UNIVERSIDAD EUROPEA DE MADRID

ESCUELA DE ARQUITECTURA, INGENIERÍA Y DISEÑO

GRADO EN INGENIERÍA INFORMÁTICA

Proyecto para el uso de vectores

Santiago Moreno
Guillermo Aos
Ismael Bouatman
Javier Franco

ÍNDICE

1. Introducción

Se ha planteado la resolución de 2 ejercicios, más uno opcional, donde cada uno de los ejercicios tiene como finalidad aplicar todos los conocimientos adquiridos en clase durante la unidad formativa 3 mediante la utilización del ejercicio programado en clase VectorPEL.

1.1 Discográfica

En este ejercicio se necesita programar con la ayuda del Vector PEL un sistema de búsqueda de álbumes por parte de una discográfica, donde se podrá buscar mediante título del álbum, canción o tipo de música, en caso de no encontrarlo saltara un mensaje de no encontrado.

1.2 Videoclub

En este ejercicio se necesita programar con la ayuda del Vector PEL un sistema de alquileres de discos. El sistema debe permitir buscar discos mediante su título. El usuario podrá ver la información de los discos no alquilados y podrá devolver los discos alquilados.

2. Requisitos

- Uso del VectorPEL.
- Resolución de varios problemas.
- Aplicar los conocimientos adquiridos en UF3.

2.1 Requisitos Discográfica.

- Solicite al usuario por consola la búsqueda del álbum,género musical o grupo.
- Almacenar dichos valores de la discográfica mediante el uso del vector PEL.
- Menú con la gestión de la búsqueda.

2.2 Requisitos Videoclub

- Pedir al usuario el título de la película
- Almacenar la información de las películas
- Permitir alquilar y devolver los discos
- Visualizar sólo la información de las películas que no estén alquiladas

3. Explicación del código:

3.1 Explicación del código Discográfica

```cpp
#include <cstdlib>
#include <iostream>
#include <cmath>
#include <vector>
#include <ranges>
#include <algorithm>
#include <stdexcept>
#include <string>
#include<bits/stdc++.h>

using namespace std;

/*Una discográfica, recientemente ha decidido actualizar sus sistemas informáticos,
pasando a almacenar todos los álbumes con la información correspondiente al
titulo del álbum, grupo al que pertenece y género musical que los representa.
Dicho programa, debe tener un sistema de búsquedas, que muestre por consola
los resultados de buscar, ya sea el grupo, el título o el género. En caso de no
encontrarlo se mostrará un mensaje por pantalla que indique que no se ha podido
encontrar el álbum.
Para realizar dicho proyecto, los álbumes se almacenarán en un vector, y debe
manejarse las búsquedas mediante estos.*/

template<typename T> // T debe ser un tipo default_initializable
class Vector {
    T* v_,     // inicio del espacio de capacidad reservada para el vector
    * space_, // fin de la secuencia de elementos almacenados
    * last_;  // fin de la capacidad reservada para el vector
```

Empezaremos añadiendo con los includes todas las librerías necesarias para llevar a cabo este proyecto.
Luego está comentado el enunciado de nuestro problema el cual tenemos que resolver.
A continuación introducimos a nuestro código el vector DEL codificado en clase, este ya viene comentado en el código.

Como en cualquier plantilla luego se instanciarla las funciones en el public donde se encuentran tantos los constructores del vector como el destructor del mismo.
Mediante la programación de este vector se puede apreciar toda la funcionalidad interna que tiene un Vector.

```cpp
    void bounds_check(std::size_t i) const { if (size() < i) throw std::out_of_range("Error"); }
public:
    Vector() // construye un vector vacío
            : v_{new T[0]}, space_{v_}, last_{v_} { }

    Vector(Vector<T> const& v)
            : v_{new T[v.capacity()]}, space_{v_ + v.size()}, last_{v_ + v.capacity()}
    {
        try {
            for (auto i = size_t{0}; i < v.size(); ++i)
                v_[i] = v[i];
        }
        catch (...) {
            delete[] v_;
            throw;
        }
    }

    auto& operator=(Vector<T> const& rhs)
    {
        auto tmp = Vector<T>{rhs};
        std::swap(v_, tmp.v_);
        std::swap(space_, tmp.space_);
        std::swap(last_, tmp.last_);
        return *this;
    }

    ~Vector() { delete[] v_; } // destruye el array apuntado por v_
```

```
// funciones de capacidad----------------------------------------
auto size()     const -> std::size_t { return space_ - v_; }
auto capacity() const -> std::size_t { return last_ - v_; }
auto empty()    const -> bool        { return v_ == space_; }

// funciones de acceso-------------------------------------------

// sin bounds checking:
auto operator[](std::size_t i)       -> T&       { return v_[i]; }
auto operator[](std::size_t i) const -> T const& { return v_[i]; }
// con bounds checking:
auto at(std::size_t i)       -> T&       { bounds_check(i); return v_[i]; }
auto at(std::size_t i) const -> T const& { bounds_check(i); return v_[i]; }

auto begin()       -> T*       { return v_; }
auto begin() const -> T const* { return v_; }
auto end()         -> T*       { return space_; }
auto end()   const -> T const* { return space_; }
```

