

La lista, Implementación estática

Tarea 03

Aldo Alexandro Vargas Meza 213495653

07/02/2017



“Reporte de actividad, que consiste en un menú cíclico con 2 opciones y una opción de salir, con tipo de programación orientado a objetos.”

Problema:

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

El programa necesita ser implementado, dado su comportamiento, en una lista. Esta lista, va a contener elementos del tipo canción, los cuales serán implementados en otra clase separada.

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;

public:
    void setTitle(const string&);
    void setAuthor(const string&);
    void setRecordL(const string&);
    void setGenre(const string&);

    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;
}

string Song::getTitle() {
    return title;
}

string Song::getAuthor() {
    return author;
}

string Song::getRecordL() {
    return recordL;
}

string Song::getGenre() {
    return genre;
}
```

La clase song está implementada con las funciones getters y setters generales. Como atributos tiene 4 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 "song.h"
#include "rankingException.h"

class rankingList {
private:
    Song newSong[50];
    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&);

};

```

```

#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);

```

```

        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(10)<<newSong[pos].getTitle()<<
    setw(10) <<newSong[pos].getAuthor()<< setw(10) <<newSong[pos].getRecordL()<<
    setw(10) <<newSong[pos].getGenre()<<endl;
}

```

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.

[illegible]

```

        cin >> r1;
        break;
    case 3:
        cout<<"Ingrese el Genero:";
        cin >>g;
        break;
    }

    }

    l.insertSong(l.getLastPos(),
l.createSong(t,a,r1,g,l.getLastPos()));

    estado = 0;
    }
    break;
    case 2: {
        int opc;
        system("cls");

        cout<<setw(30)<<"Ranking Canciones\n";
        for(int i=0; i<l.getLastPos()+1; i++) {
            l.printSong(i);
        }

        cout<<"Borrado de Cancion."<<endl;
        cout<<"Ingrese posicion a borrar."<<endl;
        cin >> opc;

        l.deleteSong(opc-1);

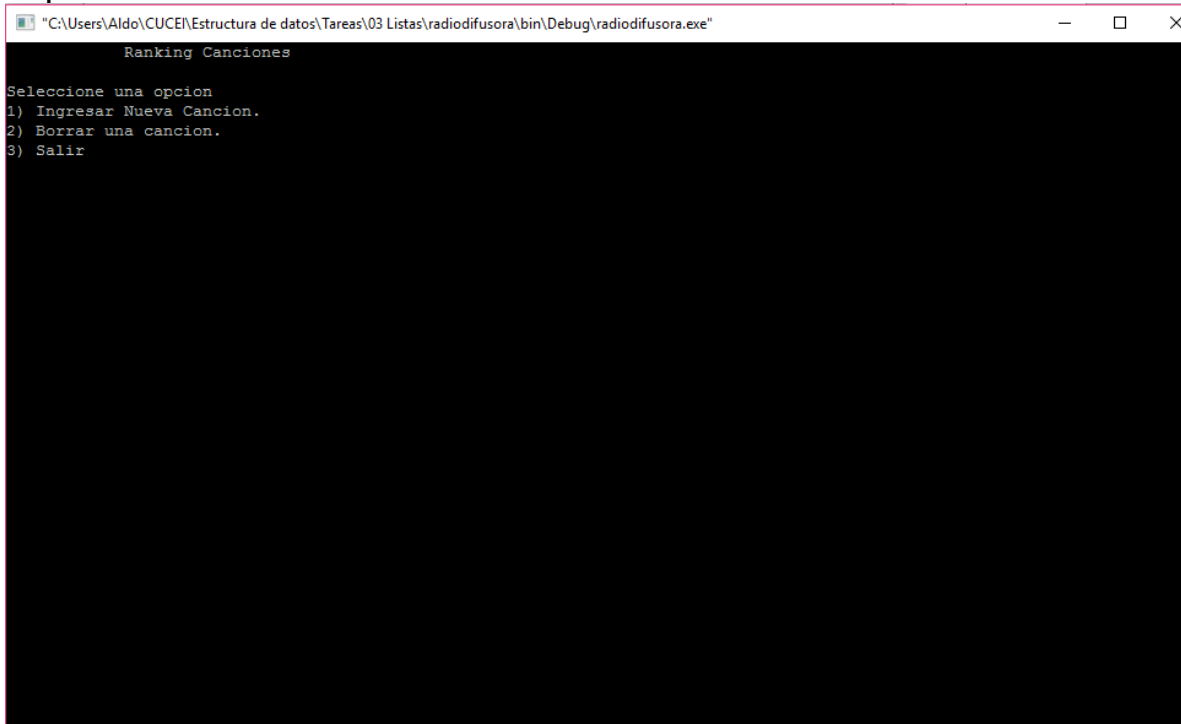
        estado = 0;
    }
    break;

    }

    }
    return 0;
}

