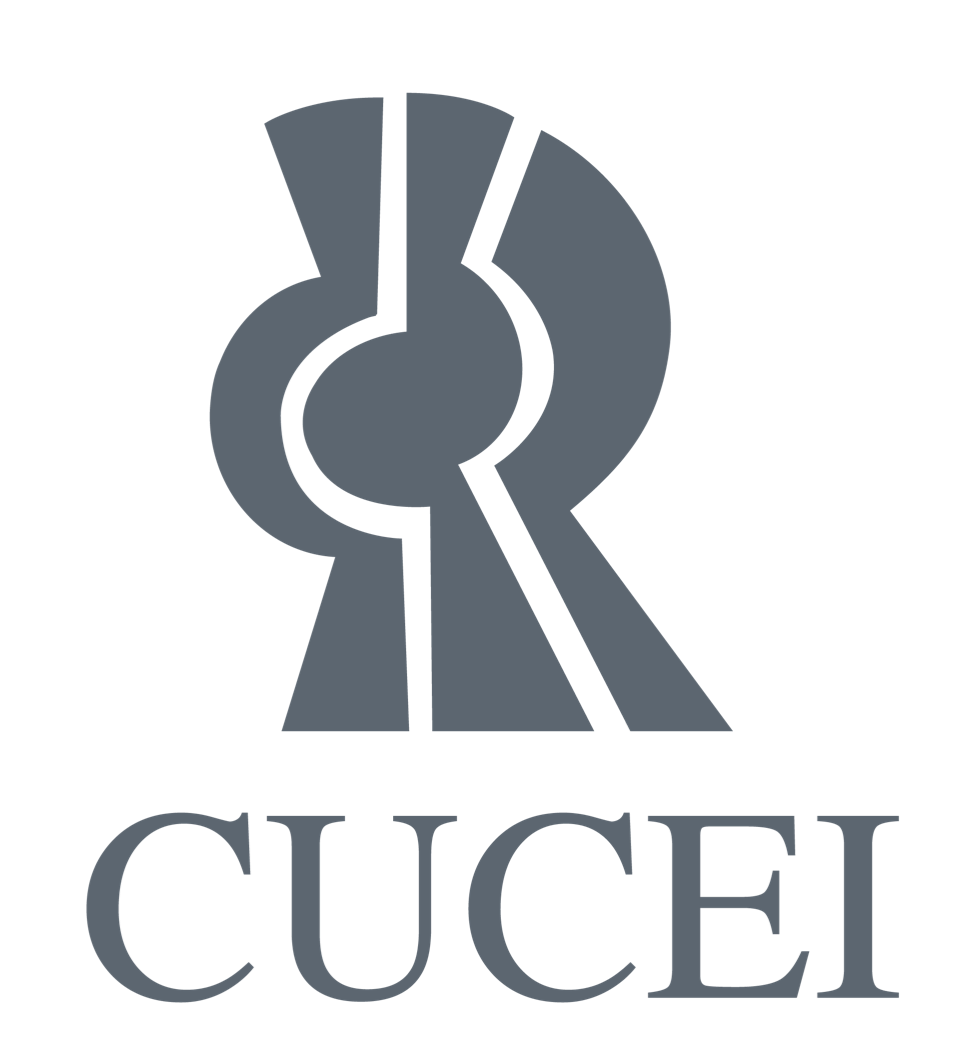
5-2-2019

**david gutierrez alvarez**

**Estructura de datos I**

****

## La Lista, implementación estática

|  |
| --- |
| **RESUMEN PERSONAL Y FORMA DE ABORDAR EL PROBLEMA** |

Esta actividad solo tenía que completarse con un archivo ya creado anteriormente, esto le agrego facilidad y dificultad, la facilidad fue que solo tenia que añadir pocas funciones, y la dificultad fue que tenia que adaptar esas funciones a lo que ya tenia creado, para evitar que el programa generara errores.