Aquí continuamos con el vector PEL donde se puede seguir viendo todo el funcionamiento del mismo para luego darle las funcionalidades que se le pueden dar al Vector en el lenguaje de programación c + +.

```
void push_back(T const& val)
{
    if (space_ == last_) { // capacidad agotada o primera vez que invocamos push_back
        std::size_t cp = capacity(),             // capacidad actual del vector
        new_cp = (cp == 0)? 2 : 2*cp; // nueva capacidad
        T* new_block = new T[new_cp]; // nuevo bloque de memoria
        try {
            for (auto i = std::size_t{}; i < cp; ++i)
                new_block[i] = v_[i];
            new_block[cp] = val;
        }
        catch (...) { // de lanzarse una excepción...
            delete[] new_block; //... destruimos el nuevo array...
            throw;              // ...y relanzamos la excepción
        }
        // destruimos el array original y reasignamos los punteros:
        delete[] v_;
        v_ = new_block;
        space_ = new_block + cp + 1;
        last_ = new_block + new_cp;
    }
    else {
        *space_ = val;
        ++space_;
    }
};
// resto de la interfaz pública...
```

Para terminar las funciones de nuestro vector PEL tenemos la función push back, esta función está a prueba de excepciones, esto con la ayuda de un try-catch. Esta es la función que más nos ayudará para poder resolver problemas en caso de que surjan.

```
void BuscarAlbum(vector <string>*cap1, vector<string>*album);
void BuscarGrupoMusical(vector <string>*cap1,vector<string>*album1);
void BuscarGeneroMusical(vector <string>*cap1, vector<string>*album2);
vector<string> unir(vector <string>*cap1, vector<string>*cap, vector<string>*album,vector <string>*album1,vector<string>*album2);
void mostrarCatalogo();
```

Instanciamos todas las funciones que tendrá nuestro código.

```
void BuscarGrupoMusical(vector <string>*cap1, vector<string>*album1){
    string usuario2;
    cout<<"Dime el grupo a buscar:";
    cin.ignore();
    getline( & cin, & usuario2);
    for (std::string const& grupoMusical : *cap1) {
        for(std::string const& grupoMusical1: *album1)
            if (grupoMusical == usuario2 && usuario2==grupoMusical1) {
                cout << "Encontrado" << endl;
                return;
            }
    }
    cout<<"No encontrado"<<endl;

};
```

Esta es la primera función de nuestro código que servirá para poder buscar por grupo musical, que recoge por parámetros el vector instanciado en el main, en el cual tendrá un for que recorre dicho vector y con el if comprobamos si lo que busca el usuario se encuentra o no en dicho vector. En caso de estar en el vector se mostrará por pantalla un mensaje de "encontrado" y en caso de no encontrarlo se mostrará un mensaje de "no encontrado".

```
void BuscarGeneroMusical(vector <string>*cap1, vector<string>*album2) {

    string usuario3;
    cout<<"Dime el genero a buscar:";
    cin.ignore();
    getline( & cin, & usuario3);
    for (std::string const& generoMusical: *cap1) {
        for(std::string const& generoMusical1: *album2)
            if (generoMusical == usuario3 && usuario3==generoMusical1) {
                cout << "Encontrado" << endl;
                return;
            }
    }

    cout << "No encontrado" << endl;


};
```

```
void BuscarAlbum(vector <string>*cap1, vector<string>*album){
    string usuario1;
    cout<<"Dime el album a buscar:";
    cin.ignore();
    getline( & cin, & usuario1);

    for (std::string const& Tipodealbum : *cap1 ) {
        for(std::string const& Tipodealbum1: *album)
            if (Tipodealbum == usuario1 && Tipodealbum1==usuario1) {
                cout << "Encontrado" << endl;
                return;
            }
    }

    cout<<"no encontrado"<<endl;


};
```

