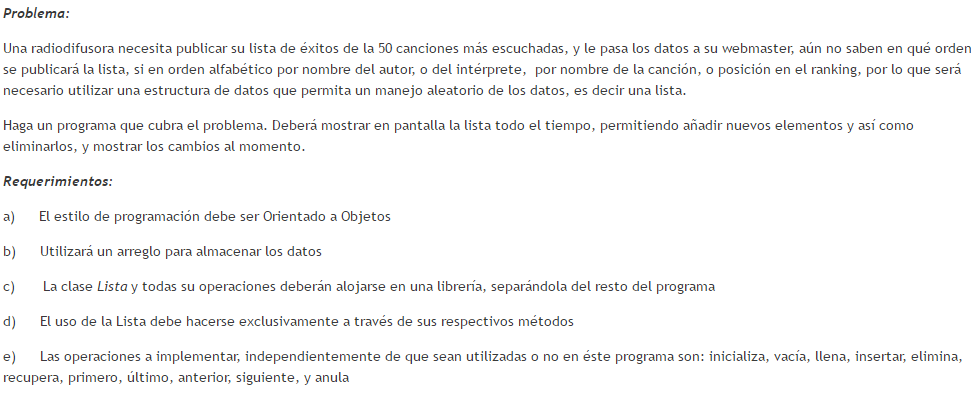
|  |
| --- |
| centro universitario de ciencias exáctas e ingenierías |
| La lista, Implementación estática |
| Tarea 03 |
|  |
| **Aldo Alexandro Vargas Meza 213495653** |
| **07/02/2017** |



