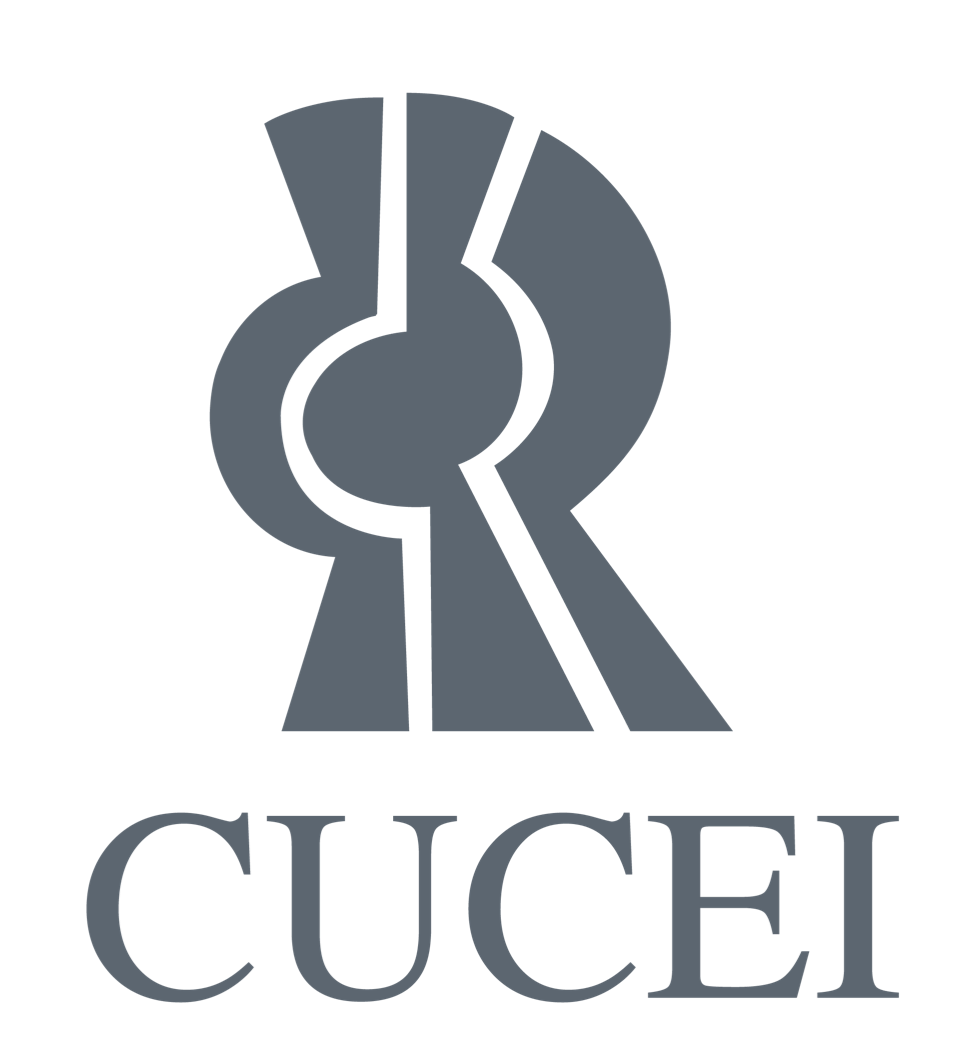
16-3-2019

**david gutierrez alvarez**

**Estructura de datos I**

****

## Apuntadores

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

Me agrado la actividad, ya que solo me tuve que encargar de modificar la parte interna del programa “la lista” y el como trabaja, pero al momento de usarlo y de forma visual funciona igual, los cambios solo son internos y de rendimiento.

Esta actividad no fue complicada, el echo de saber como funcionan los punteros me ayudo a evitar muchos de los errores que pudieron ocurrir.