Las dos siguientes funciones son la de buscar por género musical y buscar por álbum que hacen exactamente lo mismo que la función anterior que buscaba por grupo musical.

```
void mostrarCatalogo(){
    cout<<"Los albumes de la discografica son:  ";
    cout<<"Abbey road, Sticky Fingers, Entre el cielo y el suelo, Black in Black y Made in Heaven"<<endl;
    cout<<"Los grupos de muscia son: ";
    cout<<"The Beatles, The Rolling Stones, Mecano, AC/DC y Queen"<<endl;
    cout<<"Los generos musicales son: ";
    cout<<"Pop Rock, Rock and Roll, Pop, Hard Rock y Rock "<<endl;
};
```

Por último pero por ello menos importante , la última función de nuestro código que con la ayuda de los cout enseñaremos por pantalla todo el catálogo de la discográfica al usuario.

```
int main(){
    auto *cap = new vector <std::string>{ n: 10};
    auto *cap1 = new vector <std::string>{ n: 15};
    auto *album1 = new vector <std::string>{};
    album1->push_back("The Beatles");
    album1->push_back("The Rolling Stones");
    album1->push_back("Mecano");
    album1->push_back("AC/DC");
    album1->push_back("Queen");
    auto *album2 = new vector <std::string>{};
    album2->push_back("Pop Rock");
    album2->push_back("Rock and Roll");
    album2->push_back("Pop");
    album2->push_back("Hard Rock");
    album2->push_back("Rock");
    auto *album = new vector <std::string>{};
    album->push_back("Abbey Road");
    album->push_back("Sticky Fingers");
    album->push_back("Entre el cielo y el suelo");
    album->push_back("Back in Black");
    album->push_back("Made in Heaven");
```

Ya en el main como se explicó anteriormente se instalan los vectores que almacenan toda la información de nuestra discográfica, estos estarán en el free store.

```
vector <string> unir(vector <string>*cap1, vector<string>*cap, vector<string>*album,vector <string>*album1,vector<string>*album2){

    merge( first1: album->begin(),  last1: album->end(),  first2: album1->begin(),  last2: album1->end(),  result: cap->begin());
    merge( first1: album2->begin(),  last1: album2->end(),  first2: cap->begin(),  last2: cap->end(),  result: cap1->begin());
    return *cap1;
};
```

En el main está instanciado una función llamada unir que servirá para unificar todos los vectores anteriormente almacenados en uno solo, con esto lograremos cumplir todos los requisitos de la aplicación.

```
int choose;
do{ cout<<"-----------------------------------------"<<endl;
    cout<<"Elige una opcion:"<<endl;
    cout<<"-----------------------------------------"<<endl;
    cout<<"Opcion 1-Buscar por album"<<endl;
    cout<<"Opcion 2-Buscar por grupo"<< endl;
    cout<<"Opcion 3-Buscar por genero"<< endl;
    cout<<"Opcion 4-Mostrar Catalogo"<< endl;
    cout<<"Opcion 5-Salir"<<endl;
    cout<<"-----------------------------------------"<<endl;
    cin>>choose;
    switch (choose){
        case 1:
            cout<<"Entrando en la opcion 1(Buscar por album):"<<endl;
            unir(cap1, cap, album, album1, album2);
            BuscarAlbum(cap1, album);
            break;
        case 2:
            cout<<"Entrando en la opcion 2(Buscar por grupo):"<<endl;
            unir(cap1, cap, album, album1, album2);
            BuscarGrupoMusical(cap1, album1);
            break;
        case 3:
            cout<<"Entrando en la opcion 3(Buscar por genero):"<<endl;
            unir(cap1, cap, album, album1, album2);
            BuscarGeneroMusical(cap1, album2);
            break;
        case 4:
            cout<<"Mostrando Catalogo"<<endl;
            mostrarCatalogo();
           break;
       default: cout<< "Opcion no valida"<<endl;
           exit( Code: -1);
    };
}
while(choose!=5);
```

Aquí se encuentra el menú que servirá para que el usuario interactúe con la aplicación, consta de 5 opciones donde las 4 primeras servirán serán las funciones de nuestro código que son buscar tanto por género musical, por album y por grupo musical, la cuarta opción de nuestro menú servirá para que el usuario pueda ver el catálogo de la discográfica. Ya la última opción se encuentra la opción para poder salir de la aplicación.

