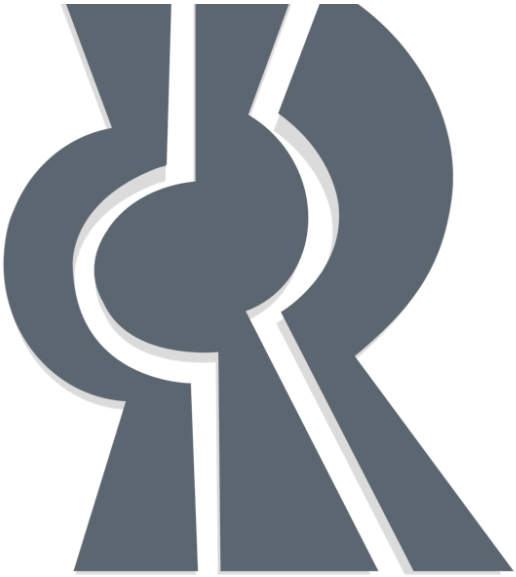
30-3-2019

LA LISTA, IMPLEMENTACIÓN DINÁMICA DOBLEMENTE LIGADA



CUCEI

david gutierrez alvarez

Estructura de datos I

RESUMEN PERSONAL Y FORMA DE ABORDAR EL PROBLEMA

Main.cpp

```cpp
#include <iostream>
#include "menu.h"

using namespace std;

int main() {
        Menu menu;

        return 0;
}
```

Menu.h

```cpp
#ifndef MENU_H
#define MENU_H

#include "list.h"
#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,
                optionErase,
                optionOut
        };
};

#endif // MENU_H
```

Menu.cpp

```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;
                    cout << songs.toString();
                }
                cout << optionAdd << ".- Insertar" << 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;
                        cout << endl << "Pocicion| Titulo\t\t| Autor\t\t\t|
Interprete\t\t| Duracion | Ranking |" << endl;
                        system("pause");
                        break;

                        case optionFind: findL();
                        break;

                        case optionErase: erase();
                        break;

                        case optionOut:
                        break;

                        default:
                        cout << "valor invalido";
                }
        } 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(songs.getFirst(), 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(songs.getFirst(), newSong);
                        option = "0";
                } else if(option == "2") {
                        songs.insert(songs.getNext(songs.getFirst()), newSong);
                        option = "0";
                } else {
                        cout << "Opcion invalida" << endl;
                }
        } while(option != "0");
}

void Menu::erase() {
        if(songs.empty()) {
                cout << "La lista esta vacia" << endl;
        } else {
                string position;
                cout << "Ingresa la cancion a eliminar:";
                getline(cin, position);
                cin.ignore();
                songs.erase(songs.find(song));
        }

}

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;
```

```
    }
    cout << songs.retrieve(songs.find(song));
    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;

        std::string toString();

        //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

```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::toString() {
    Cursor cursor;
    string line;

    cursor.Gotoxy(8, cursor.wherey());
    line += "| ";
    line += getTitle();
    cursor.Gotoxy(32, cursor.wherey());
    line += "| ";
    line += getAuthor();
    cursor.Gotoxy(56, cursor.wherey());
    line += "| ";
    line += getInterprete();
    cursor.Gotoxy(80, cursor.wherey());
    line += "| ";
    line += getDuration();
    cursor.Gotoxy(91, cursor.wherey());
    line += "| ";
    cursor.Gotoxy(96, cursor.wherey());
    line += to_string(getRanking());
    cursor.Gotoxy(101, cursor.wherey());
    line += "| ";
    return  line;
}

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 {
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(); }

    };

    class Node {
    private:
        Type data;
        Node *next;
        Node *prev;

    public:
        Node();
        Node(const Type &);

        Type &getData();
        Node *getNext() const;
        Node *getPrev() const;

        void setData(const Type &);
        void setNext(Node *);
        void setPrev(Node *);
    };

private:
    Node *anchor;

    bool validPos(Node*) const;
    void copyAll(const List &);

public:
    List();
    List(const List &);

    ~List();

    bool empty() const;

    void insert(Node *, const Type &);
    void erase(Node *);

    Node *getFirst() const;
    Node *getLast() const;
    Node *getPrev(Node *) const;
    Node *getNext(Node *) const;

    Node *find(const Type &) const;
    Type &retrieve(Node *);

    std::string toString() const;

    void deleteAll();

    List &operator = (const List &);
};