Abordad el problema fue sencillo, solo tenia que añadir dos funciones mas, búsqueda lineal y búsqueda binaria, y cada uno tenia que buscar por nombre y por interprete

|  |
| --- |
| **Main.cpp** |
| **#include <iostream>**  **#include "menu.h"**  ***using* *namespace* std;**  **int main() {**  **Menu menu;**  ***return* 0;**  **}** |

|  |
| --- |
| **Menu.h** |
| **#ifndef MENU\_H**  **#define MENU\_H**  **#include "list.h"**  **#include "list.cpp"**  **#include "songs.h"**  ***class* Menu {**  ***private*:**  **List<Songs> songs;*/\*lista* *de* *canciones\*/***  **Songs song; */\*back* *de* *la* *cancion* *a* *agregar\*/***  ***public*:**  **Menu();**  **void add();**  **void addPosition(*const* Songs &);**  **void show(*const* int &);**  **void showAll();**  **void erase();**  **void gotoxy(int, int);**  **void findL();**  **void findB();**  ***enum* Options {**  ***optionAdd* = 1,**  ***optionShow*,**  ***optionFind*,**  ***optionErase*,**  ***optionOut***  **};**  **};**  **#endif *//* *MENU\_H*** |

|  |
| --- |
| **Menu.cpp** |
| **#include "menu.h"**  **#include <windows.h>**  ***using* *namespace* std;**  **Menu::Menu() {**  **int option;**  ***do*{**  **system("cls");**  **cout << "\t\t\t\t\t\t .:MENU:." << endl;**  ***if*(songs.empty()) {**  **cout << "\t\t\t\t\t .:LISTA VACIA:." << endl;**  **} *else* {**  **showAll();**  **}**  **cout << optionAdd << ".- Inertar" << endl**  **<< optionShow << ".- Mostrar" << endl**  **<< optionFind << ".- Buscar" << endl**  **<< optionErase << ".- Borrar" << endl**  **<< optionOut << ".- salir" << endl**  **<< "Elige una opcion: ";**  **cin >> option;**  **cin.ignore();**  ***switch* (option) {**  ***case* optionAdd: add();**  ***break*;**  ***case* optionShow:**  **int position;**  **cout << "Ingresa el numero de cancion a mostrar: ";**  **cin >> position;**  **show(position);**  ***break*;**  ***case* optionFind:**  **cout << "tu busqueda es" << endl**  **<< "1.- lineal" << endl**  **<< "2.- binaria" << endl;**  **cin >> option;**  **cin.ignore();**  ***switch* (option) {**  ***case* 1:**  **findL();**  ***break*;**  ***case* 2:**  **findB();**  ***break*;**  **}**  ***break*;**  ***case* optionErase: erase();**  ***break*;**  ***case* optionOut:**  ***break*;**  ***default*:**  **cout << "valor invalido";**  **}**  **system("pause");**  **} *while*(option != optionOut);**  **}**  **void Menu::add() {**  **string data;**  **int ranking, position = 0;**  **cout << "Nombre de la cancion: ";**  **getline(cin, data);**  **song.setTitle(data);**  **cout << "Nombre del autor: ";**  **getline(cin, data);**  **song.setAuthor(data);**  **cout << "Nombre del interprete: ";**  **getline(cin, data);**  **song.setInterprete(data);**  ***do*{**  **cout << "\n formato '01:23'\nDuracion de la cancion: ";**  **getline(cin, data);**  **} *while*(!song.validTime(data));**  **song.setDuration(data);**  **cout << "Posicion del ranking: ";**  **cin >> ranking;*/\*por* *validar\*/***  **song.setRanking(ranking);**  **cin.ignore();**  ***if*(!songs.empty()) {**  **cout << "desea escojer el punte de inserccion, 1/0: ";**  **cin >> position;**  **cin.ignore();**  **}**  ***if*(position == 1) {**  **addPosition(song);**  **} *else* {**  **songs.insert(song);**  **}**  **}**  **void Menu::addPosition(*const* Songs &newSong) {**  **int position;**  **string option;**  ***do* {**  **cout << "Posicion de interes: ";**  **cin >> position;*/\*por* *validar\*/***  **cout << "1.- antes del punto de interes" << endl**  **<< "2.