|  |
| --- |
| **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 erase();**  **void findL();**  **void findB();**  **void order();**  **void change(*const* int &);**  ***enum* Options {**  ***optionAdd* = 1,**  ***optionShow*,**  ***optionFind*,**  ***optionOrder*,**  ***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* {**  **cout << "Pocicion| Titulo\t\t| Autor\t\t\t| Interprete\t\t| Duracion | Ranking |" << endl;**  **songs.print();**  **}**  **cout << *optionAdd* << ".- Insertar" << endl**  **<< *optionShow* << ".- Mostrar" << endl**  **<< *optionFind* << ".- Buscar" << endl**  **<< *optionOrder* << ".- Ordenar" << 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;**  **cout << endl << "Pocicion| Titulo\t\t| Autor\t\t\t| Interprete\t\t| Duracion | Ranking |" << endl;**  **songs.print(position);**  **system("pause");**  ***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* *optionOrder*: order();**  ***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.insert(newSong, songs.getPrev(position));**  **option = "0";**  **} *else* *if*(option == "2") {**  **songs.insert(newSong, songs.getNext(position));**  **option = "0";**  **} *else* {**  **cout << "Opcion invalida" << endl;**  **}**  **} *while*(option != "0");**  **}**  **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::findL() {**  **string name, interprete;**  **int option;**  **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*);**  **song.setTitle(name);**  ***break*;**  ***case* 2:**  **cout << "dame el interprete: ";**  **getline(*cin*, *interprete*);**  **song.setInterprete(interprete);**  **song.setOrder(option);*/\*con* *esto* *analiza* *el* *interprete* *en* *vez* *del* *titulo\*/***  ***break*;**  **}**  **songs.print(songs.find(*song*));**  **system("pause");**  **}**  **void Menu::findB() {**  **string name, interprete;**  **int option;**  **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*);**  **song.setTitle(name);**  **songs.findB(*song*);**  ***break*;**  ***case* 2:**  **cout << "dame el interprete: ";**  **getline(*cin*, *interprete*);**  **song.setInterprete(interprete);**  **song.setOrder(option);*/\*con* *esto* *analiza* *el* *interprete* *en* *vez* *del* *titulo\*/***  ***break*;**  **}**  **songs.print(songs.findB(*song*));**  **system("pause");**  **}**  **void Menu::order() {**  **string name, interprete;**  **int option;**  **cout << "ordenar lista" << endl**  **<< "1.- titulo" << endl**  **<< "2.- interprete" << endl;**  **cin >> option;**  **cin.ignore();**  ***switch* (option) {**  ***case* 1:**  **change(0);*/\*asigna* *al* *titulo* *como* *valor* *a* *comparar\*/***  ***break*;**  ***case* 2:**  **change(1);*/\*asigna* *al* *interprete* *como* *valor* *a* *comparar\*/***  ***break*;**  **}**  **cout << "que metodo de ordenamiento quieres utilizar" << endl**  **<< "1.- bubleSort" << endl**  **<< "2.- shellSort" << endl**  **<< "3.- insertionSort" << endl**  **<< "4.- selectSort" << endl;**  **cin >> option;**  **cin.ignore();**  ***switch* (option) {**  ***case* 1: songs.bubble();**  ***break*;**  ***case* 2: songs.shell();**  ***break*;**  ***case* 3: songs.insertion();**  ***break*;**  ***case* 4: songs.select();**  ***break*;**  **}**  **}**  **void Menu::change(*const* int &e) {**  ***for* (int i(0) ;i <= songs.getLast() ;i++) {**  **songs[i].setOrder(e);**  **}**  **system("pause");**  **}** |

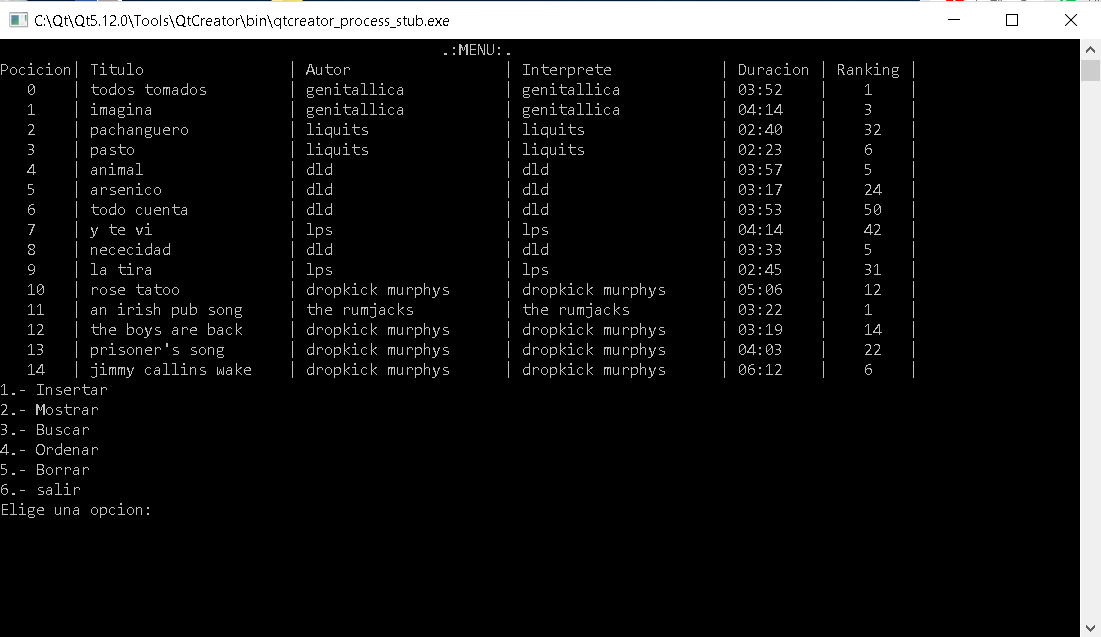
|  |
| --- |
| **Songs.h** |
| **#ifndef SONGS\_H**  **#define SONGS\_H**  **#include <iostream>**  **#include "cursor.h"**  ***class* Songs {**  ***private*:**  **std::string title;*/\*titulo* *de* *la* *cancion\*/***  **std::string author;*/\*autor\*/***  **std::string interprete;*/\** *interprete\*/***  **std::string duration;*/\*duraccion* *de* *la* *cancion\*/***  **int ranking;*/\*posicion* *en* *el* *ranking\*/***  ***public*:**  **int order;**  **Songs();**  **Songs(*const* Songs &);**  **Songs *operator*=(*const* Songs &);**  **bool *operator*==(*const* Songs &) *const*;**  **bool *operator*!=(*const* Songs &) *const*;**  **bool *operator*<(*const* Songs &) *const*;**  **bool *operator*>(*const* Songs &) *const*;**  **bool *operator*<=(*const* Songs &) *const*;**  **bool *operator*>=(*const* Songs &) *const*;**  ***//Funcion* *Amiga* *para* *Serealizar* *el* *objeto***  ***friend* std::ostream &*operator*<<(std::ostream &, *const* 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 &);**  **int getOrder() *const*;**  **void setOrder(*const* int &);**  **};**  **#endif *//* *SONGS\_H*** |

