

## Main.cpp

```
#include <iostream>

#include "Pelicula.h"

#include "Administrador.h"

#include "Serie.h"


using namespace std;


int main()
{
    system("COLOR 0A");


    Administrador a;

    Series_TV s;


    string opc;


    a.Recuperar();


    while (true)
    {
        cout << endl;
        cout << "-----BLIM-----" << endl;
        cout << "1) MENU PELICULAS" << endl;
        cout << "2) MENU SERIES" << endl;
        cout << "3) SALIR" << endl;
        cout << "-----" << endl;
        cout << endl;
        cout << "Ingresa una opcion: ";
```

```

getline(cin,opc);

cout << endl;


//MENU PELICULAS
if (opc=="1")
{

while (true)
{
    //MENU PELICULAS
    cout << "-----PELICULAS-----" << endl;
    cout << "1) AGREGAR PELICULA" << endl; //LISTO
    cout << "2) MOSTRAR PELICULAS" << endl; //LISTO
    cout << "3) BUSCAR PELICULA" << endl; //LISTO
    cout << "4) MODIFICAR PELICULA" << endl; //LISTO "Varias Opciones"
    cout << "5) ELIMINAR PELICULA" << endl; //LISTO
    cout << "0) SALIR" << endl;
    cout << "-----" << endl;
    cout << endl;
    cout << "Ingresa una opcion: ";
    getline(cin,opc);
    cout << endl;


    //Opcion 1 "Agregar Peliculas"
    if (opc=="1")
    {
        Pelicula p;
    }
}
}

```

```

        cin >> p;

        a.Agregar(p);

        a.Respalidar();

        cin.ignore();

    }

//Opcion 2 "Mostrar Peliculas"
else if (opc=="2")
{
    a.Mostrar();

    cin.ignore();
}

//Opcion 3 "Buscar Peliculas"
else if (opc=="3")
{
    Pelicula p;

    a.Buscar(p);

    cin.ignore();
}

//Opcion 4 "Modificar Pelicula"
else if (opc=="4")
{
    //MENU MODIFICAR PELICULAS

    while (true)
    {
        cout << "-----MODIFICAR-----" << endl;

```

```
cout << "1) MODIFICAR NOMBRE" << endl;
cout << "2) MODIFICAR GENERO" << endl;
cout << "3) MODIFICAR ESTRENO" << endl;
cout << "4) MODIFICAR IDIOMA" << endl;
cout << "5) MODIFICAR TODOS LOS ATRIBUTOS" << endl;
cout << "0) SALIR" << endl;
cout << "-----" << endl;
cout << "Ingrese una opcion: ";
getline(cin,opc);
cout << endl;
```

```
if (opc=="1")
{
    Pelicula p;
    a.ModificarNombre(p);
    a.Respaldar();
    cin.ignore();
}
```

```
else if (opc=="2")
{
    Pelicula p;
    a.ModificarGenero(p);
    a.Respaldar();
    cin.ignore();
}
```

```
else if (opc=="3")
{
```

```

        Pelicula p;
        a.ModificarEstreno(p);
        a.Respalidar();
        cin.ignore();
    }

    else if (opc=="4")
    {
        Pelicula p;
        a.ModificarIdioma(p);
        a.Respalidar();
        cin.ignore();
    }

    else if (opc=="5")
    {
        Pelicula p;
        a.ModificarTodo(p);
        a.Respalidar();
        cin.ignore();
    }

    else if (opc=="0")
    {
        return main();
    }
}
}

```

//Opcion 5 "Eliminar Pelicula"

```

else if (opc=="5")
{
    string nombre;

    cout << "Ingrese la pelicula a borrar: ";

    cin >> nombre;

    a.Eliminar(nombre);

    cout << "PELICULA ELIMINADA" << endl;

    cin.ignore();

    a.Respalidar();

    cin.ignore();

}

//Opcion 0 "SALIR"
else if (opc=="0")
{
    break;

}

}

```

```

else if (opc=="2")
{
    //MENU SERIES

    cout << "-----SERIES-----" << endl;

    cout << "1) AGREGAR SERIES" << endl;

    cout << "2) MOSTRAR SERIES" << endl;

```

```
cout << "3) BUSCAR SERIE" << endl;
cout << "4) MODIFICAR SERIE" << endl;
cout << "5) ELIMINAR SERIE" << endl;
cout << "0) SALIR" << endl;
cout << "-----" << endl;
cout << endl;
cout << "Ingrese una opcion: ";
getline(cin,opc);
cout << endl;
```

```
//Opcion 1 "AGREGAR SERIE"
```

```
if (opc=="1")
```

```
{
    s.Capturar();
}
```

```
//opcion 2 "MOSTRAR SERIES"
```

```
else if (opc=="2")
```

```
{
    s.Mostrar();
}
```

```
//opcion 3 "BUSCAR SERIE"
```

```
else if (opc=="3")
```

```
{
    s.Buscar();
}
```

```
//opcion 4 "MODIFICAR SERIE"
```

```
else if (opc=="4")
{
    s.Modificar();
}

//opcion 5 "ELIMINAR SERIE"
else if (opc=="5")
{
    s.Eliminar();
}

//opcion 0 "SALIR"
else if (opc=="0")
{
    return main();
}
}

else if(opc=="0")
{
    return 0;
}
}
}
```



## **Serie.h**

```
#include <iostream>
```

```
#include <fstream>
```

```
#include <cstring>
```

```
#include <string.h>
```

```
#include <cstdlib>
```

```
#include "Idl.h"
```

```
#include "cola.h"
```

```
#define TAM_LARGO 50
```

```
#define TAM_MEDIO 40
```

```
#define TAM_CORTO 3
```

```
class Series_TV
```

```
{
```

```
public:
```

```
    char Nombre[TAM_LARGO], Genero[TAM_LARGO], Director[TAM_LARGO],  
    Temporadas[TAM_CORTO], Fecha[TAM_MEDIO];
```

```
    void Capturar();
```

```
    void Mostrar();
```

```
    void Buscar();
```

```
    void Modificar();
```

```
    void Eliminar();
```

```
} Serie;
```

```
Cola<Series_TV> cola;
```

```
int dimT, dimA, dimD, dimI, dimF, dimP, opc;
```

```
void Series_TV::Capturar()
```

```
{
```

```
    system("cls");
```

```
    fflush(stdin);
```

```
    cout<<"NOMBRE: ";
```

```
    cin.getline(Nombre,TAM_LARGO);
```

```
    cout<<"GENERO: ";
```

```
    cin.getline(Genero,TAM_LARGO);
```

```
    cout<<"DIRECTOR: ";
```

```
    cin.getline(Director,TAM_LARGO);
```

```
    cout<<"TEMPORADAS: ";
```

```
    cin.getline(Temporadas,TAM_CORTO);
```

```
    cout<<"ESTRENO: ";
```

```
    cin.getline(Fecha,TAM_MEDIO);
```

```
    ///Abrimos el archivo y se agregan los datos de acorde a sus dimensiones antes  
    especificadas
```

```
    ofstream Archivo("Series.txt",ios::app);
```

```
    dimT = strlen(Nombre);
```

```
    dimA = strlen(Genero);
```

```
    dimD = strlen(Director);
```

```
    dimI = strlen(Temporadas);
```

```
    dimF = strlen(Fecha);
```

```
///Se agregan cada uno de los datos
Archivo.write((char*)&dimT, sizeof(int));
Archivo.write((char*)&Nombre, dimT);
Archivo.write((char*)&dimA, sizeof(int));
Archivo.write((char*)&Genero, dimA);
Archivo.write((char*)&dimD, sizeof(int));
Archivo.write((char*)&Director, dimD);
Archivo.write((char*)&dimI, sizeof(int));
Archivo.write((char*)&Temporadas, dimI);
Archivo.write((char*)&dimF, sizeof(int));
Archivo.write((char*)&Fecha, dimF);
Archivo.close(); ///Se cierra el archivo
```

```
cola.push(Serie); ///Los mismos datos se anexan a la estructura cola.
```

```
cout << "SE AGREGO LA SERIE CON EXITO" << endl;
}
```

```
void Series_TV::Mostrar()
{
    ifstream lectura("Series.txt");
    ///Validamos que el archivo exista
    if(!lectura.good())
    {
        cout<<"\nEl archivo no existe...";
    }
    else
```

```
{
```

```
    cout << left;  
    cout << setw(18) << "NOMBRE ";  
    cout << setw(18) << "GENERO ";  
    cout << setw(18) << "DIRECTOR ";  
    cout << setw(18) << "TEMPORADAS ";  
    cout << setw(18) << "ESTRENO ";  
    cout << endl;
```

```
///Abrimos el archivo en modo lectura
```

```
while(!lectura.eof())
```

```
{
```

```
    lectura.read((char*)&dimT, sizeof(int));  
    lectura.read((char*)&Nombre, dimT);  
    Nombre[dimT] = '\0';  
    lectura.read((char*)&dimA, sizeof(int));  
    lectura.read((char*)&Genero, dimA);  
    Genero[dimA] = '\0';  
    lectura.read((char*)&dimD, sizeof(int));  
    lectura.read((char*)&Director, dimD);  
    Director[dimD] = '\0';  
    lectura.read((char*)&dimI, sizeof(int));  
    lectura.read((char*)&Temporadas, dimI);  
    Temporadas[dimI] = '\0';  
    lectura.read((char*)&dimF, sizeof(int));
```

```
lectura.read((char*)&Fecha, dimF);
```

```
Fecha[dimF] = '\0';
```

```
if(lectura.eof())
```

```
    break;
```

```
cout << setw(18) << Nombre;
```

```
cout << setw(18) << Genero;
```

```
cout << setw(18) << Director;
```

```
cout << setw(18) << Temporadas;
```

```
cout << setw(18) << Fecha;
```

```
cout << endl;
```

```
if(lectura.eof())
```

```
    break;
```

```
}
```

```
}
```

```
///Cerramos el archivo
```

```
lectura.close();
```

```
///Respetando los lineamientos de la estructura cola, mostramos el frente de esta.
```

```

if(cola.empty())
{

}
else
{
    cout << setw(18) << cola.front().Nombre;
    cout << setw(18) << cola.front().Genero;
    cout << setw(18) << cola.front().Director;
    cout << setw(18) << cola.front().Temporadas;
    cout << setw(18) << cola.front().Fecha;
}

}