```


La ejecución del programa da como resultado las siguientes impresiones:

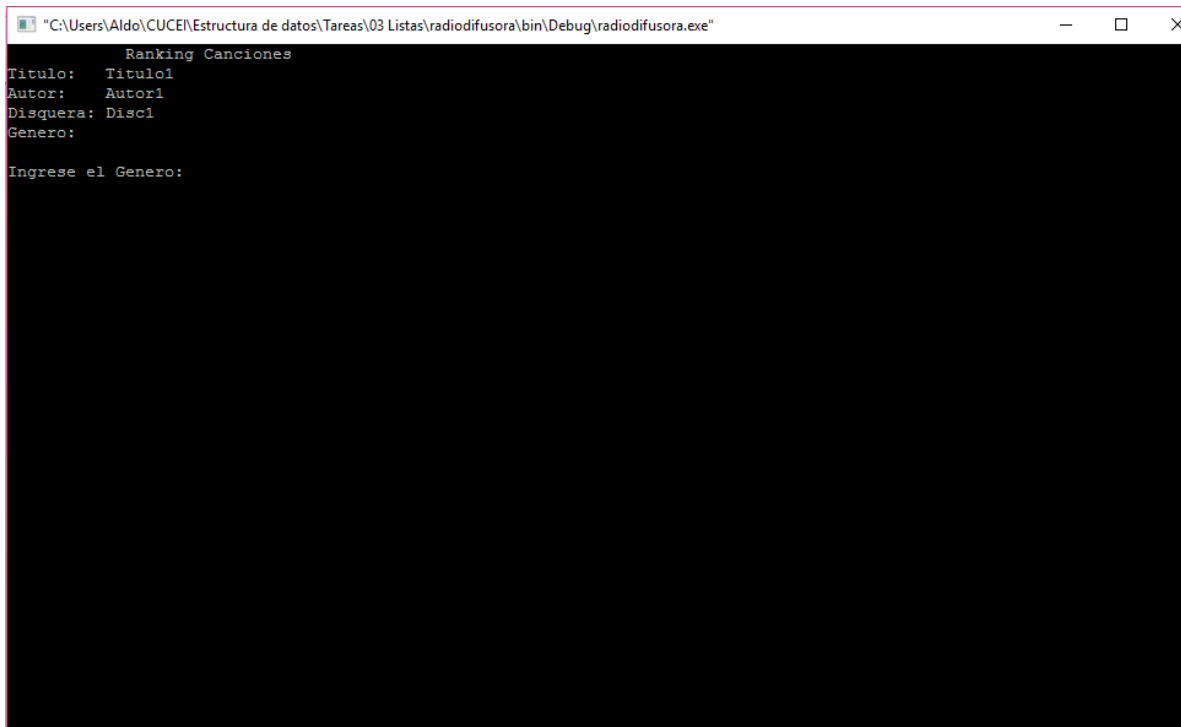


```
"C:\Users\Aldo\CUCE\Estructura de datos\Tareas\03 Listas\radiodifusora\bin\Debug\radiodifusora.exe"

Ranking Canciones

Seleccione una opcion
1) Ingresar Nueva Cancion.
2) Borrar una cancion.
3) Salir
```

Ingresando canciones:



```
"C:\Users\Aldo\CUCE\Estructura de datos\Tareas\03 Listas\radiodifusora\bin\Debug\radiodifusora.exe"

Ranking Canciones

Titulo: Titulo1
Autor: Autor1
Disquera: Discl
Genero:

Ingrese el Genero:
```

```
"C:\Users\Aldo\CUCE\Estructura de datos\Tareas\03 Listas\radiodifusora\bin\Debug\radiodifusora.exe"

Ranking Canciones
[1] Titulo1 Autor1 Disc1 12
[2] a a a a
[3] b b b b
[4] c c c c
[5] d d d d
[6] e e e e

Seleccione una opcion
1) Ingresar Nueva Cancion.
2) Borrar una cancion.
3) Salir
```

Eliminando canciones:

```
"C:\Users\Aldo\CUCE\Estructura de datos\Tareas\03 Listas\radiodifusora\bin\Debug\radiodifusora.exe"

Ranking Canciones
[1] Titulo1 Autor1 Disc1 12
[2] a a a a
[3] b b b b
[4] c c c c
[5] d d d d
[6] e e e e

Borrado de Cancion.
Ingrese posicion a borrar.
3
```

```
"C:\Users\Aldo\CUCE\Estructura de datos\Tareas\03 Listas\radiodifusora\bin\Debug\radiodifusora.exe"

Ranking Canciones
[1]  Titulo1  Autor1  Disc1  12
[2]      a      a      a      a
[3]      c      c      c      c
[4]      d      d      d      d
[5]      e      e      e      e

Seleccione una opcion
1) Ingresar Nueva Cancion.
2) Borrar una cancion.
3) Salir
```