deleteData(songsList.findData(name));
            cout << "Cancion eliminada" << endl;
        } catch(SongsListException ex) {
```

```
            cout << "ERROR, no se pudo eliminar la cancion" << endl;
        }
        cout << endl << endl;
}

void Menu::showSongs() {
        system(CLEAR);
        songsList.printData();
        cout << "Presione \"Enter\" para continuar" << endl;
        cin.get();
}
```

## SongsList.h

```
#ifndef SONGSLIST_H
#define SONGSLIST_H

#include <exception>

#include "song.h"

#define ARRAY_SIZE 1024

class SongsListException : public std::exception {
```

```
private:
    std::string msg;
public:
    explicit SongsListException(const char* message) : msg(message) {}
    explicit SongsListException(const std::string& message) :msg(message) {}
    virtual ~SongsListException() throw () {}
    virtual const char* what() const throw () {
        return msg.c_str();
    }
};


class SongsList {
private:
    Song data[ARRAY_SIZE];
    int last;
    bool isValidPos(int);
public:
    void initialize();


    bool isEmpty();
    bool isFull();
```

```
        void insertData(int, const Song&);
        void deleteData(int);

        int getFirstPos();
        int getLastPos();
        int getPrevPos(int);
        int getNextPos(int);
        int findData(const string&);

        Song retrieve(int);

        void sortData();

        void printData();

        void deleteAll();
};

#endif // SONGSLIST_H
```

## SongsList.cpp

```
#include "songsList.h"
#include <iostream>
```

```
using namespace std;


bool SongsList::isValidPos(int p) {
	return p >= 0 and p <=last;
}


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


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


bool SongsList::isFull() {
	return last==ARRAY_SIZE-1;
}


void SongsList::insertData(int p, const Song& song) {
	if(isFull()) {
		throw SongsListException("The list is full, trying to insert");
```

```
    }
    if(p != -1 and !isValidPos(p)) {
        throw SongsListException("Invalid position, trying to insert");
    }
    for(int i = last; i > p; i--) {
        data[i + 1] = data[i];
    }
    data[p + 1] = song;
    last++;
}

void SongsList::deleteData(int p) {
    if(isEmpty()) {
        throw SongsListException("DATA UNDERRUN. The list is empty, trying to
delete");
    }
    if(!isValidPos(p)) {
        throw SongsListException("INVALID POSITION, trying ot delete");
    }
    for(int i = p; i < last; i++) {
        data[i] = data[i + 1];
    }
    last--;
```

```
}

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

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

int SongsList::getPrevPos(int p) {
	if(isEmpty() or !isValidPos(p) or p == 0) {
		return -1;
	}
	return p-1;
}

int SongsList::getNextPos(int p) {
	if(isEmpty() or !isValidPos(p) or p == last - 1) {
```

```
            return -1;
        }
        return p-1;
}

int SongsList::findData(const string& name) {
        for(int i = 0; i <= last; i++) {
            if(data[i].name == name) {
                return i;
            }
        }
        return -1;
}

Song SongsList::retrieve(int p) {
        if(isEmpty()) {
            throw SongsListException("DATA UNDERRUN, list empty, trying to get data");
        }
        if(!isValidPos(p)) {
            throw SongsListException("INVALID POSITION, trying to get data");
        }
        return data[p];
```

```
}

void SongsList::sortData() {

}

void SongsList::printData() {
	int i;
	cout << "Canciones: " << endl;
	for (i = 0; i <= last; i++) {
		cout << "Nombre " << data[i].name << endl
			<< "Autor " << data[i].author << endl
			<< "Interprete " << data[i].player << endl
			<< "Ranking " << data[i].ranking << endl << endl;
	}
	if(i==0) {
		cout << "No hay canciones en el ranking" << endl;
	}
	cout << endl;
}

void SongsList::deleteAll() {
	last=-1;
```

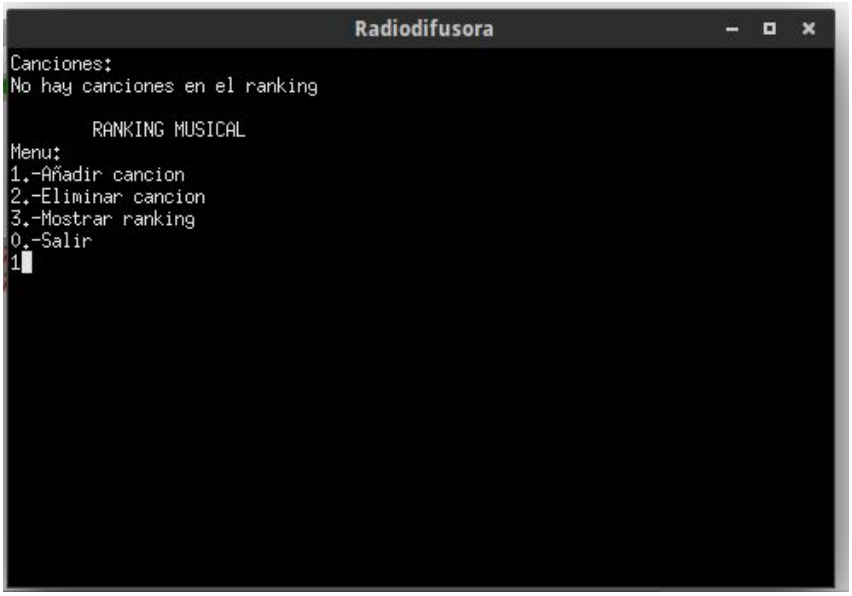
```
}
```

## Song.h