|  |
| --- |
| **Songs.cpp** |
| **#include "songs.h"**  ***using* *namespace* std;**  **int Songs::getOrder() *const* {**  ***return* order;**  **}**  **void Songs::setOrder(*const* int &ord) {**  **order = ord;**  **}**  **Songs::Songs() : order(0) { }**  **Songs::Songs(*const* Songs &copy) : title(copy.title), author(copy.author), interprete(copy.interprete), duration(copy.duration), ranking(copy.ranking){ }**  **Songs Songs::*operator*=(*const* Songs &copy) {**  **title = copy.title;**  **author = copy.author;**  **interprete = copy.interprete;**  **duration = copy.duration;**  **ranking = copy.ranking;**  ***return* \**this*;**  **}**  **bool Songs::*operator*==(*const* Songs &comp) *const* {**  ***if*(comp.order == 0) {**  ***return* *this*->title == comp.title;**  **}**  ***return* *this*->interprete == comp.interprete;**  **}**  **bool Songs::*operator*!=(*const* Songs &comp) *const* {**  ***if*(comp.order == 0) {**  ***return* *this*->title != comp.title;**  **}**  ***return* *this*->interprete != comp.interprete;**  **}**  **bool Songs::*operator*>(*const* Songs &comp) *const* {**  ***if*(comp.order == 0) {**  ***return* *this*->title > comp.title;**  **}**  ***return* *this*->interprete > comp.interprete;**  **}**  **bool Songs::*operator*<(*const* Songs &comp) *const* {**  ***if*(comp.order == 0) {**  ***return* *this*->title < comp.title;**  **}**  ***return* *this*->interprete < comp.interprete;**  **}**  **bool Songs::*operator*<=(*const* Songs &comp) *const* {**  ***if*(comp.order == 0) {**  ***return* *this*->title <= comp.title;**  **}**  ***return* *this*->interprete <= comp.interprete;**  **}**  **bool Songs::*operator*>=(*const* Songs &comp) *const* {**  ***if*(comp.order == 0) {**  ***return* *this*->title >= comp.title;**  **}**  ***return* *this*->interprete >= comp.interprete;**  **}**  **ostream &*operator*<<(ostream &os,*const* Songs &song) {*/\*toString\*/***  **Cursor cursor;**  **cursor.Gotoxy(8, cursor.wherey());**  **os << "| ";**  **os << song.getTitle();**  **cursor.Gotoxy(32, cursor.wherey());**  **os << "| ";**  **os << song.getAuthor();**  **cursor.Gotoxy(56, cursor.wherey());**  **os << "| ";**  **os << song.getInterprete();**  **cursor.Gotoxy(80, cursor.wherey());**  **os << "| ";**  **os << song.getDuration();**  **cursor.Gotoxy(91, cursor.wherey());**  **os << "| ";**  **cursor.Gotoxy(96, cursor.wherey());**  **os << song.getRanking();**  **cursor.Gotoxy(101, cursor.wherey());**  **os << "| " << endl;**  ***return* os;**  **}**  **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*;**  **}** |