- Despues del punto de interes" << endl**  **<< "opcion: ";**  **cin >> option;**  ***if*(option == "1") {**  **songs.insertPosition(newSong, songs.before(position));**  **option = "0";**  **} *else* *if*(option == "2") {**  **songs.insertPosition(newSong, songs.after(position));**  **option = "0";**  **} *else* {**  **cout << "Opcion invalida" << endl;**  **}**  **} *while*(option != "0");**  **}**  **void Menu::show(*const* int &position) {**  ***if*(songs.empty()) {**  **cout << "La lista esta vacia" << endl;**  **} *else* {**  **cin.ignore();**  **cout << "Titulo: "**  **<< songs.show(position).getTitle() << endl**  **<< "Autor: "**  **<< songs.show(position).getAuthor() << endl**  **<< "Interprete: "**  **<< songs.show(position).getInterprete() << endl**  **<< "Duracion: "**  **<< songs.show(position).getDuration() << endl**  **<< "Posicion en Rangking: "**  **<< songs.show(position).getRanking() << endl;**  **}**  **}**  **void Menu::showAll() {**  **cout << "Pocicion| Titulo\t\t| Autor\t\t\t| Interprete\t\t| Duracion | Ranking |" << endl;**  ***for* (int i = 0; i <= songs.last(); i++) {**  ***for* (int i=0;i<102;i++) {**  **cout << "\_";**  **}**  **gotoxy(3, i+2);**  **cout << i;**  **gotoxy(8, i+2);**  **cout << "| ";**  **cout << songs.show(i).getTitle();**  **gotoxy(32, i+2);**  **cout << "| ";**  **cout << songs.show(i).getAuthor();**  **gotoxy(56, i+2);**  **cout << "| ";**  **cout << songs.show(i).getInterprete();**  **gotoxy(80, i+2);**  **cout << "| ";**  **cout << songs.show(i).getDuration();**  **gotoxy(91, i+2);**  **cout << "| ";**  **gotoxy(96, i+2);**  **cout << songs.show(i).getRanking();**  **gotoxy(101, i+2);**  **cout << "|" << endl;**  **}**  **cout << endl;**  **}**  **void Menu::erase() {**  ***if*(songs.empty()) {**  **cout << "La lista esta vacia" << endl;**  **} *else* {**  **int position;**  **cout << "Ingresa la posicion del dato a eliminar:";**  **cin >> position;**  **cin.ignore();**  **songs.erase(position);**  **}**  **}**  **void Menu::gotoxy(int x, int y) {**  **HANDLE hcon = GetStdHandle(STD\_OUTPUT\_HANDLE);**  **COORD dwPos;**  **dwPos.X = x;**  **dwPos.Y = y;**  **SetConsoleCursorPosition(hcon, dwPos);**  **}**  **void Menu::findL() {**  **string name, interprete;**  **int option;**  **bool no = *false*;**  **cout << "Busqueda lineal" << endl**  **<< "1.- nombre" << endl**  **<< "2.- interprete" << endl;**  **cin >> option;**  **cin.ignore();**  ***switch* (option) {**  ***case* 1:**  **cout << "dame el nombre: " << endl;**  **getline(*cin*, *name*);**  ***for* (int i = 0; i <= songs.last(); i++) {**  ***if*(songs.show(i).getTitle() == name) {**  **show(i);*/\*la* *cancion* *encontrada\*/***  **}**  **}**  ***if*(!no) {**  **cout << "no se encontro la cancion" << endl;**  **system("pause");**  **}**  ***break*;**  ***case* 2:**  **cout << "dame el interprete: ";**  **getline(*cin*, *interprete*);**  ***for* (int i = 0; i <= songs.last(); i++) {**  ***if*(songs.show(i).getInterprete() == interprete) {**  **show(i);*/\*la* *cancion* *encontrada\*/***  **}**  **}**  ***if*(!no) {**  **cout << "no se encontro la cancion" << endl;**  **system("pause");**  **}**  ***break*;**  **}**  **}**  **void Menu::findB() {**  **string name, interprete;**  **int option, i(0), j(songs.last()), m;**  **bool no = *false*;**  **cout << "Busqueda binaria" << endl**  **<< "1.- nombre" << endl**  **<< "2.- interprete" << endl;**  **cin >> option;**  **cin.ignore();**  ***switch* (option) {**  ***case* 1:**  **cout << "dame el nombre: " << endl;**  **getline(*cin*, *name*);**  ***while*(i <= j) {**  **m = (i+j) / 2;**  ***if*(songs.show(m).getTitle() == name) {**  **show(m);**  **no = *true*;**  **}**  ***if*(name < songs.show(m).getTitle()) {**  **j = m-1;**  **} *else* {**  **i = m+1;**  **}**  **}**  ***if*(!no) {**  **cout << "no se encontro la cancion" << endl;**  **system("pause");**  **}**  ***break*;**  ***case* 2:**  **cout << "dame el interprete: ";**  **getline(*cin*, *interprete*);**  ***while*(i <= j) {**  **m = (i+j) / 2;**  ***if*(songs.show(m).getInterprete() == interprete) {**  **show(m);**  **no = *true*;**  **}**  ***if*(name < songs.show(m).getInterprete()) {**  **j = m-1;**  **} *else* {**  **i = m+1;**  **}**  **}**  ***if*(!no) {**  **cout << "no se encontro la cancion" << endl;**  **system("pause");**  **}**  ***break*;**  **}**  **}**  ***//* *songs.findDataL();*** |