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



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& rl) {**

**recordL = rl;**

**}**

**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& rl, const string& g, const int& p ) {**

**Song temp;**

**temp.setTitle(t);**

**temp.setAuthor(a);**

**temp.setRecordL(rl);**

**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.

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

**#include <iostream>**

**#include "ranking.h"**

**using namespace std;**

**int main() {**

**rankingList l;**

**l.initialize();**

**int estado = 0;**

**while(estado != 3) {**

**switch(estado) {**

**case 0: {**

**system("cls");**

**int opc;**

**/\*Impresion de la tabla\*/**

**cout<<setw(30)<<"Ranking Canciones\n";**

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

**l.printSong(i);**

**}**

**cout<<" \nSeleccione una opcion\n";**

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

**cout<<"2) Borrar una cancion.\n";**

**cout<<"3) Salir\n";**

**cin >>opc;**

**if(opc == 1) estado = 1;**

**if(opc == 2) estado = 2;**

**if(opc == 3) estado = 3;**

**}**

**break;**

**case 1: {**

**system("cls");**

**string t,a,rl,g;**

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

**system("cls");**

**cout<<setw(30)<<"Ranking Canciones\n";**

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

**l.printSong(i);**

**}**

**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 >> t;**

**break;**

**case 1:**

**cout<<"Ingrese el Autor:";**

**cin >> a;**

**break;**

**case 2:**

**cout<<"Ingrese la Disquera:";**

**cin >> rl;**

**break;**

**case 3:**

**cout<<"Ingrese el Genero:";**

**cin >>g;**

**break;**

**}**

**}**

**l.insertSong(l.getLastPos(), l.createSong(t,a,rl,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;**

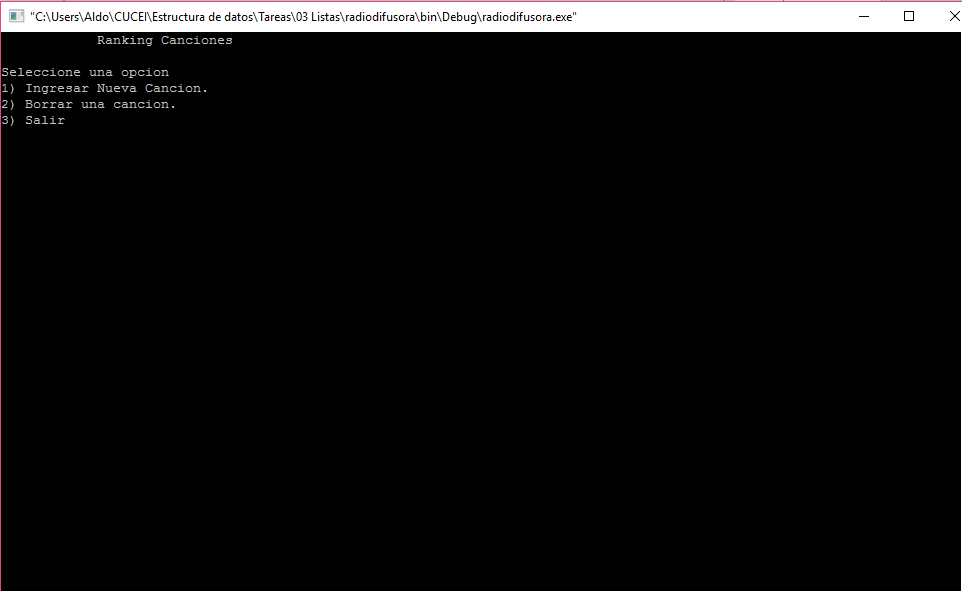
**}**

**}**

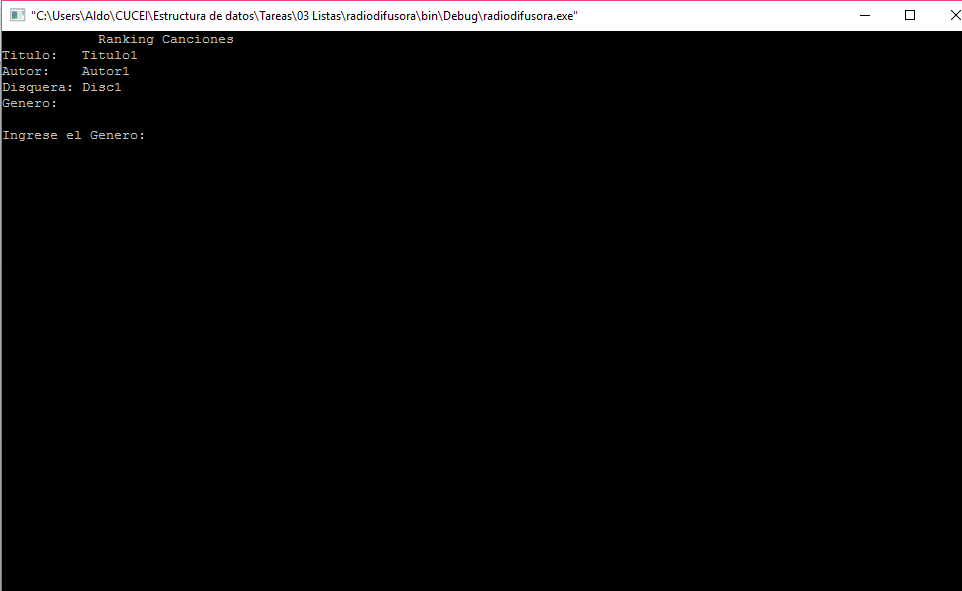
**return 0;**

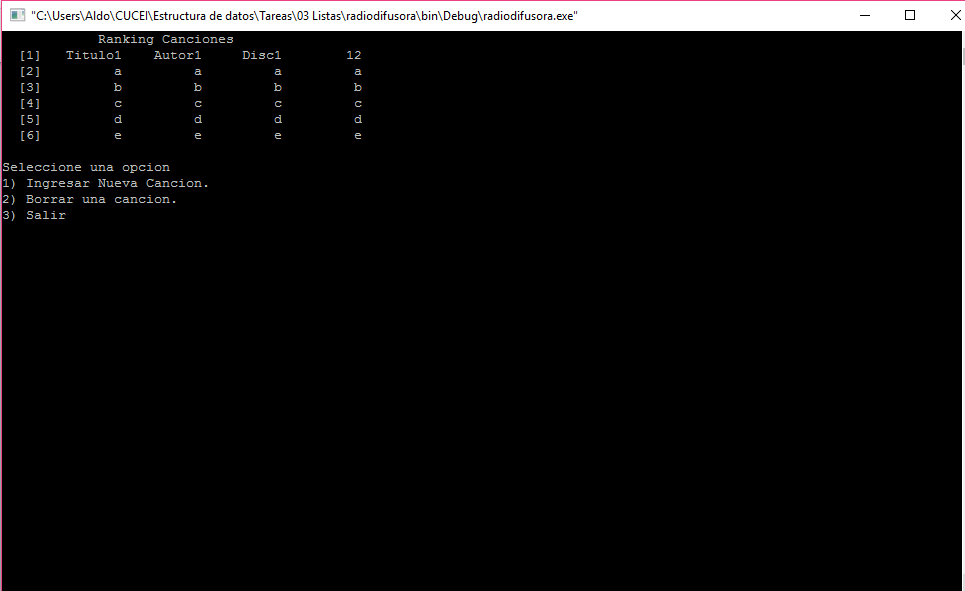
**}**

La ejecución del programa da como resultado las siguientes impresiones:



Ingresando canciones:





Eliminando canciones:

