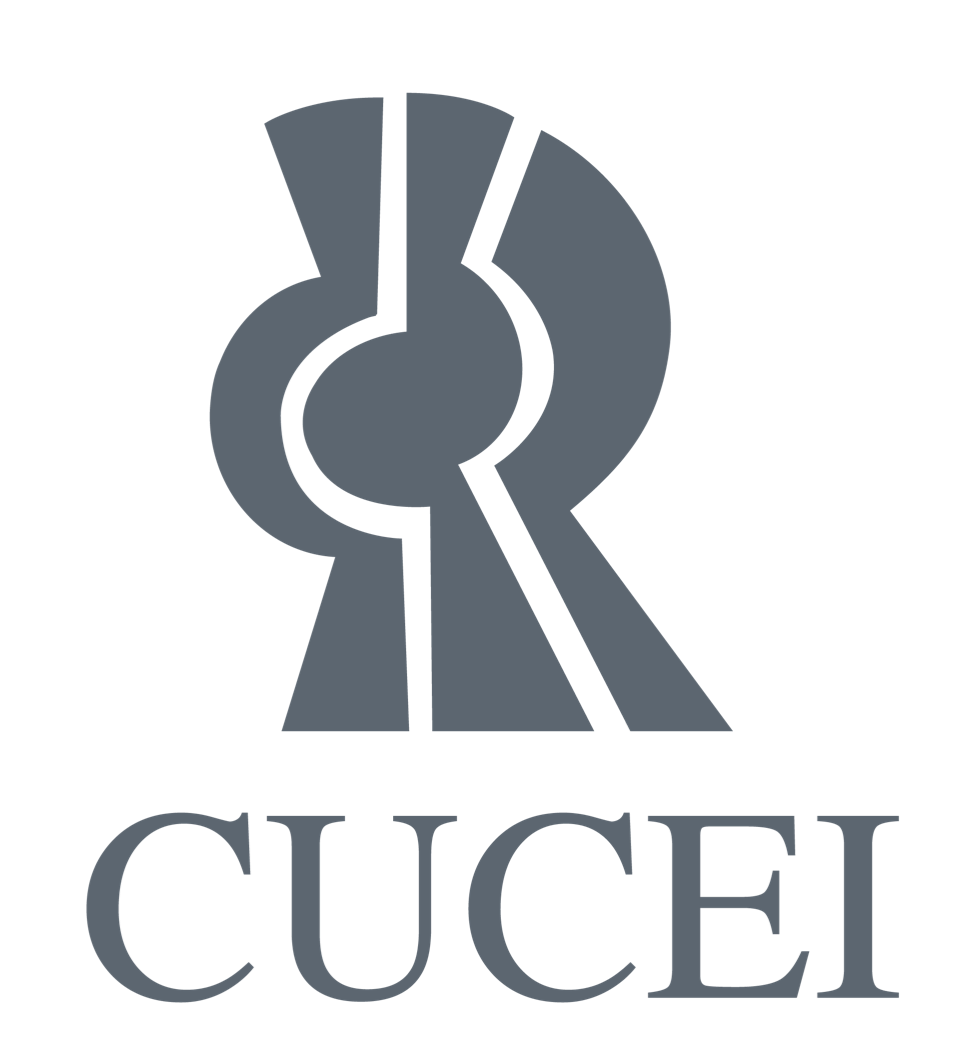
2-3-2019

**david gutierrez alvarez**

**Estructura de datos I**

****

## Métodos de ordenamiento iterativos

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

Esta actividad fue un poco mas complicada, porque tuve que cambiar varias cosas que tenía mal hechas, también tuve que usar sobrecarga de operadores mas inteligentes para que dependiendo el caso buscar dentro de la canción el titulo o el intérprete, también usar la sobrecarga al comparar valores de una forma más eficiente.