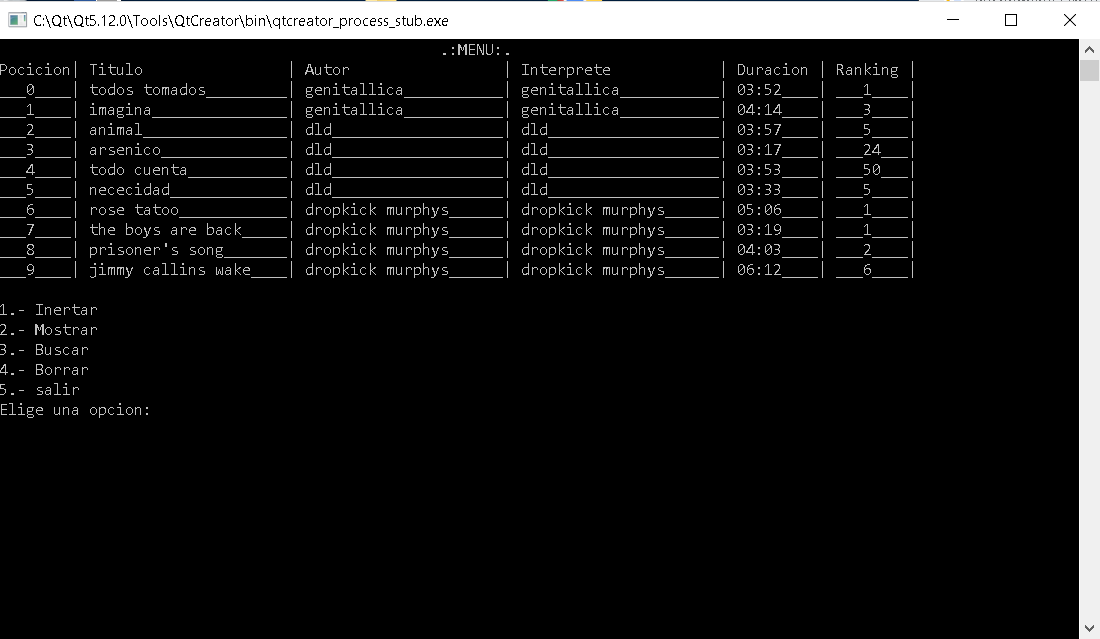
|  |
| --- |
| **Songs.h** |
| **#ifndef SONGS\_H**  **#define SONGS\_H**  **#include <iostream>**  ***class* Songs {**  ***private*:**  **std::string title;*/\*titulo* *de* *la* *cancion\*/***  **std::string author;*/\*autor\*/***  **std::string interprete;*/\** *interprete\*/***  **std::string duration;*/\*duraccion* *de* *la* *cancion\*/***  **int ranking;*/\*posicion* *en* *el* *ranking\*/***  ***public*:**  **Songs();**  **std::string getTitle() *const*;**  **void setTitle(*const* std::string &);**  **std::string getAuthor() *const*;**  **void setAuthor(*const* std::string &);**  **std::string getInterprete() *const*;**  **void setInterprete(*const* std::string &);**  **std::string getDuration() *const*;**  **void setDuration(*const* std::string &);**  **int getRanking() *const*;**  **void setRanking(*const* int &value);**  **bool validTime(*const* std::string &);**  **};**  **#endif *//* *SONGS\_H*** |

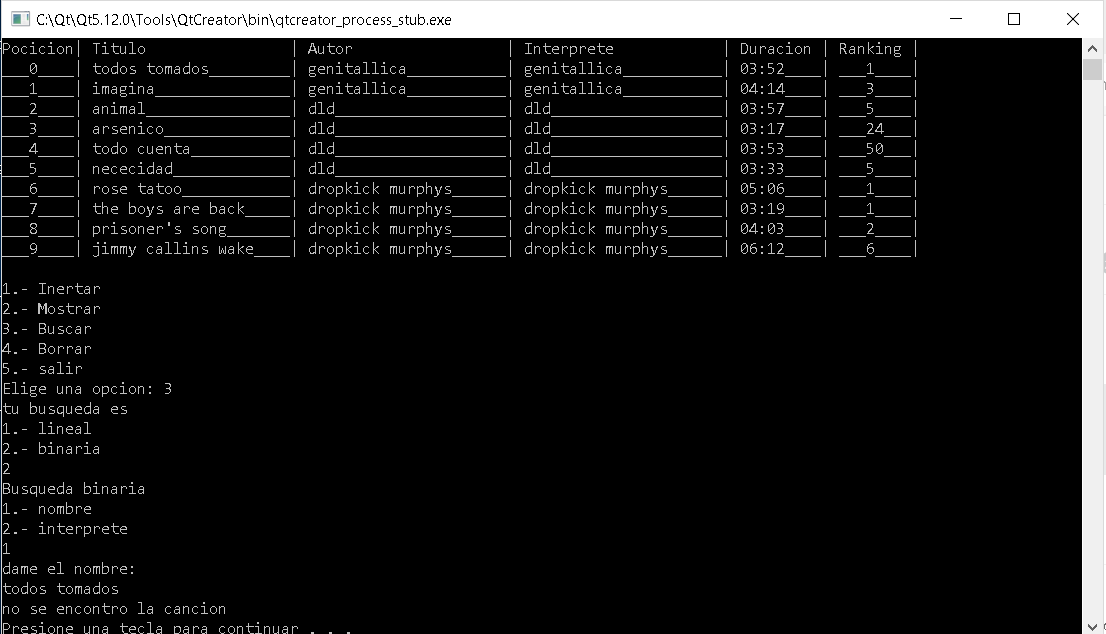
|  |
| --- |
| **Songs.cpp** |
| **#include "songs.h"**  ***using* *namespace* std;**  **Songs::Songs() {**  **}**  **string Songs::getTitle() *const* {**  ***return* title;**  **}**  **void Songs::setTitle(*const* string &value) {**  **title = value;**  **}**  **string Songs::getAuthor() *const* {**  ***return* author;**  **}**  **void Songs::setAuthor(*const* string &value) {**  **author = value;**  **}**  **string Songs::getInterprete() *const* {**  ***return* interprete;**  **}**  **void Songs::setInterprete(*const* string &value) {**  **interprete = value;**  **}**  **string Songs::getDuration() *const* {**  ***return* duration;**  **}**  **void Songs::setDuration(*const* string &value) {**  **duration = value;**  **}**  **int Songs::getRanking() *const* {**  ***return* ranking;**  **}**  **void Songs::setRanking(*const* int &value) {**  **ranking = value;**  **}**  **bool Songs::validTime(*const* string &value) {**  ***if*(value.size() != 5) {**  ***/\*si* *no* *tiene* *estilo* *de* *tiempo* *'01:23'* *no* *es* *valido***  ***5* *digitos\*/***  ***return* *false*;**  **}**  ***for* (int i = 0; i < 5; i++) {**  ***if*(i != 2) {**  ***/\*aqui* *solo* *analisa* *los* *digitos\*/***  ***if*(value[i] < 48 *or* value[i] > 57) {**  ***/\*aqui* *se* *revisa* *que* *si* *sean* *digitos\*/***  ***return* *false*;**  **}**  **} *else* *if*(value[i] != 58) {**  ***/\*aqui* *se* *revisa* *el* *':'\*/***  ***return* *false*;**  **}**  **}**  ***/\*si* *paso* *todo* *sin* *retornar* *falso,* *el* *dato* *introduccido* *es* *valido\*/***  ***return* *true*;**  **}** |