3.2 Explicación del código Videoclub

```cpp
#include <cstdlib>
#include <iostream>
#include <cmath>
#include <vector>
#include <ranges>
#include <algorithm>
#include <stdexcept>
#include <string>
#include<bits/stdc++.h>

using namespace std;

/*Una discográfica, recientemente ha decidido actualizar sus sistemas informáticos,
pasando a almacenar todos los álbumes con la información correspondiente al
titulo del álbum, grupo al que pertenece y género musical que los representa.
Dicho programa, debe tener un sistema de búsquedas, que muestre por consola
los resultados de buscar, ya sea el grupo, el título o el género. En caso de no
encontrarlo se mostrará un mensaje por pantalla que indique que no se ha podido
encontrar el álbum.
Para realizar dicho proyecto, los álbumes se almacenarán en un vector, y debe
manejarse las búsquedas mediante estos.*/

template<typename T> // T debe ser un tipo default_initializable
class Vector {
    T* v_,     // inicio del espacio de capacidad reservada para el vector
    * space_, // fin de la secuencia de elementos almacenados
    * last_;  // fin de la capacidad reservada para el vector
```

Empezaremos añadiendo con los includes todas las librerías necesarias para llevar a cabo este proyecto.
Luego está comentado el enunciado de nuestro problema el cual tenemos que resolver.
A continuación introducimos a nuestro código el vector DEL codificado en clase, este ya viene comentado en el código.

Como en cualquier plantilla luego se instanciarla las funciones en el public donde se encuentran tantos los constructores del vector como el destructor del mismo.
Mediante la programación de este vector se puede apreciar toda la funcionalidad interna que tiene un Vector.

```cpp
    void bounds_check(std::size_t i) const { if (size() < i) throw std::out_of_range("Error"); }
public:
    Vector() // construye un vector vacío
            : v_{new T[0]}, space_{v_}, last_{v_} { }

    Vector(Vector<T> const& v)
            : v_{new T[v.capacity()]}, space_{v_ + v.size()}, last_{v_ + v.capacity()}
    {
        try {
            for (auto i = size_t{0}; i < v.size(); ++i)
                v_[i] = v[i];
        }
        catch (...) {
            delete[] v_;
            throw;
        }
    }

    auto& operator=(Vector<T> const& rhs)
    {
        auto tmp = Vector<T>{rhs};
        std::swap(v_, tmp.v_);
        std::swap(space_, tmp.space_);
        std::swap(last_, tmp.last_);
        return *this;
    }

    ~Vector() { delete[] v_; } // destruye el array apuntado por v_
```

```
// funciones de capacidad-----------------------------------------
auto size()     const -> std::size_t { return space_ - v_; }
auto capacity() const -> std::size_t { return last_ - v_; }
auto empty()    const -> bool        { return v_ == space_; }

// funciones de acceso--------------------------------------------

// sin bounds checking:
auto operator[](std::size_t i)       -> T&       { return v_[i]; }
auto operator[](std::size_t i) const -> T const& { return v_[i]; }
// con bounds checking:
auto at(std::size_t i)       -> T&       { bounds_check(i); return v_[i]; }
auto at(std::size_t i) const -> T const& { bounds_check(i); return v_[i]; }

auto begin()       -> T*       { return v_; }
auto begin() const -> T const* { return v_; }
auto end()         -> T*       { return space_; }
auto end()   const -> T const* { return space_; }
```

Aquí continuamos con el vector PEL donde se puede seguir viendo todo el funcionamiento del mismo para luego darle las funcionalidades que se le pueden dar al Vector en el lenguaje de programación c + +.

```
void push_back(T const& val)
{
    if (space_ == last_) { // capacidad agotada o primera vez que invocamos push_back
        std::size_t cp = capacity(),             // capacidad actual del vector
        new_cp = (cp == 0)? 2 : 2*cp; // nueva capacidad
        T* new_block = new T[new_cp]; // nuevo bloque de memoria
        try {
            for (auto i = std::size_t{}; i < cp; ++i)
                new_block[i] = v_[i];
            new_block[cp] = val;
        }
        catch (...) { // de lanzarse una excepción...
            delete[] new_block; //... destruimos el nuevo array...
            throw;              // ...y relanzamos la excepción
        }
        // destruimos el array original y reasignamos los punteros:
        delete[] v_;
        v_ = new_block;
        space_ = new_block + cp + 1;
        last_ = new_block + new_cp;
    }
    else {
        *space_ = val;
        ++space_;
    }
};
// resto de la interfaz pública...
```

Para terminar las funciones de nuestro vector PEL tenemos la función push back, esta función está a prueba de excepciones, esto con la ayuda de un try-catch. Esta es la función que más nos ayudará para poder resolver problemas en caso de que surjan.

Aquí tenemos la clase disco que va a contener todas las propiedades de los discos,(Tipo, nombre de la película,el precio…). Ponemos los datos privados y declaramos los getters y setters para poder acceder y manipularlos.

```
class Disco{
private:
    int precio;
    string nombre_peli;
    bool alquilado;
    string tipo;

public:
    int getPrecio(){
        return precio;
    }
    string getNomre_peli(){
        return nombre_peli;
    }
    bool getAlquilado(){
        return alquilado;
    }
    string getTipo(){
        return tipo;
    }


    void setPrecio(int p){
        precio=p;
    }
    void setNombre_pel(string n){
        nombre_peli=n;
    }
    void setAlquilado(bool a){
        alquilado=a;
    }
    void setTipo(string t){
        tipo=t;
    }
```

Esto es el constructor de la clase y más abajo nos encontramos con las instalación de las tres funciones del programa.

```
    Disco(int precio, string nombre_peli,bool alquilado,string tipo){
        this->precio=precio;
        this->nombre_peli=nombre_peli;
        this->alquilado=alquilado;
        this->tipo=tipo;
    }


};

void buscar(vector<Disco> listaDiscos);
bool alquilado(vector<Disco>& listaDiscos);
bool devolver(vector<Disco>& listaDiscos);
```

Aquí damos valor a las variables y metemos los constructores dentro del vector tipo Disco, que es el nombre de la clase.

```
Disco disco1(10,"Titanic",false,"dvd");
Disco disco2(20,"Fast and Furious",true,"dvd");
Disco disco3 (15,"Alien",false,"dvd");
Disco disco4 (24,"Paco", true,"bluray");

vector<Disco> listaDiscos;
listaDiscos.push_back(disco1);
listaDiscos.push_back(disco2);
listaDiscos.push_back(disco3);
listaDiscos.push_back(disco4);
```

Para el menú que vamos a mostrar al usuario hemos decidido utilizar un switch anidado a un do while() para que después de cada vez que el usuario introduzca una opción se le vuelva a mostrar el menú. La opción 1 es buscar películas mediante el título de esta, la opción 2 corresponde a alquilar las películas y la tres a devolverlas.

```
    int choose;
    do{ cout<<"-----------------------------------------"<<endl;
        cout<<"Elige una opcion:"<<endl;
        cout<<"-----------------------------------------"<<endl;
        cout<<"Opcion 1-Buscar pelicula" <<endl;
        cout<<"Opcion 2-Alquilar"<< endl;
        cout<<"Opcion 3-Devolver"<< endl;
        cout<<"Opcion 4-Salir"<<endl;
        cout<<"-----------------------------------------"<<endl;
        cin>>choose;
        switch (choose){
            case 1:
                buscar(listaDiscos);
                break;
            case 2:
                cout<<"Entrando en la opcion 2(Alquilar):"<<endl;
                alquilado(listaDiscos);
                break;
            case 3:
                cout<<"Entrando en la opcion 3(Devolver):"<<endl;
                devolver(listaDiscos);
                break;
            default: "Opcion no valida";
                exit(-1);
        };
    }
    while(choose!=4);
};
```