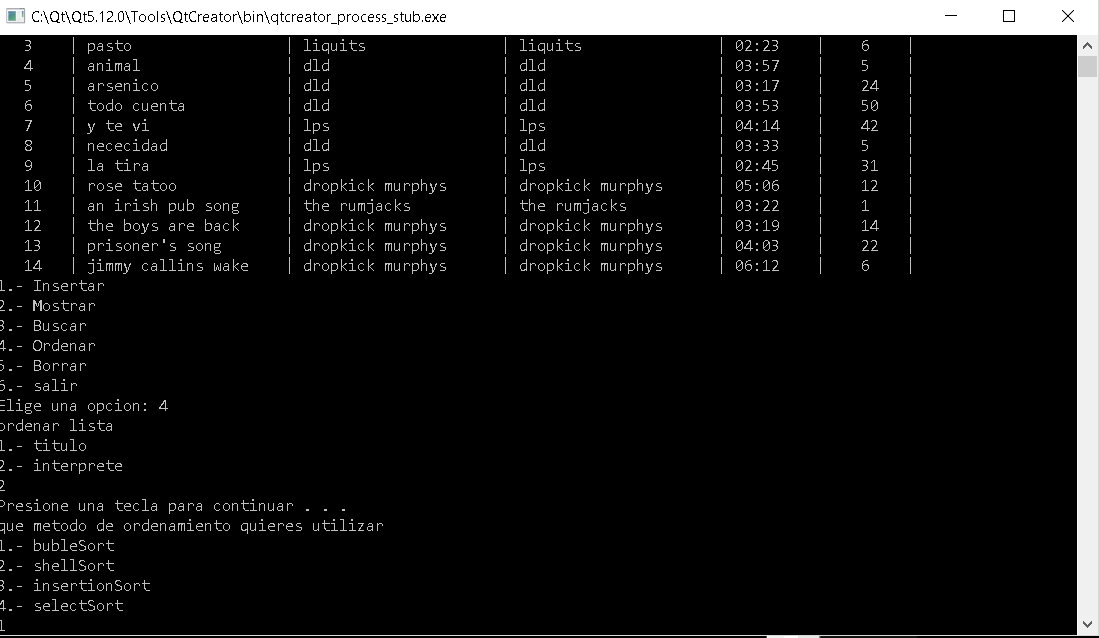
|  |
| --- |
| **List.h** |
| **#ifndef LIST\_H**  **#define LIST\_H**  **#include <iostream>**  **#include <iostream>**  **#include <string>**  ***template* <*typename* Type, int ARRAYSIZE = 3000>**  ***class* List {**  ***private*:**  **Type \*\*data;**  **int last;**  **void copyAll(*const* List& l) {**  **deleteAll();**  ***for*(last = -1; last < l.last; data[++last] = *new* Type(\*l.data[last])){**  ***if* (data[last] == *nullptr*)**  ***throw* Exception("Something went wrong in List constructor, memory not available");**  **}**  **}**  **bool validPos(*const* int& p) {**  ***return* p >= 0 *or* p <= last;**  **}**  ***//void* *mergeSort(const* *int* *&left,* *const* *int* *&right);***  **void sort(*const* int& l, *const* int& r) {*/\*Quick\*/***  ***if*(l >= r)**  ***return*;**  ***if* (l + 1 == r) {**  ***if*(\*data[l] > \*data[r]){**  **std::swap(data[l], data[r]);**  **}**  ***return*;**  **}**  **int i = l, j = r;**  ***while*(i < j) {**  ***while*(i < j *and* \*data[i] <= \*data[r]) {**  **i++;**  **}**  ***while*(i < j *and* \*data[j] >= \*data[r]) {**  **j--;**  **}**  ***if*(i != j) {**  **std::swap(data[i], data[j]);**  **}**  **}**  ***if*(i != r){**  **std::swap(data[i], data[r]);**  **}**  ***if*(i > l) {**  **sort(l, i - 1);**  **}**  ***if*(i < j) {**  **sort(i + 1, j);**  **}**  **}**  ***public*:**  ***class* Exception : *public* std::exception {**  ***private*:**  **std::string msg;**  ***public*:**  ***explicit* Exception(*const* char\* message) : msg(message) { }**  ***explicit* Exception(*const* std::string& message) : msg(message) { }**  ***virtual* ~*Exception*() *throw* () { }**  ***virtual* *const* char\* *what*() *const* *throw* () { *return* msg.c\_str(); }**  **};**  **List() {**  ***if*((data = *new* Type\*[ARRAYSIZE]) == *nullptr*) {**  ***throw* Exception("Something went wrong in List constructor, memory not available");**  **}**  ***for*(last = ARRAYSIZE; last >= 0; data[--last] = *nullptr*);**  **}**  **List(*const* List& l) : List() {**  **copyAll(l);**  **}**  **~List() {**  **deleteAll();**  ***delete*[] \*data;**  **}**  **Type &*operator* [] (int &e) {**  ***if*(empty()) {**  ***throw* Exception("lista vacia, []");**  **}**  ***if*(e > last) {**  ***throw* Exception("posicion invalida, []");**  **}**  ***return* \*data[e];**  **}**  **bool empty() {**  ***return* last == -1;**  **}**  **bool full() {**  ***return* last == ARRAYSIZE - 1;**  **}**  **void insert(*const* Type &e, int p) {**  ***if*(full()) {**  ***throw* Exception("Can not insert data in a full List");**  **}**  ***if*(p != -1 *and* !validPos(p)) {**  ***throw* Exception("There is an invalid position, trying to insert data into List");**  **}**  ***for*(int i = last++; i > p; data[i + 1] = data[i]);**  ***if*((data[p + 1] = *new* Type(e)) == *nullptr*) {**  ***throw* Exception("Something went wrong inserting new data in List");**  **}**  **}**  **void insert(*const* Type &e) {**  **insert(e, getLast());**  **}**  **void erase(int p) {**  ***if*(!validPos(p)) {**  ***throw* Exception("There is an invalid position, trying to delete data from List");**  **}**  ***for*(int i = p; i < last; i++) {**  **data[i] = data[i+1];**  **}**  ***delete* data[last--];**  **}**  **int getFirst() {**  ***return* last == -1 ? -1 : 0;**  **}**  **int getLast() {**  ***return* last;**  **}**  **int getPrev(*const* int& p) {**  ***return* p == 0 *or* !validPos(p) ? -1 : p - 1;**  **}**  **int getNext(*const* int& p) {**  ***return* p == last *or* !validPos(p) ? -1 : p + 1;**  **}**  **int find( Type &e) {**  ***for*(int i = 0; i <= last; i++) {**  ***if*(\*data[i] == e) {**  ***return* i;**  **}**  **}**  ***return* -1;**  **}**  **int findB(Type &e) {*/\*busqueda* *binario\*/***  **int i(0), j(last), m;**  ***while* (i <= j) {**  **m = (i+j) / 2;**  ***if*(\*data[m] == e) {**  ***return* m;**  **}**  ***if*(e < \*data[m]) {**  **j = m-1;**  **} *else* {**  **i = m+1;**  **}**  **}**  ***return* -1;**  **}**  **Type retrieve(*const* int p) {**  ***if*(!validPos(p)) {**  ***throw* Exception("Invalid position, trying to retrieve data from List");**  **}**  ***return* \*data[p];**  **}**  **void sort() {*/\*Quick\*/***  **sort(0, last);**  **}**  **void print() {**  ***for*(int i = 0; i <= last; i++) {**  **std::cout << i << \*data[i];**  **}**  **}**  **void print(*const* int &position) {**  ***if*(empty()){**  ***throw* Exception("la lista esta vacia");**  **} *else* *if*(!validPos(position)) {**  ***throw* Exception("posicion invalida");**  **} *else* {**  **std::cout << " " << position << \*data[position];**  ***//* *return* *this->data[position];***  **}**  ***//return* *data[0];***  **}**  **void deleteAll() {**  ***for*( ; last >= 0; last--) {**  ***delete* data[last];**  **data[last--] = *nullptr*;**  **}**  **}**  **List& *operator* = (*const* List& l) {**  **deleteAll();**  **copyAll(l);**  ***return* \**this*;**  **}**  ***friend* std::ostream& *operator* << (std::ostream& os, List& l) {**  ***for*(int i = 0; i <= l.last; i++) {**  **os << l.data[i] << std::endl;**  **}**  ***return* os;**  **}**  ***friend* std::istream& *operator* >> (std::istream& is, List& l) {**  **Type myType;**  **int i = 0;**  ***while* (is >> myType)**  ***if*((l.data[i++] = *new* Type(myType)) == *nullptr*) {**  ***throw* Exception("Hay una posicion no valida, intentando insertar en >> operator");**  **}**  ***return* is;**  **}**  **void merge();*/\*meotodo* *de* *ordenamiento\*/***  **void bubble() {*/\*Burbuja* *Mejorada\*/***  **int band,i,j;**  **i = last-1;**  ***do* {**  **band=0;**  **j=0;**  ***while*(j < i) {**  ***if*(data[j] > data[j+1]) {**  **std::swap(data[j], data[j+1]);**  **band=1;**  **}**  **j++;**  **}**  **i--;**  **}*while*(band==1);**  **}**  **void shell() {*//shell***  **int dif, i = 0;**  **float fact = 0.75;**  **dif=(last-1)\*fact;**  ***while*(dif>0) {**  ***while*(i<last-1-dif) {**  ***if*(data[i] > data[i+dif]) {**  **std::swap(data[i+dif], data[i]);**  **}**  **i++;**  **}**  **dif\*=fact;**  **}**  **}**  **void insertion() {*//Insersion***  **int i = 1 ,j;**  **Type \*aux;**  ***while*(i < last){**  **aux = data[i];**  **j=i;**  ***while*(j >0 *and* aux < data[j-1]){**  **data[j] = data[j-1];**  **j--;**  **}**  ***if*(i!=j){**  **data[j] = aux;**  **}**  **i++;**  **}**  **}**  **void select() {*//Seleccion***  **int i,j,menor;**  **i=0;**  ***while*(i<last-1){**  **menor=i;**  **j=i+1;**  ***while*(j<last){**  ***if*(data[j] < data[menor])**  **menor=j;**  **j++;**  **}**  ***if*(menor!=i){**  **std::swap(data[i], data[menor]);**  **}**  **i++;**  **}**  **}**  **bool *operator* == (*const* List &l) {**  ***return* *this*->data == l.data;**  **}**  **};**  **#endif *//* *LIST\_H*** |