/// Implementacion

/// Node ///
template<typename Type>
List<Type>::Node::Node() : next(nullptr), prev(nullptr) { }

template<typename Type>
```

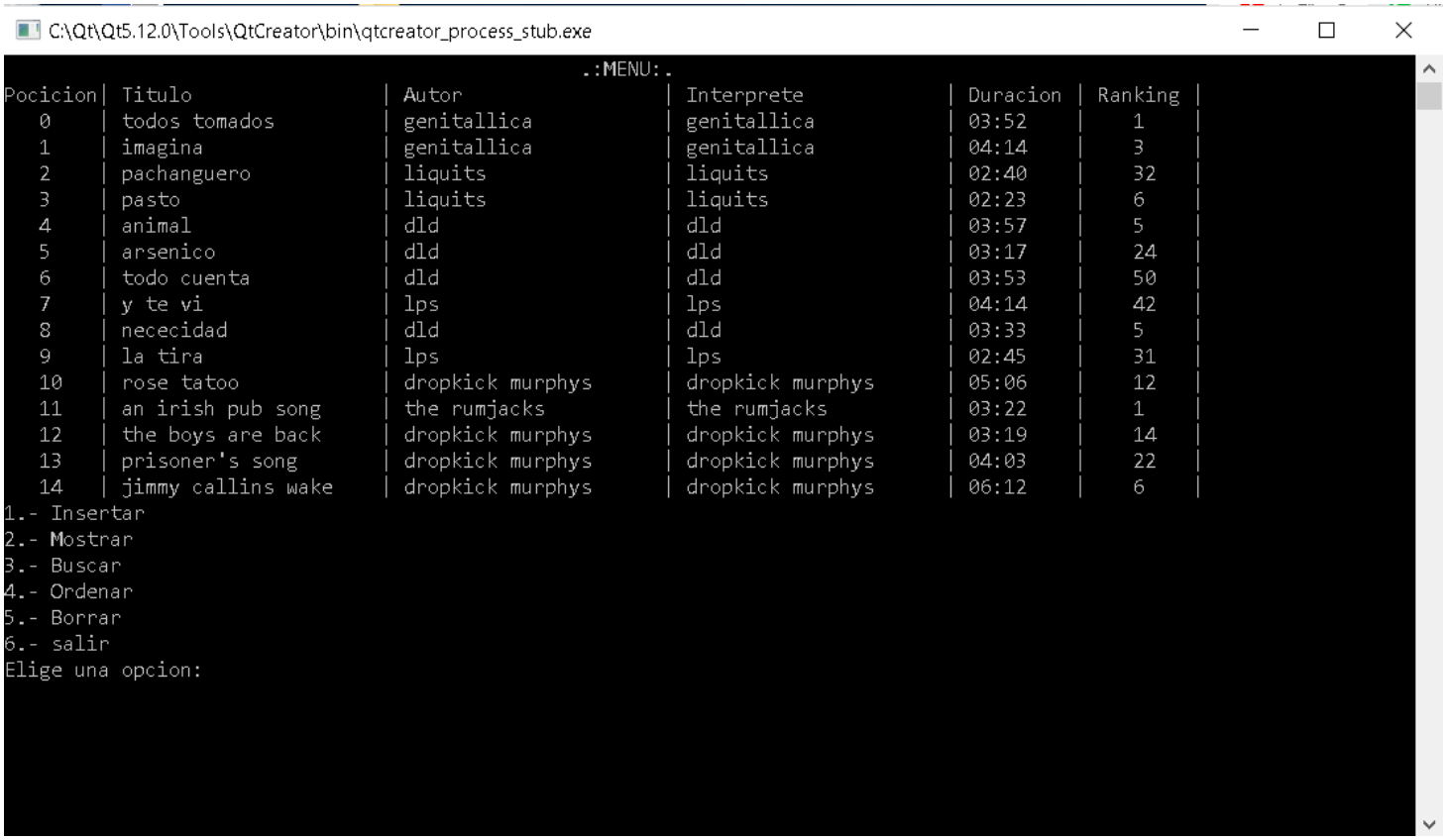
```
List<Type>::Node::Node(const Type &e) : data(e), next(nullptr), prev(nullptr) { }

template<typename Type>
Type &List<Type>::Node::getData() {
    return data;
}

template<typename Type>
typename List<Type>::Node* List<Type>::Node::getNext() const {
    return next;
}

template<typename Type>
typename List<Type>::Node* List<Type>::Node::getPrev() const {
    return prev;
}

template<typename Type>
void List<Type>::Node::setData(const Type &e) {
    data = e;
}

template<typename Type>
void List<Type>::Node::setNext(List<Type>::Node *p) {
    next = p;
}

template<typename Type>
void List<Type>::Node::setPrev(List<Type>::Node *p) {
    prev = p;
}

/// List ///
template<typename Type>
bool List<Type>::validPos(List<Type>::Node *p) const {
    if(empty()) {
        return  false;
    }