Esta es la función buscar al principio le preguntamos por el nombre de la película, mediante el for recordemos el vector y con el primer if comprobamos si el nombre que nos introduce el usuario coincide con las pelis que hay. Siguiendo con la función encontramos un segundo if ya que el enunciado resaltaba que solo había que mostrar la información de las películas que no están alquiladas. Esto lo conseguimos gracias al if que comprueba que la peli está alquilada o no mediante un booleano. Incluimos como parámetro el Vector<Disco> para poder utilizarlo.

```
void buscar(vector<Disco> listaDiscos){
    string nombre_peli;
    cout<<"Introduce el nombre de la pelicula que quieres buscar: "<<endl;
    cin>>nombre_peli;
    for (int i = 0; i < listaDiscos.size(); ++i) {
        if (listaDiscos[i].getNomre_peli()==nombre_peli){
            if(listaDiscos[i].getAlquilado()==true){
                cout<<"La pelicula esta alquilada"<<endl;
            }else{
                cout<<"La pelicula "<<nombre_peli<<endl;
                cout<<"El precio es: "<<listaDiscos[i].getPrecio()<<endl;
                cout<<"El tipo es: "<<listaDiscos[i].getTipo()<<endl;
                cout<<"La pelicula no esta alquilada"<<endl;
            }
        }
    }
}
```

```
bool alquilado(vector<Disco>& listaDiscos) {
    int n;
    for (int i = 0; i < listaDiscos.size(); i++) {
        if (listaDiscos[i].getAlquilado() == false) {
            cout << i<<": "<<listaDiscos[i].getNomre_peli() <<endl;
        }
    }

    cout<<"Indica el numero de peli para alquilar"<<endl;
    cin>>n;
    listaDiscos[n].setAlquilado( a: true);
};
```

En la función alquilado también tenemos como parámetro el Vector, Hacemos un for para recorrer el vector. Primero y lo más importante comprobamos que las pelis no están alquiladas y por consiguiente las imprimimos. Una vez imprimidas le preguntamos el número asignado a la peli introducimos el número ingresado por el usuario en el vector como posición y en esa posición cambiamos el parámetro boolean a true. Convirtiendo la peli de no alquilada a alquilada.

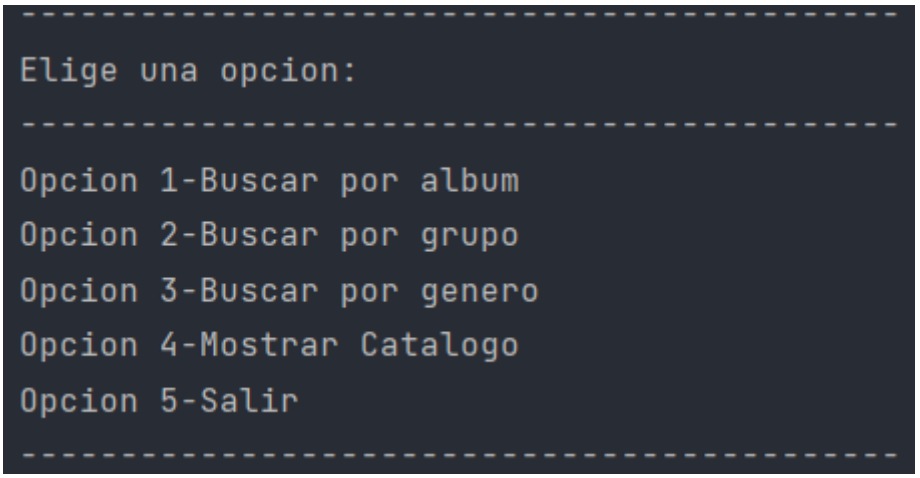
```
bool devolver(vector<Disco>& listaDiscos) {
    int n;
    for (int i = 0; i < listaDiscos.size(); i++) {
        if (listaDiscos[i].getAlquilado() == true) {
            cout << i << ": " << listaDiscos[i].getNomre_peli() << endl;
        }
    }
    cout << "Indica el numero de peli para devolver" << endl;
    cin >> n;
    listaDiscos[n].setAlquilado(false);
};
```

En la función devolver también tenemos como parámetro el Vector, Hacemos un for para recorrer el vector. Primero y lo más importante comprobamos que las pelis están alquiladas y por consiguiente las imprimimos. Una vez imprimidas le preguntamos el número asignado a la peli introducimos el número ingresado por el usuario en el vector como posición y en esa posición cambiamos el parámetro boolean a false. Convirtiendo la peli de alquilada a no alquilada. Como se puede ver a simple vista la función de alquilar y devolver son las mismas pero con valores antagónicos