|  |
| --- |
| **List.h** |
| **#ifndef LIST\_H**  **#define LIST\_H**  **#include <iostream>**  ***template* <*typename* Type>**  ***class* List {**  ***private*:**  ***static* *const* int SIZE = 50;**  **Type data[SIZE];**  **int counter;**  **bool validPosition(*const* int &);**  ***public*:**  **List();**  **bool empty();*/\*revisa* *si* *esta* *vacia\*/***  **bool full();*/\*revisa* *si* *esta* *llena\*/***  **void insert(*const* Type &);*/\*elemento* *a* *insertar* *al* *final* *de* *la* *lista\*/***  **void insertPosition(*const* Type &, *const* int &position);**  ***/\*inserta* *elemento* *en* *posicion* *elejida,* *"dato,* *lugar"\*/***  **void erase(int &);*/\*borra* *un* *dato* *de* *la* *lista\*/***  **int first();*/\*devuelve* *la* *primer* *posocion\*/***  **int last();*/\*devuelve* *la* *ultima* *posicion\*/***  **int before(*const* int &);*/\*anterior,* *devuleve* *posicion* *actual\*/***  **int after(*const* int &);*/\*siguente,* *devuelve* *posicion* *siguente\*/***  **Type show(*const* int &);*/\*retorna* *el* *elemento* *para* *poder* *mostrarlo\*/***  **void remove();*/\*elimina* *toda* *la* *lista\*/***  **};**  **#endif *//* *LIST\_H*** |

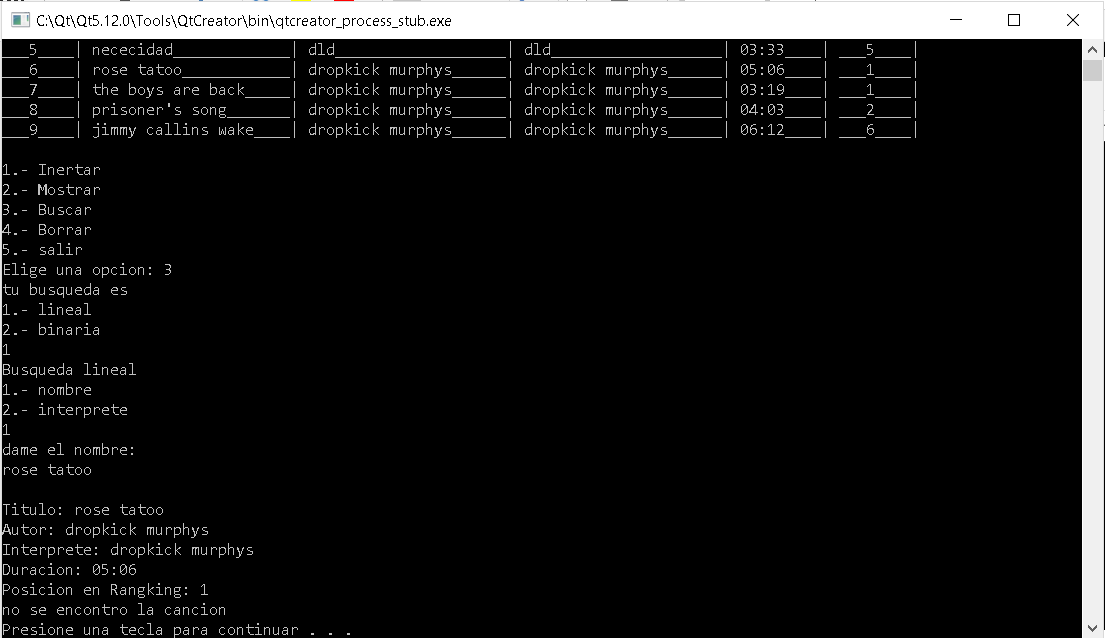
|  |
| --- |
| **List.cpp** |
| **#include "list.h"**  ***template*<*typename* Type>**  **List<Type>::List() : counter(0) { }**  ***template*<*typename* Type>**  **bool List<Type>::validPosition(*const* int &position) {**  ***if*(position >= counter *or* position < 0) {**  ***return* *false*;**  **}**  ***return* *true*;**  **}**  ***template*<*typename* Type>**  **bool List<Type>::empty() {**  ***return* counter == 0;**  **}**  ***template*<*typename* Type>**  **bool List<Type>::full() {**  ***return* counter == SIZE;**  **}**  ***template*<*typename* Type>**  **void List<Type>::insert(*const* Type &newElement) {**  ***if*(full()) {**  **std::cout << std::endl << "la lista esta llena" << std::endl;**  **} *else* {**  **data[counter] = newElement;**  **counter++;**  **}**  **}**  ***template*<*typename* Type>**  **void List<Type>::insertPosition(*const* Type &newElement, *const* int &position) {**  ***if*(full()) {**  **std::cout << std::endl << "la lista esta llena" << std::endl;**  **} *else* *if*(!validPosition(position)) {**  **std::cout << std::endl << "posicion invalida" << std::endl;**  **} *else* {**  ***for* (int i(counter); i >= position; i--) {**  **data[i+1] = data[i];**  **}**  **data[position] = newElement;**  **counter++;**  **}**  **}**  ***template*<*typename* Type>**  **void List<Type>::erase(int &position) {**  **position--;**  ***if*(!validPosition(position)) {**  **std::cout << "posicion invalida" << std::endl;**  **} *else* {**  ***for* (int i = position; i < counter; i++) {**  **data[i] = data[i+1];**  **}**  **counter--;**  **}**  **}**  ***template*<*typename* Type>**  **int List<Type>::first() {**  ***if*(empty()) {**  ***return* -1;**  **}**  ***return* 0;**  **}**  ***template*<*typename* Type>**  **int List<Type>::last() {**  ***return* counter-1;**  **}**  ***template*<*typename* Type>**  **int List<Type>::before(*const* int &position) {**  ***if*(!validPosition(position)) {**  ***return* -1;**  **}**  ***return* position-1;**  **}**  ***template*<*typename* Type>**  **int List<Type>::after(*const* int &position) {**  ***if*(!validPosition(position)) {**  ***return* -1;**  **}**  ***return* position+1;**  **}**  ***template*<*typename* Type>**  **Type List<Type>::show(*const* int &position) {**  ***if*(empty()){**  **std::cout << "la lista esta vacia" << std::endl;**  **} *else* *if*(!validPosition(position)) {**  **std::cout << "posicion invalida" << std::endl;**  **} *else* {**  ***return* data[position];**  **}**  **}**  ***template*<*typename* Type>**  **void List<Type>::remove() {**  **counter = 0;**  **}** |