    Node * aux(anchor);

    do {
        if(aux == p) {
            return  true;
        }
        aux = aux->getNext();
    }while (aux != anchor);

    return false;
}

template<typename Type>
void List<Type>::copyAll(const List &l) {
    Node *aux(l.anchor);
    Node *last(nullptr);
    Node *newNode;

    do{
        newNode = new Node(aux->getData());

        if(newNode ==  nullptr) {
            throw List<Type>::Exception("Memoria no disponible, coplyAll");
        }

        if(last ==  nullptr) {
            anchor = newNode;
```

```
        } else {
            last->setNext(newNode);
            newNode->setPrev(last);
        }

        last = newNode;
        aux = aux->getNext();
    } while (aux != l.anchor);

    last->setNext(anchor);
    anchor->setPrev(last);
}

template<typename Type>
List<Type>::List() : anchor(nullptr) { }

template<typename Type>
List<Type>::List(const List &l) {
    copyAll(l);
}

template<typename Type>
List<Type>::~List() {
    deleteAll();
}

template<typename Type>
List<Type> &List<Type>::operator = (const List<Type> &l) {
    deleteAll();

    copyAll(l);

    return *this;
}

template<typename Type>
bool List<Type>::empty() const {
    return anchor == nullptr;
}

template<typename Type>
void List<Type>::insert(List<Type>::Node *p, const Type &e) {
    if(p != nullptr and !validPos(p)) {
        throw  Exception("posicion invalida, insert");
    }

    Node *aux(new  Node(e));

    if(aux == nullptr) {
        throw Exception("memoria no disponible, insert");
    }

    if(p == nullptr) { // inserta al principio
        if(empty()) { // insertar el primer elemento
            aux->setPrev(aux);
            aux->setNext(aux);
        } else { // no es el primer elemeneto
            aux->setPrev(getLast());
            aux->setNext(anchor);
            getLast()->setNext(aux);//
            anchor->setPrev(aux);
        }

        anchor = aux;
    } else { // insertar en otra posicion
        aux->setPrev(p);
```

```
        aux->setNext(p->getNext());

        p->getNext()->setPrev(aux);
        p->getPrev()->setNext(aux);////
    }
}

template<typename Type>
void List<Type>::erase(List<Type>::Node *p) {
    if(!validPos(p)) {
        throw Exception("posicion invalida, erase");
    }
    p->getPrev()->setNext(p->getNext());
    p->getNext()->setPrev(p->getPrev());

    if(p == anchor) {//eliminando al primero
        if(p->getNext() == p) {
            anchor == nullptr;
        } else {
            anchor = anchor->getNext();
        }

    }
    delete p;
}

template<typename Type>
typename List<Type>::Node *List<Type>::getFirst() const {
    return  anchor;
}

template<typename Type>
typename List<Type>::Node *List<Type>::getLast() const {
    if(empty()) {
        return nullptr;
    }

    Node *aux(anchor);
    aux = aux->getPrev();
    return aux;
}

template<typename Type>
typename List<Type>::Node *List<Type>::getPrev(List<Type>::Node *p) const {

    if(p == anchor or !validPos(p)) {
        return  nullptr;
    }

    return p->getPrev();
}

template<typename Type>
typename List<Type>::Node *List<Type>::getNext(List<Type>::Node *p) const {
    if(!validPos(p) or p->getNext() == anchor) { // encapsulamiento
        return nullptr;
    }
    return p->getNext();
}

template<typename Type>
typename List<Type>::Node *List<Type>::find(const Type &e) const /**/{
    Node *aux(anchor);

    while (aux != nullptr and aux->getData() != e) {
        aux = aux->getNext();
```

```
    }
    return  aux;
}

template<typename Type>
Type &List<Type>::retrieve(List<Type>::Node *p) {
    if(!validPos(p)) {
        throw Exception("posicion invalida, retrieve");
    }
    return p->getData();
}

template<typename Type>
std::string List<Type>::toString() const {
    std::string result = "\n";

    if(!empty()){

        Node *aux(anchor);
        do {
            result += aux->getData().toString() + "\n";
            aux = aux->getNext();
        } while (aux != anchor);
    }
    return result;
}

template<typename Type>
void List<Type>::deleteAll() {
    if(empty()) {
        return;
    }

    Node *mark(anchor);
    Node *aux;

    do {
        aux = anchor;
        anchor = anchor->getNext();
        delete aux;
    } while (anchor != nullptr);//modify
}

#endif // LIST_H
```

CAPTURAS DE PANTALLA



Como vemos aquí, visiblemente el programa no tiene cambios