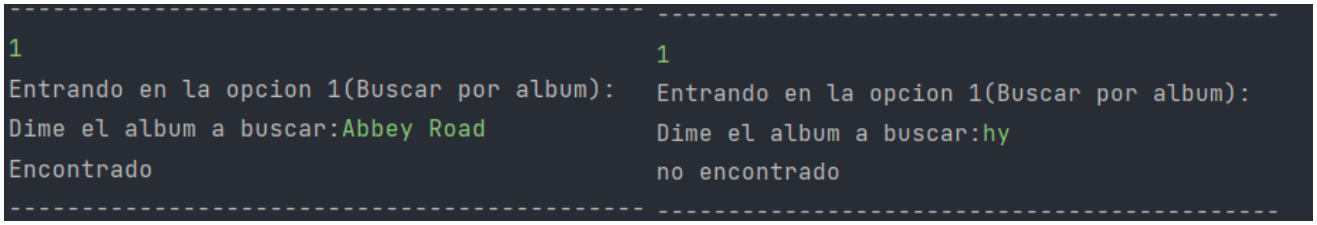
4. Manual de usuario

4.1 Manual de usuario Discográfica

Al iniciar nuestra aplicación aparecerá el siguiente menú.



Al iniciar la aplicación nos aparecerá un menú, donde podremos elegir la opción que queramos.

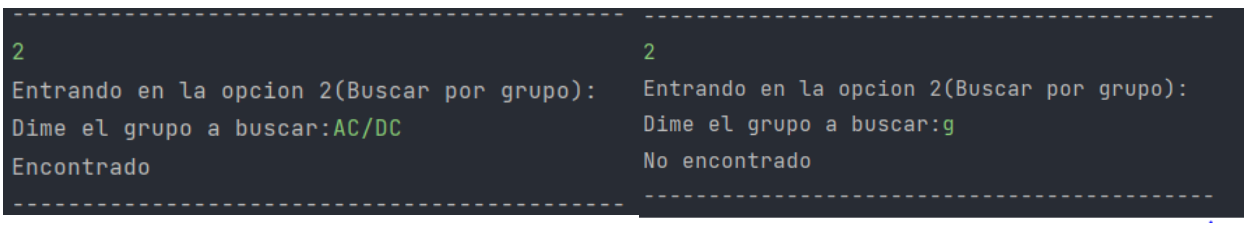


Opción 1 - Buscar por álbum:

En esta opción tendremos que escribir un nombre de un álbum. Si el álbum está guardado en el vector, el programa enviará el mensaje "Encontrado", de igual manera de que si no está enviará el mensaje "No encontrado".

Opción 2 y Opción 3- Buscar por grupo musical y género musical:

El funcionamiento de estas opciones es el mismo que el de la opción .



```
-------------------------------------------
3
Entrando en la opcion 3(Buscar por genero):
Dime el genero a buscar:Trap
No encontrado
-------------------------------------------
```

```
-------------------------------------------
3
Entrando en la opcion 3(Buscar por genero):
Dime el genero a buscar:Rock
Encontrado
-------------------------------------------
```

Opción 4- Mostrar Catálogo:

En esta opción se nos muestran los álbumes, los géneros musicales y grupos musicales guardados.

```
-------------------------------------------
4
Mostrando Catalogo
Los albumes de la discografica son:  Abbey road, Sticky Fingers, Entre el cielo y el suelo, Black in Black y Made in Hea
ven
Los grupos de muscia son: The Beatles, The Rolling Stones, Mecano, AC/DC y Queen
Los generos musicales son: Pop Rock, Rock and Roll, Pop, Hard Rock y Rock
-------------------------------------------
```

Opción 5-Salir:

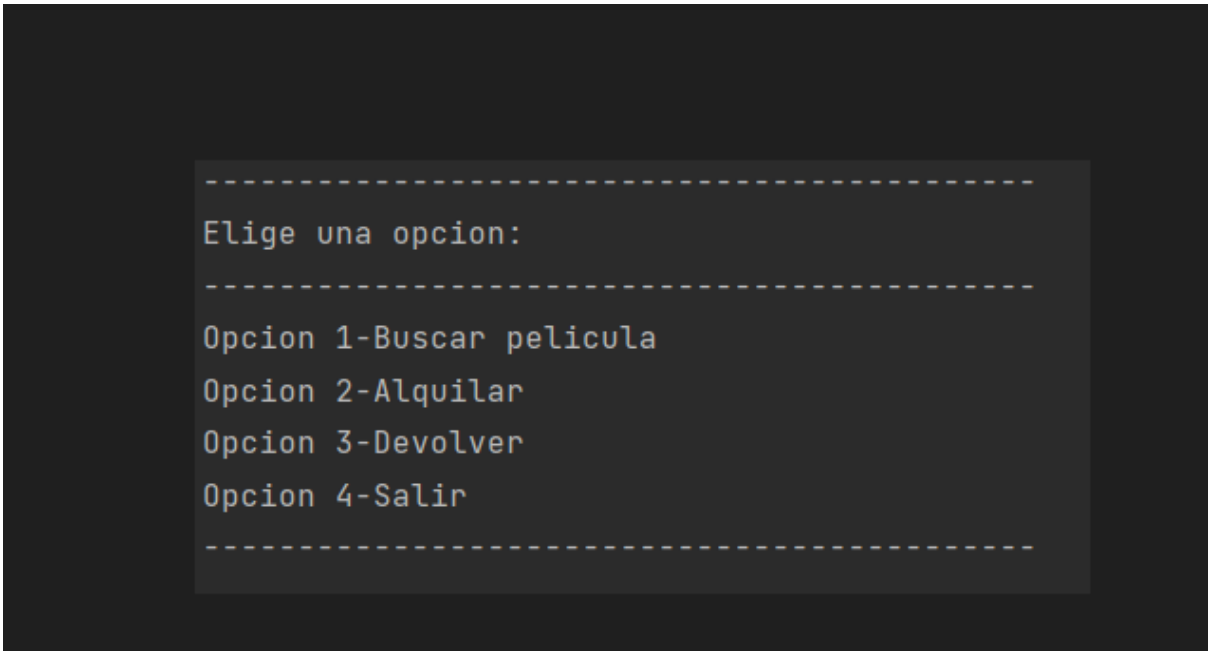
La opción salir, nos permite cortar la ejecución del programa.

```
-------------------------------------------------
5
Saliendo...

Process finished with exit code -1
```

Cada vez que elegimos una opción, menos la opción salir, el programa nos permite volver a elegir, mostrándonos de nuevo el menú de opciones.

4.2 Manual de Usuario Videoclub



Cuando iniciamos el programa nos aparece este menú que nos muestras 4 opciones:

1. Buscar película
2. Alquilar
3. Devolver
4. Salir

```
1
Introduce el nombre de la pelicula que quieres buscar:
Titanic
La pelicula Titanic
El precio es: 10
El tipo es: dvd
La pelicula no esta alquilada
```

Opción 1: Buscar película

Introducimos el nombre de la peli, como la peli no esta alquilada nos muestra todas sus propiedades

```
2
Entrando en la opcion 2(Alquilar):
0: Titanic
2: Alien
Indica el numero de peli para alquilar
0
```

Opción 2: Alquilar

Introducimos el número asignado y el programa nos alquila la peli seleccionada.

Opción 3: Devolver

```
-----------------------------------------------
3
Entrando en la opcion 3(Devolver):
0: Titanic
1: Fast and Furious
3: Paco
Indica el numero de peli para devolver
0
```

Nos muestra las pelis que estan alquiladas (Titanic la alquilamos en al opción 2 previa) e indicamos el número asociado a la peli

El programa devuelve la peli seleccionada

```
-----------------------------------------------
3
Entrando en la opcion 3(Devolver):
1: Fast and Furious
3: Paco
Indica el numero de peli para devolver
```

Para comprobar el correcto funcionamiento vemos como Titanic ya no aparece en la sección de devolver porque no está alquilada

5. Conclusión

Hemos aprendido a aplicar todos los conocimientos adquiridos durante la unidad formativa anterior como la actual, la unidad formativa 3. Donde hemos tenido varias dificultades y hemos sabido afrontarlas en la mayoría de lo posible y donde no, hemos aprendido a cambiar el rumbo del código y como grupo hemos mejorado las labores de comunicación y de coordinación a la hora de trabajar en grupo.