|  |
| --- |
| **Cursor.h** |
| **#ifndef GOTO\_H**  **#define GOTO\_H**  **#include <windows.h>**  ***class* Cursor {**  ***public*:**  **Cursor() { }**  **void Gotoxy(int x, int y) {**  **HANDLE hcon = GetStdHandle(STD\_OUTPUT\_HANDLE);**  **COORD dwPos;**  **dwPos.X = x;**  **dwPos.Y = y;**  **SetConsoleCursorPosition(hcon, dwPos);**  **}**  **int wherex() {**  **CONSOLE\_SCREEN\_BUFFER\_INFO csbi;**  **GetConsoleScreenBufferInfo(GetStdHandle(STD\_OUTPUT\_HANDLE), &csbi);**  ***return* csbi.dwCursorPosition.X;**  **}**  **int wherey() {**  **CONSOLE\_SCREEN\_BUFFER\_INFO csbi;**  **GetConsoleScreenBufferInfo(GetStdHandle(STD\_OUTPUT\_HANDLE), &csbi);**  ***return* csbi.dwCursorPosition.Y;**  **}**  **};**  **#endif *//* *GOTO\_H*** |

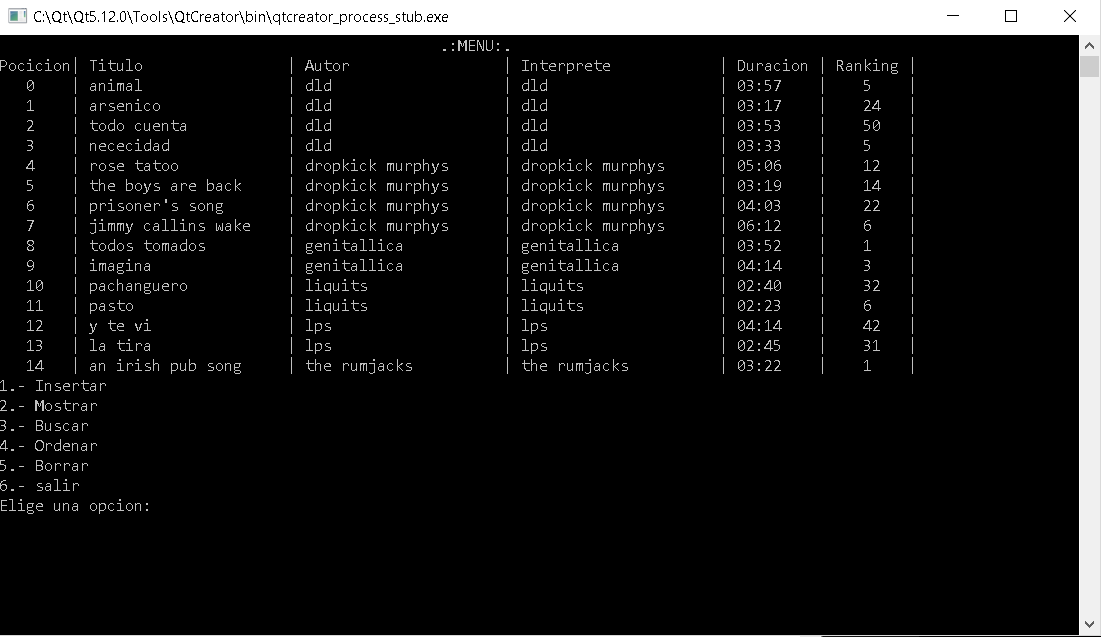
|  |
| --- |
| **CAPTURAS DE PANTALLA** |



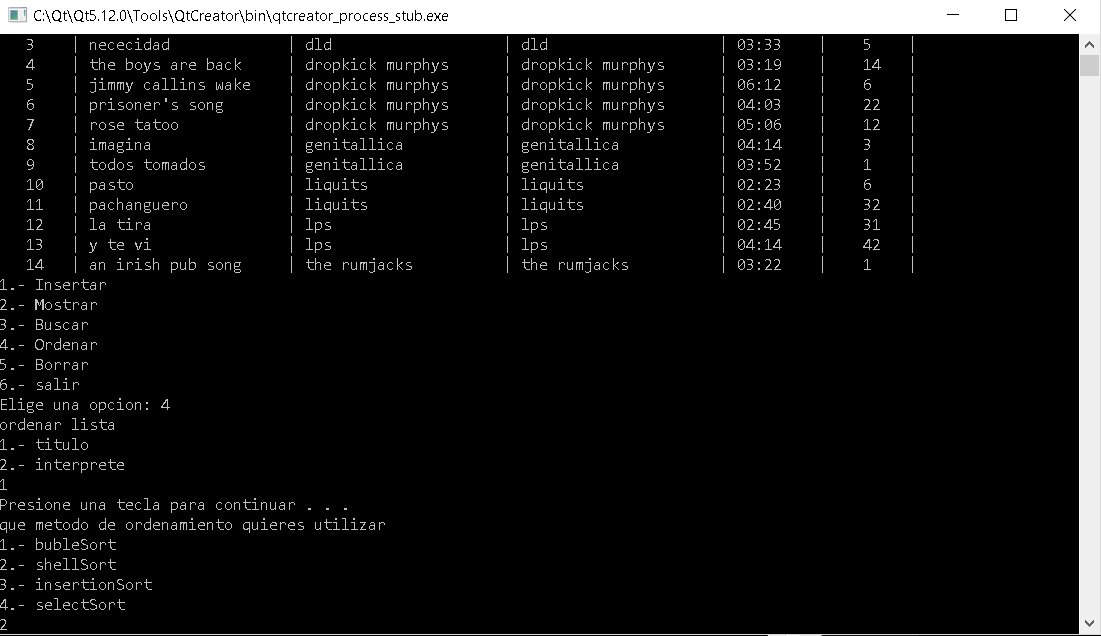
Como vemos aquí, visiblemente el programa no tiene cambios



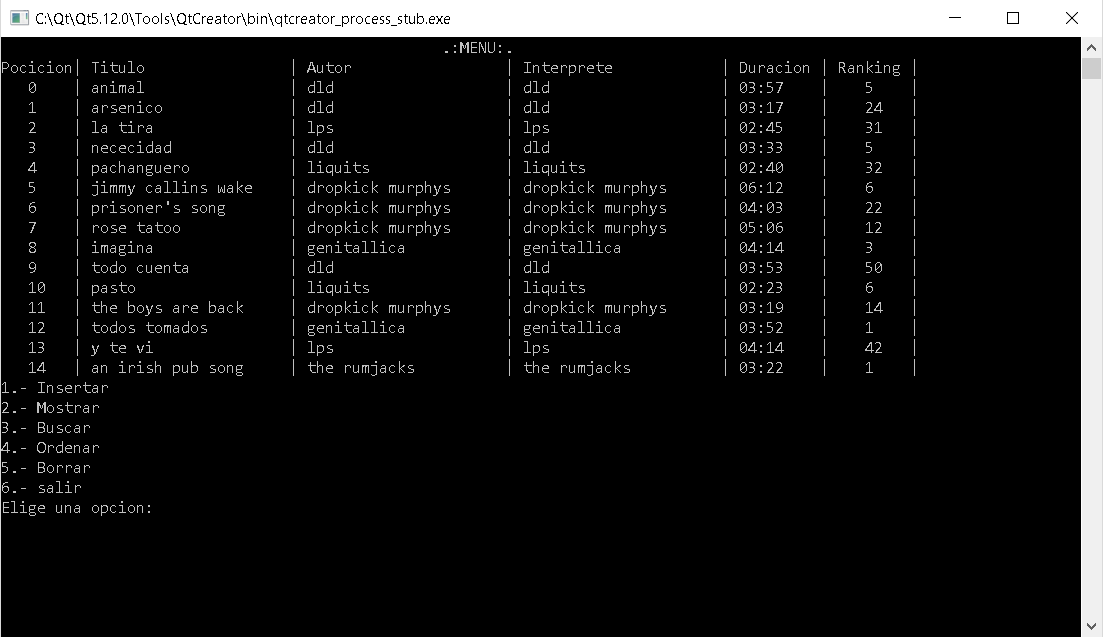
Aquí lo ordenamos por el interprete y con método burbuja