A pesar de la complicidad me agrado bastante ya que pude aplicar bastantes cosas aprendidas en clase, me estrese un poco y aprendí mas.

|  |
| --- |
| **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.showAll();**  **}**  **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.show(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.append(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.append(newSong, songs.before(position));**  **option = "0";**  **} *else* *if*(option == "2") {**  **songs.append(newSong, songs.after(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.show(songs.findDataL(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.findDataB(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.show(songs.findDataB(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.bubbleSort();**  ***break*;**  ***case* 2: songs.shellSort();**  ***break*;**  ***case* 3: songs.insertionSort();**  ***break*;**  ***case* 4: songs.selectSort();**  ***break*;**  **}**  **}**  **void Menu::change(*const* int &e) {**  ***for* (int i(0) ;i <= songs.last() ;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*** |

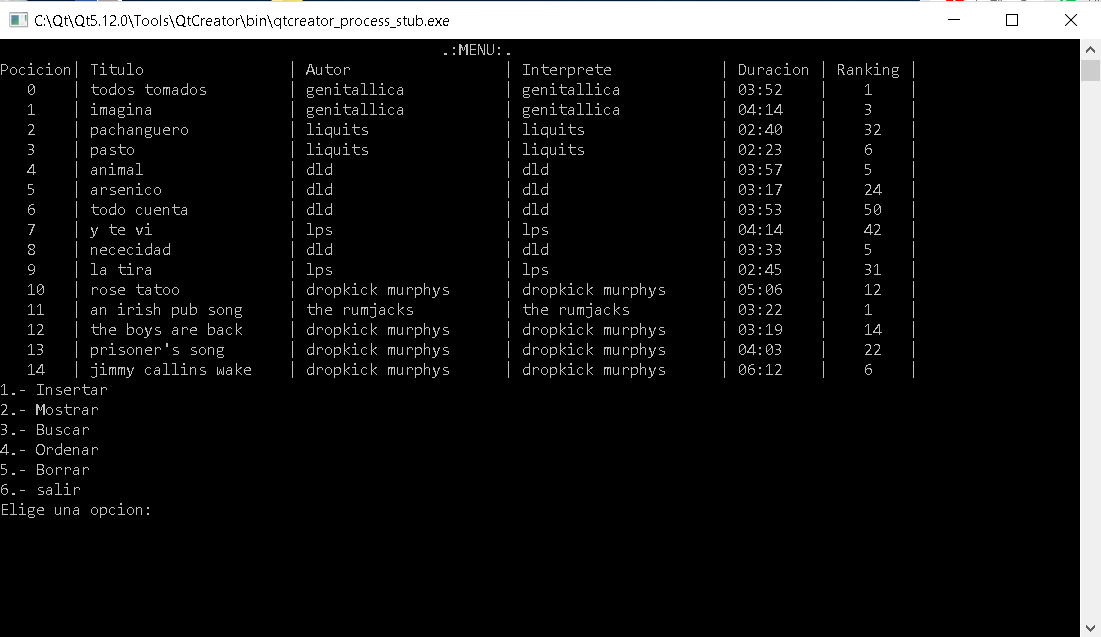
|  |
| --- |
| **Songs.cpp** |
| **#include "songs.h"**  ***using* *namespace* std;**  **int Songs::getOrder() *const* {**  ***return* order;**  **}**  **void Songs::setOrder(*const* int &ord) {**  **order = ord;**  **}**  **Songs::Songs() : order(0) { }**  **Songs::Songs(*const* Songs &copy) : title(copy.title), 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>**  ***template* <*typename* Type>**  ***class* List {**  ***private*:**  ***static* *const* int SIZE = 3000;**  **Type data[SIZE];**  **size\_t counter;**  **bool validPosition(*const* int &);**  **void mergeSort(*const* int &left, *const* int &right);**  **void quickSort(*const* int &left, *const* int &right);**  ***public*:**  **List();**  **Type &*operator*[](int &);**  **bool empty();*/\*revisa* *si* *esta* *vacia\*/***  **bool full();*/\*revisa* *si* *esta* *llena\*/***  **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\*/***  **void sortData(Type &, Type &);**  **void append(*const* Type &);*/\*elemento* *a* *insertar* *al* *final* *de* *la* *lista\*/***  **void append(*const* Type &, *const* int &);*/\*inserta* *en* *una* *posicion* *exacta\*/***  **void erase(int &);*/\*borra* *un* *dato* *de* *la* *lista\*/***  **void remove();*/\*elimina* *toda* *la* *lista\*/***  **int findDataL(*const* Type &);*/\*busqueda* *lineal\*/***  **int findDataB(*const* Type &);*/\*busqueda* *binario\*/***  **void mergeSort();*/\*meotodo* *de* *ordenamiento\*/***  **void quickSort();*/\*meotodo* *de* *ordenamiento\*/* */\*revisar\*/***  **Type show(*const* int &);*/\*retorna* *el* *elemento* *para* *poder* *mostrarlo\*/***  **void showAll();*/\*retorna* *el* *elemento* *para* *poder* *mostrarlo\*/***  **void bubbleSort();*//Burbuja* *Mejorada***  **void shellSort();*//shell***  **void insertionSort();*//Insersion***  **void selectSort();*//Seleccion***  **};**  **#endif *//* *LIST\_H*** |