```
#ifndef SONG_H
#define SONG_H

#include <string>

using namespace std;

struct Song {
public:
    int ranking;
    string name;
    string author;
    string player;
};

#endif // SONG_H
```

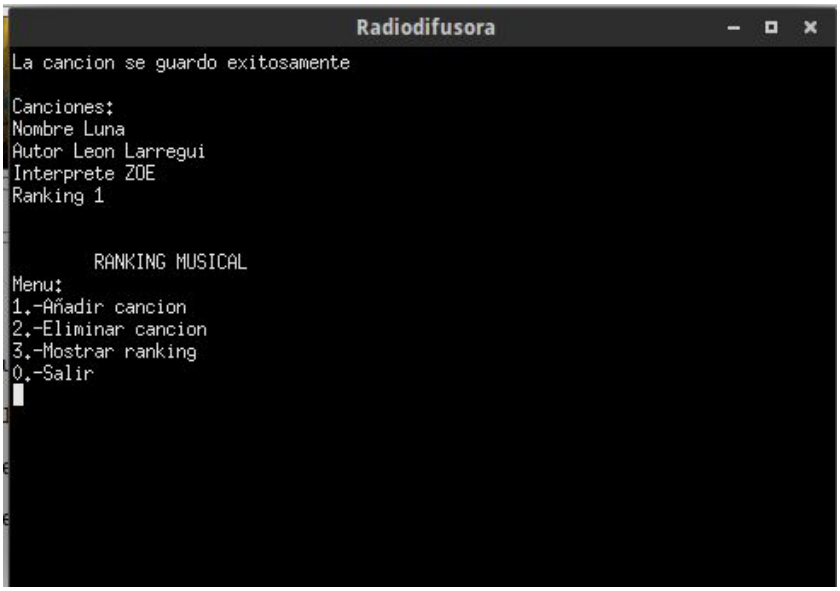
## Ejecución satisfactoria:

Se elige la opción “añadir canción”




Se llena el formulario.

Y aparece un mensaje que avisa que se guardo

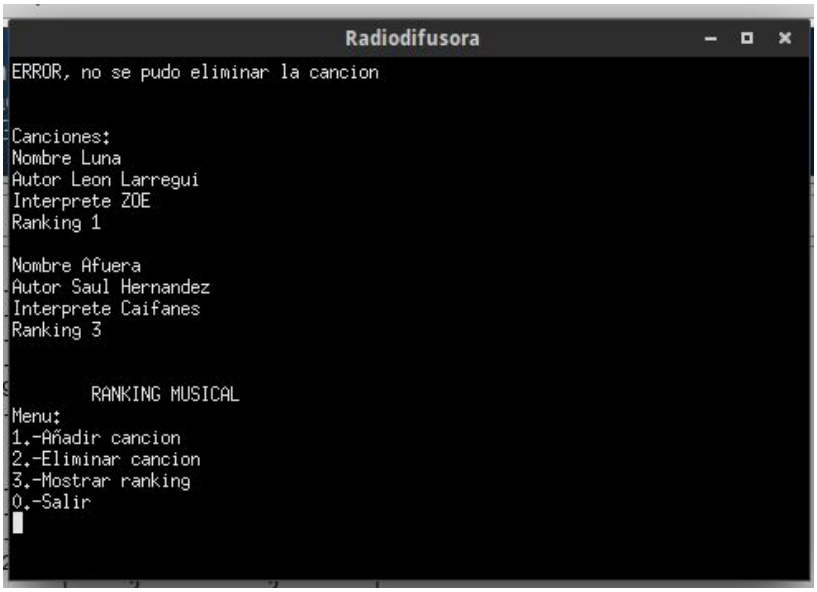



Se agregan otras dos canciones a la lista.

Para eliminar una canción solo se ingresa el nombre.


Aparece un mensaje que avisa que ya se elimino y muestra el listado de canciones donde ya no aparece la canción eliminada.

Mensaje de error de si se ingresa el nombre de una canción que no existe.




Método”showSongs” nos muestra el listado

Al ingresar 0 en el menu se termina la aplicación.