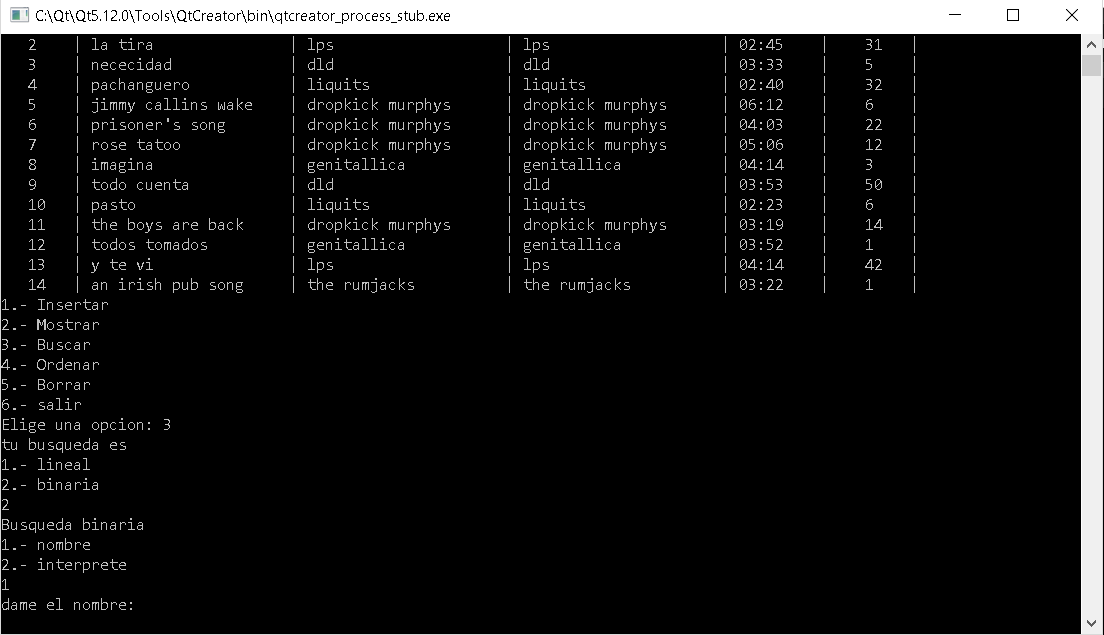
Y vemos que la lista se ordena sin problemas



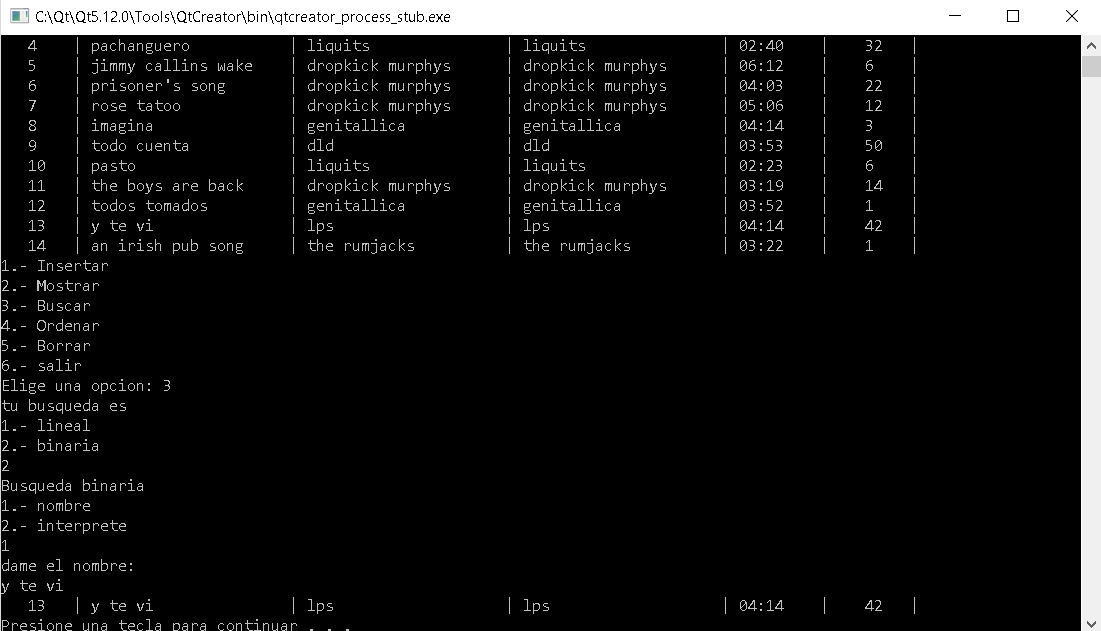
También intentamos ordenar por el titulo