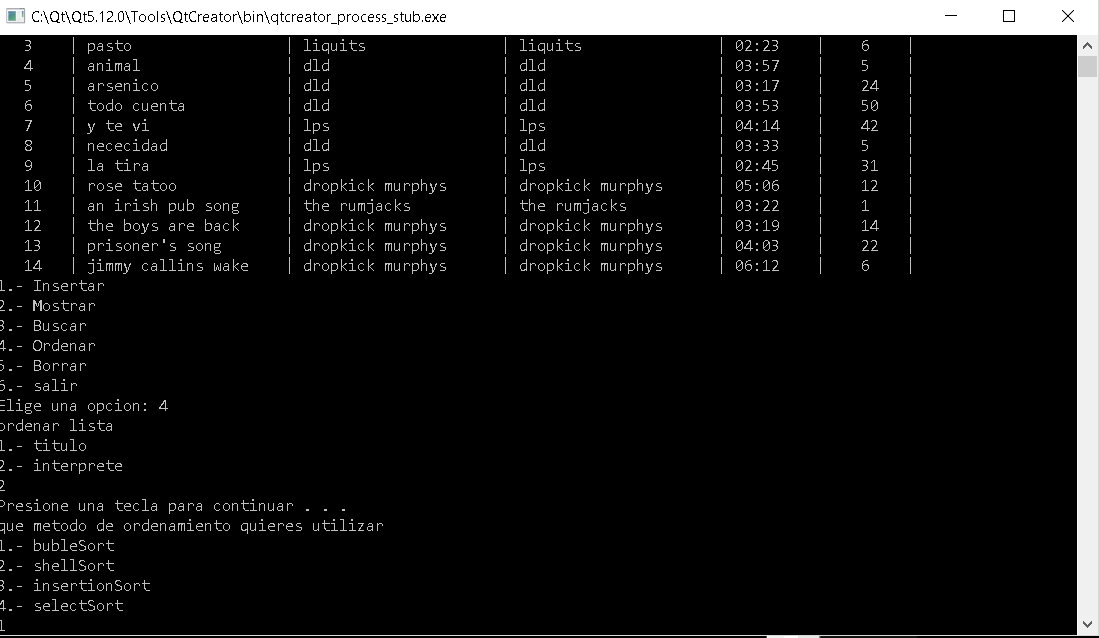
|  |
| --- |
| **List.cpp** |
| **#include "list.h"**  **#include <stdexcept>**  ***template*<*typename* Type>**  **List<Type>::List() : counter(0) { }**  ***template*<*typename* Type>**  **Type &List<Type>::*operator*[](int &e) {**  ***if*(empty()) {**  **std::cout << "empty list";**  **}**  ***if*(e>=counter) {**  **std::cout << "invalid position ";**  **}**  ***return* data[e];**  **}**  ***template*<*typename* Type>**  **bool List<Type>::validPosition(*const* int &position) {**  ***if*(position >= counter *or* position < 0) {**  ***return* *false*;**  **}**  ***return* *true*;**  **}**  ***template*<*typename* Type>**  **void List<Type>::mergeSort() {**  **mergeSort(0, counter);**  **}**  ***template*<*typename* Type>**  **void List<Type>::mergeSort(*const* int &left, *const* int &right) {**  ***if*(left >= right) {**  ***/\*criterio* *de* *paro* *\*/***  ***return*;**  **}**  ***/\*copia* *a* *temporal\*/***  **Type temp[SIZE];**  ***for* (int z(left); z <= right; z++) {**  **temp[z] = data[z];**  **}**  **int m((left+right)/2);**  **mergeSort(left, m);**  **mergeSort(m+1, right);**  **int i(left), j(m+1), x(left);**  ***while* (i <= m *and* j <= right) {**  ***while* (i <= m *and* temp[i] <= temp[j]) {**  **data[x++] = temp[i++];**  **}**  ***if*(i <= m) {**  ***while* (j <= right *and* temp[j] <= temp[i]) {**  **data[x++] = temp[j++];**  **}**  **}**  **}**  ***while* (i <= m) {**  **data[x++] = temp[i++];**  **}**  ***while* (j <= right) {**  **data[x++] = temp[j++];**  **}**  **}**  ***template*<*typename* Type>**  **void List<Type>::quickSort() {**  **quickSort(0, counter);**  **}**  ***template*<*typename* Type>**  **void List<Type>::quickSort(*const* int &left, *const* int &right) {**  ***if*(left >= right) {**  ***/\*criterio* *de* *paro* *\*/***  ***return*;**  **}**  **int i(left),j(right);**  ***while* (i < j) {**  ***while* (i < j *and* data[i] <= data[right]) {**  **i++;**  **}**  ***while* (i < j *and* data[j] >= data[right]) {**  **j--;**  **}**  ***if*(i != j) {**  **sortData(*data[i]*, *data[j]*);**  **}**  **}**  ***if*(i != right) {**  **sortData(*data[i]*, *data[right]*);**  **}**  **quickSort(left, i-1);**  **quickSort(i+1, right);**  **}**  ***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>::append(*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>::append(*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*<*class* Type>**  **void List<Type>::sortData(Type &a, Type &b) {**  **Type aux(a);**  **a = b;**  **b = aux;**  **}**  ***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* <*class* Type>**  **int List<Type>::findDataL(*const* Type &e) {**  ***for* (size\_t i = 0; i <= counter; i++) {**  ***if*(data[i] == e) {**  ***return* i;**  **}**  **}**  ***return* -1;**  **}**  ***template* <*class* Type>**  **int List<Type>::findDataB(*const* Type &e) {**  **int i(0), j(counter), 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;**  **}**  ***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* {**  **std::cout << " " << position << data[position];**  ***return* data[position];**  **}**  ***return* data[0];**  **}**  ***template*<*typename* Type>**  **void List<Type>::showAll() {**  ***if*(empty()){**  **std::cout << "la lista esta vacia" << std::endl;**  **} *else* {**  ***for* (int i(0); i < counter; i++) {**  **std::cout << " " << i << data[i];**  **}**  **}**  **}**  ***template*<*typename* Type>**  **void List<Type>::bubbleSort() {**  **int band,i,j;**  **i = counter-1;**  ***do* {**  **band=0;**  **j=0;**  ***while*(j < i) {**  ***if*(data[j] > data[j+1]) {**  **sortData(*data[j]*, *data[j+*1*]*);**  **band=1;**  **}**  **j++;**  **}**  **i--;**  **}*while*(band==1);**  **}**  ***template*<*typename* Type>**  **void List<Type>::shellSort() {**  **int dif, i = 0;**  **float fact = 0.75;**  **dif=(counter-1)\*fact;**  ***while*(dif>0) {**  ***while*(i<counter-1-dif) {**  ***if*(data[i] > data[i+dif]) {**  **sortData(*data[i+dif]*, *data[i]*);**  **}**  **i++;**  **}**  **dif\*=fact;**  **}**  **}**  ***template*<*typename* Type>**  **void List<Type>::insertionSort() {**  **int i = 1 ,j;**  **Type aux;**  ***while*(i < counter){**  **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++;**  **}**  **}**  ***template*<*typename* Type>**  **void List<Type>::selectSort() {**  **int i,j,menor;**  **i=0;**  ***while*(i<counter-1){**  **menor=i;**  **j=i+1;**  ***while*(j<counter){**  ***if*(data[j] < data[menor])**  **menor=j;**  **j++;**  **}**  ***if*(menor!=i){**  **sortData(*data[i]*, *data[menor]*);**  **}**  **i++;**  **}**  **}**  ***template*<*typename* Type>**  **void List<Type>::remove() {**  **counter = 0;**  **}** |