|  |
| --- |
| **CAPTURAS DE PANTALLA** |

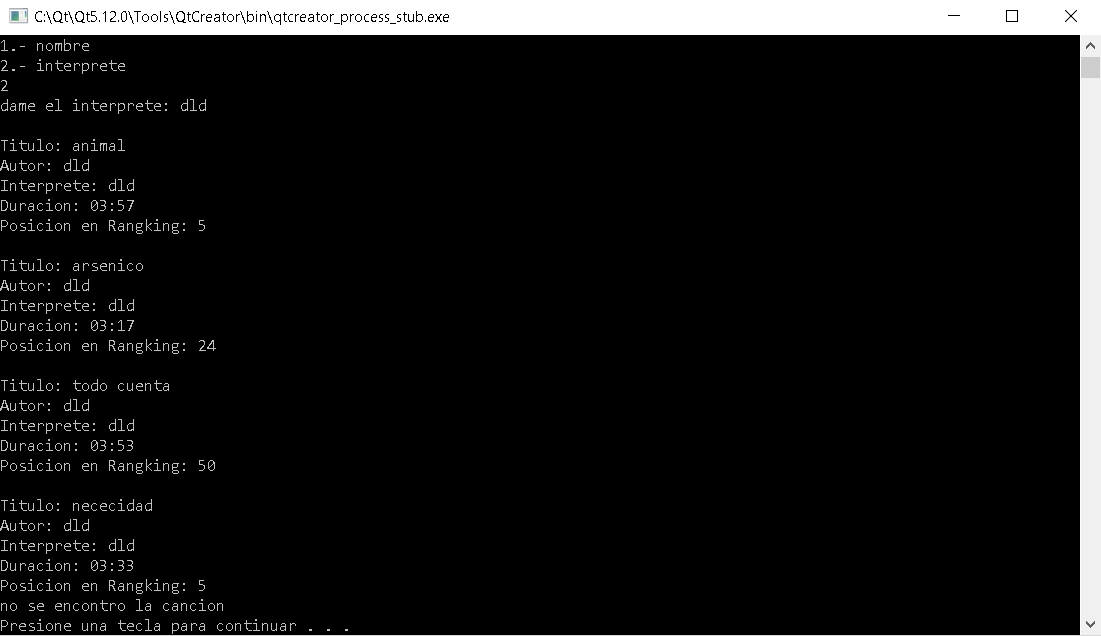




Aquí podemos ver que la búsqueda binaria no sirve si no están ordenados los, ya que me dice que no se encuentra la canción cuando si existe



Aquí vemos que la búsqueda lineal no genera ningún problema



Aquí buscamos por inteprete y me devuelve las 4 canciones de ese mismo artista