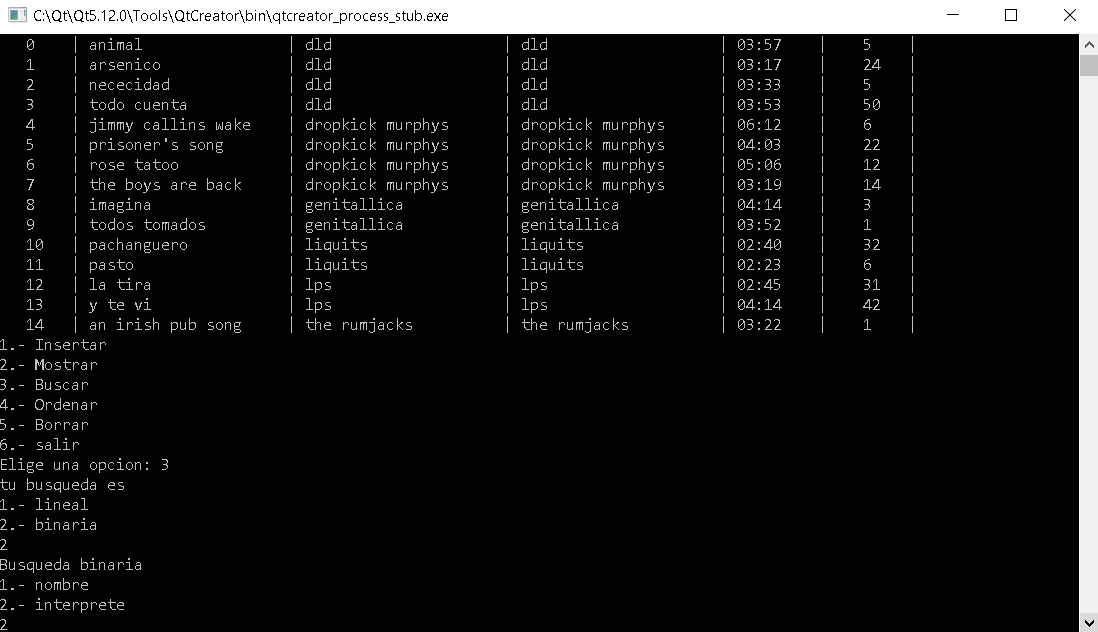
Y de misma manera se ordena



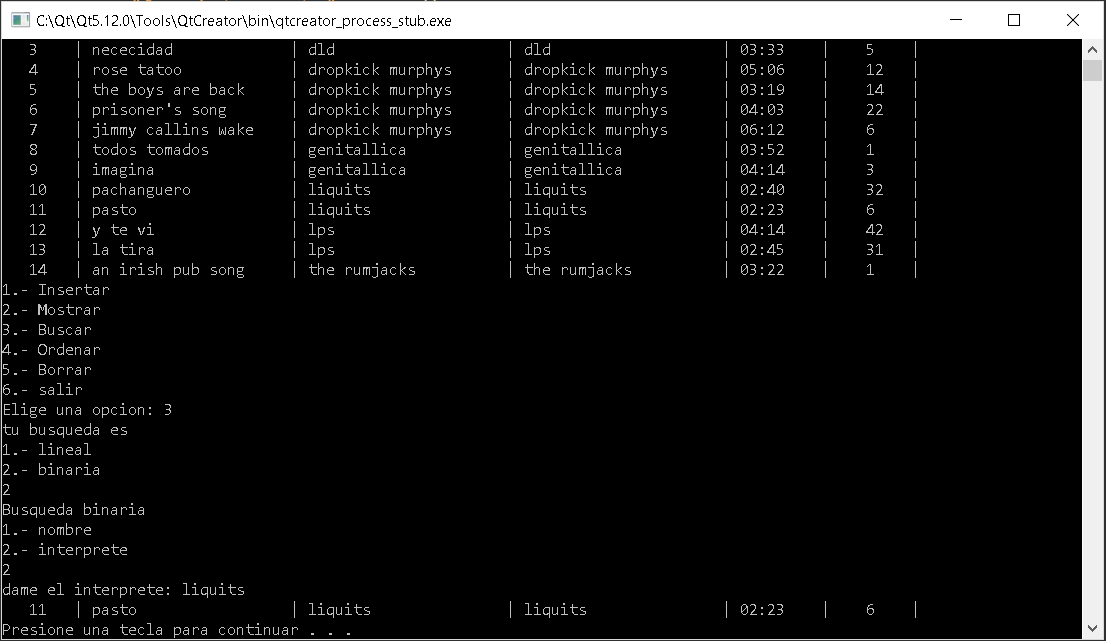
Intentando con la búsqueda binaria por nombre



Y todo funciona perfectamente



Ahora aremos una búsqueda por inteprete



Y como debe suceder, funciona sin generar algún error