|  |
| --- |
| **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*** |

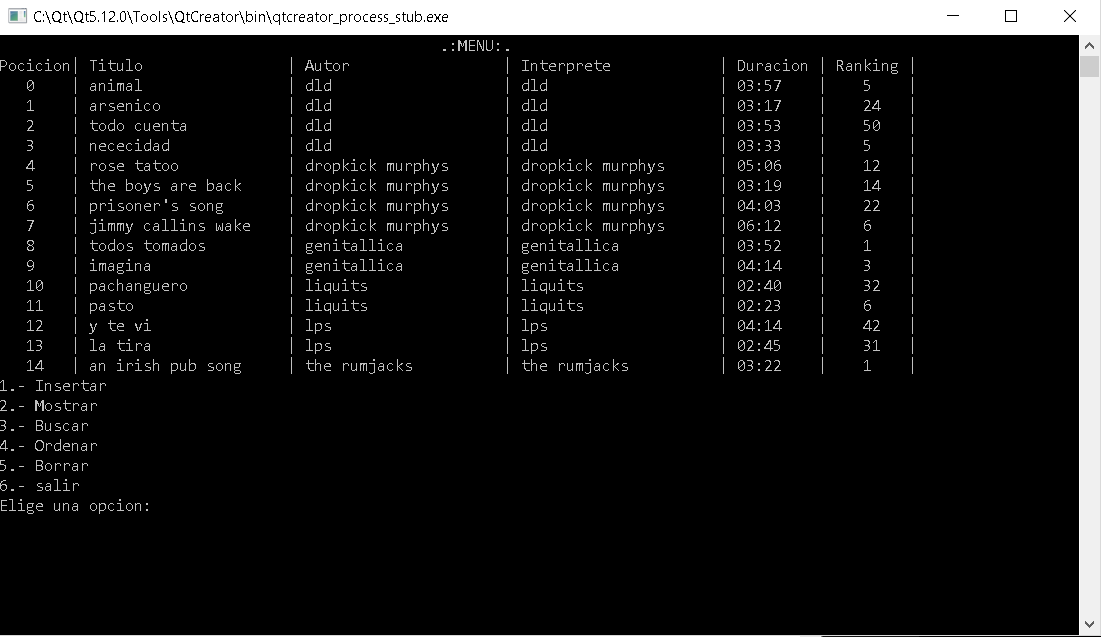
|  |
| --- |
| **CAPTURAS DE PANTALLA** |



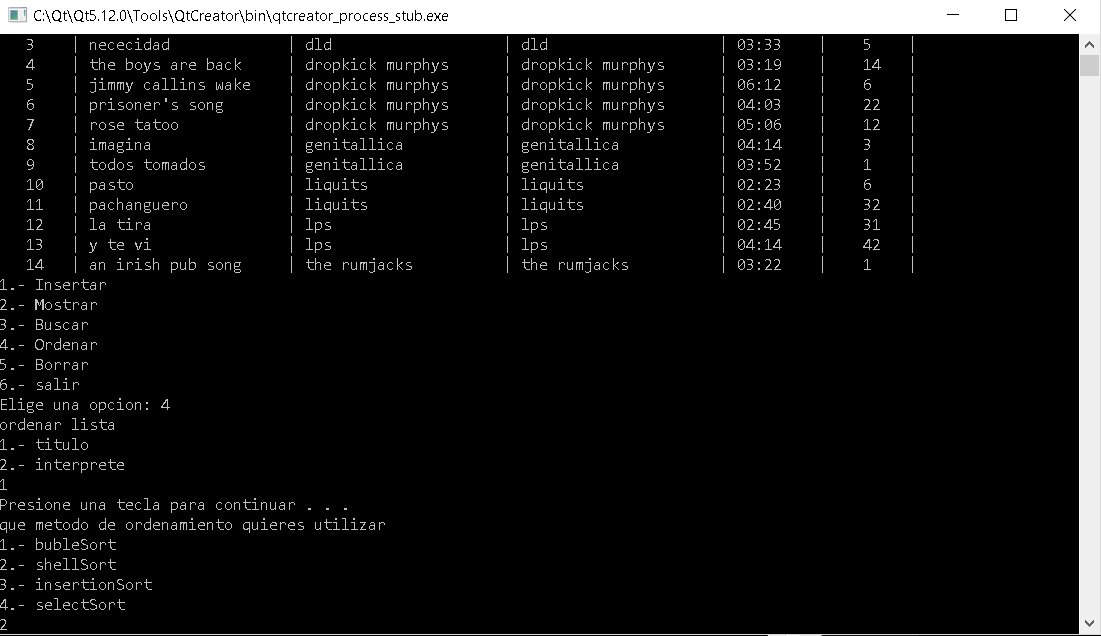
Lista de canciones



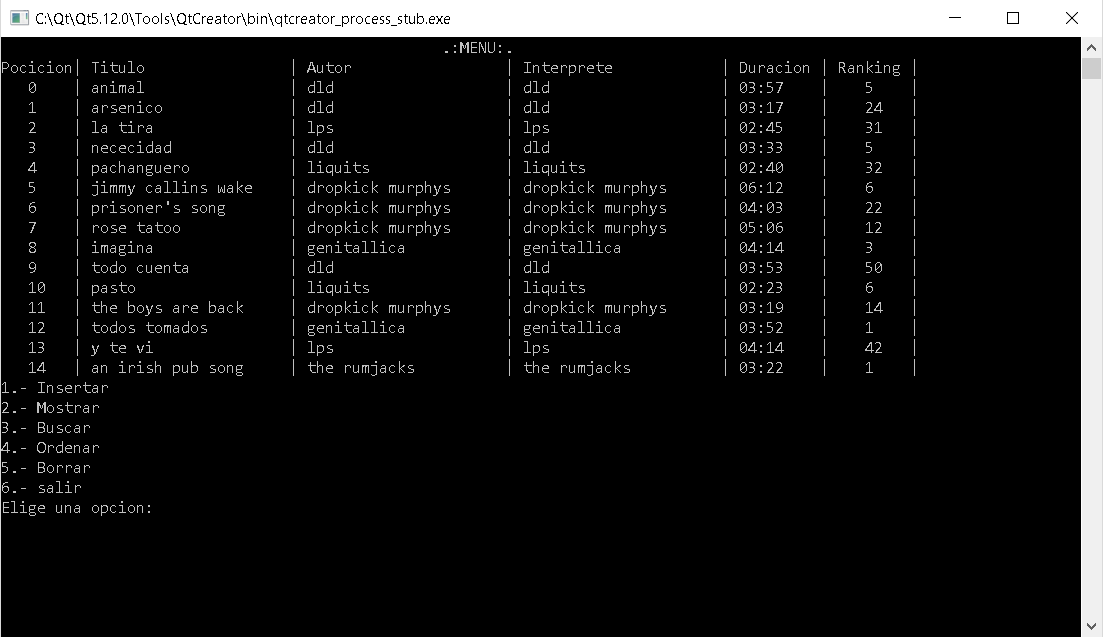
Ordenando por el interprete y con método burbuja



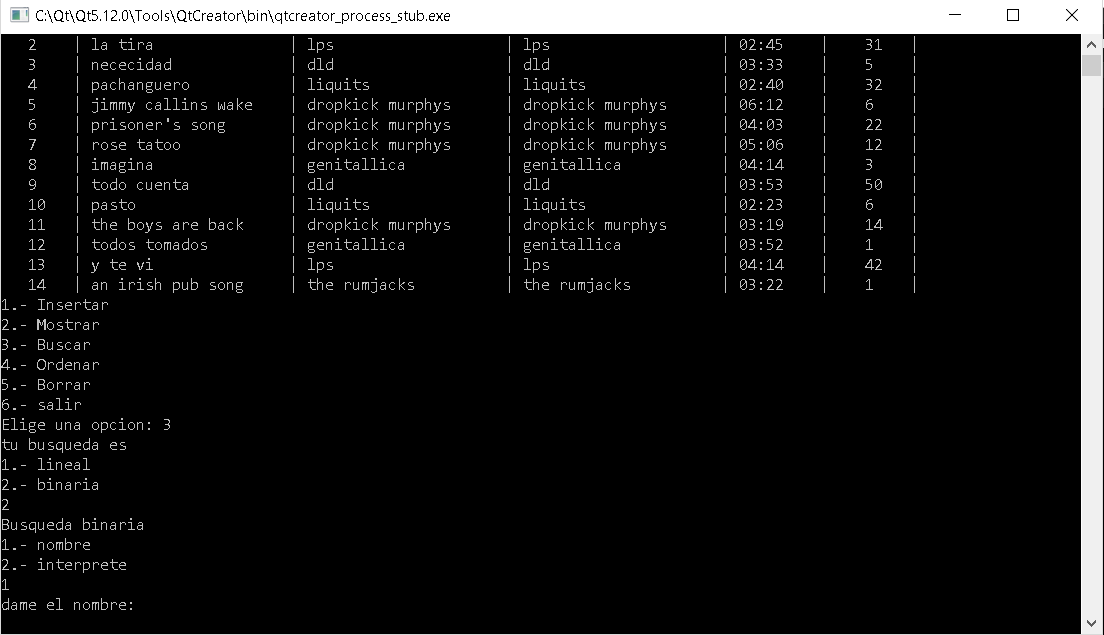
Aquí vemos la lista completamente ordena



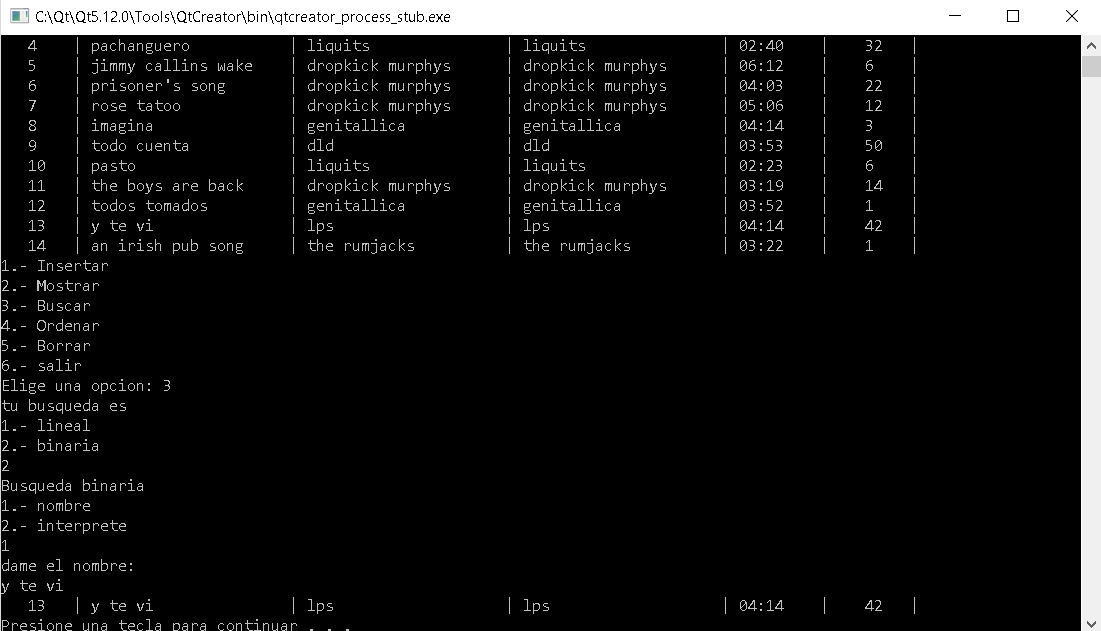
Aquí ordenamos por titulo



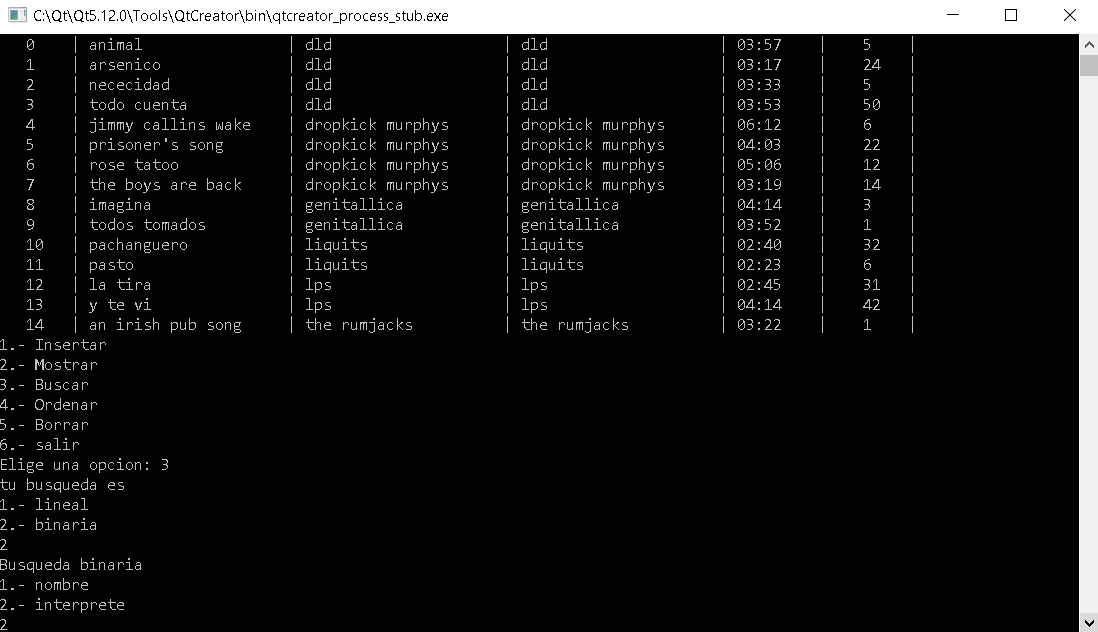
Vemos que la lita ya esta ordenda



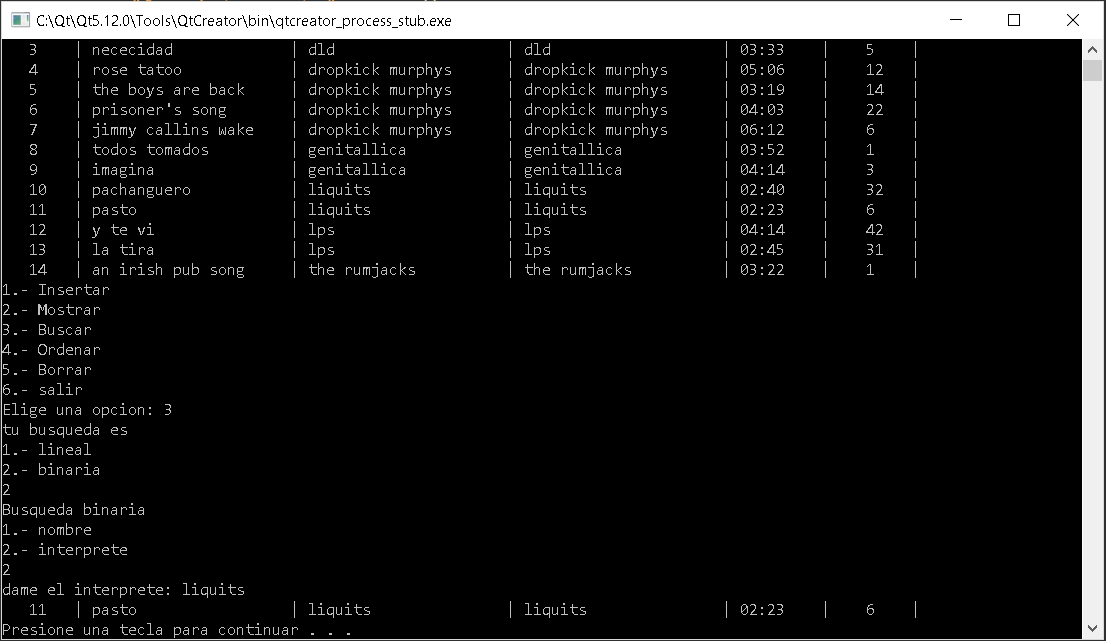
Aremos una búsqueda binaria ya que están ordenados y será mediante el nombre, ya que es como esta ordendado



Aquí nos damos cuenta de que si funciona, nos imprime todos los datos de la canción



Ahora aremos una búsqueda por inteprete



Aquí vemos que si encuentra el dato, me retorna una de las canciones