```

```

void Series_TV::Buscar()
{
    ///Declaramos la variable para la busqueda
    char NombreBuscado[TAM_LARGO];
    int band = 0;
    system("cls");
    ///abrimos el archivo en modo lectura
    ifstream lectura("Series.txt");
    if(!lectura.good())
    {
        cout<<"\nEl archivo no existe...";
    }
}

```

```

}
else
{
    fflush(stdin);

    cout << "INGRESE EL NOMBRE DE LA SERIE: ";
    cin.getline(NombreBuscado,TAM_LARGO);

    while(!lectura.eof() && !band)
    {
        lectura.read((char*)&dimT, sizeof(int));
        lectura.read((char*)&Nombre, dimT);
        Nombre[dimT] = '\0';
        lectura.read((char*)&dimA, sizeof(int));
        lectura.read((char*)&Genero, dimA);
        Genero[dimA] = '\0';
        lectura.read((char*)&dimD, sizeof(int));
        lectura.read((char*)&Director, dimD);
        Director[dimD] = '\0';
        lectura.read((char*)&dimI, sizeof(int));
        lectura.read((char*)&Temporadas, dimI);
        Temporadas[dimI] = '\0';
        lectura.read((char*)&dimF, sizeof(int));
        lectura.read((char*)&Fecha, dimF);
        Fecha[dimF] = '\0';
        ///Hacemos la condicional de la busqueda
        if(strcmp(NombreBuscado, Nombre) == 0)
        {

```

```

        cout << endl;

        cout << "NOMBRE: " << Nombre << endl;

        cout << "GENERO: " << Genero << endl;

        cout << "DIRECTOR: " << Director << endl;

        cout << "TEMPORADAS: " << Temporadas << endl;

        cout << "ESTRENO: " << Fecha << endl;

        band = 1;

    }

}

if (!band)

{

    ///Mandamos un error si el nombre buscado no existe

    cout << "NO SE ENCUENTRA LA SERIE QUE BUSCO" << endl;

}

}

lectura.close();///Cerramos el archivo

}

```

```

void Series_TV::Modificar()

{

    /// Declaramos una bander y la variable del nombre a buscar

    int band = 0;

    char NombreBuscadoMod[TAM_LARGO];

    ///Abrimos el archivo en modo lectura

    ifstream lectura("Series.txt");

    if(!lectura.good())

```



```

{
    cout<<"\nEl archivo no existe...";
}
else
{
    cout << endl;
    cout << "-----MODIFICAR SERIE-----" << endl;
    fflush(stdin);
    cout << "INGRESE LA SERIE QUE DESEA MODIFICAR: ";
    cin.getline(NombreBuscadoMod, TAM_LARGO);
    while(!lectura.eof() && !band)
    {
        lectura.read((char*)&dimT, sizeof(int));
        lectura.read((char*)&Nombre, dimT);
        Nombre[dimT] = '\0';
        lectura.read((char*)&dimA, sizeof(int));
        lectura.read((char*)&Genero, dimA);
        Genero[dimA] = '\0';
        lectura.read((char*)&dimD, sizeof(int));
        lectura.read((char*)&Director, dimD);
        Director[dimD] = '\0';
        lectura.read((char*)&dimI, sizeof(int));
        lectura.read((char*)&Temporadas, dimI);
        Temporadas[dimI] = '\0';
        lectura.read((char*)&dimF, sizeof(int));
        lectura.read((char*)&Fecha, dimF);
        Fecha[dimF] = '\0';
    }
}

```

```

    ///Aplicamos la condicional, si el nombre existe muestra los datos
    if(strcmp(NombreBuscadoMod, Nombre) == 0)
    {
        cout << endl;

        cout << "NOMBRE" << Nombre << endl;

        cout << "GENERO: " << Genero << endl;

        cout << "DIRECTOR: " << Director << endl;

        cout << "TEMPORADAS: " << Temporadas << endl;

        cout << "ESTRENO: " << Fecha << endl;

        band = 1;

        cout<<"DESEA MODIFICAR? SI=1 NO=0: "; ///Cuestionamos al usuario si
realmente quiere modifivar
        cin>>opc;
    }

}

lectura.close(); ///Cerramos el archivo en modo lectura

if(opc == 1)
{
    ///Abrimos el archivo en modo escritura
    ///Y creamos un archivo temporal para aplicar las modificaciones
    ifstream lectura("Series.txt");
    ofstream temporal("temporal.txt", ios::app);
    while(!lectura.eof())
    {
        lectura.read((char*)&dimT, sizeof(int));

        lectura.read((char*)&Nombre, dimT);
    }
}

```

```

Nombre[dimT] = '\0';

lectura.read((char*)&dimA, sizeof(int));

lectura.read((char*)&Genero, dimA);

Genero[dimA] = '\0';

lectura.read((char*)&dimD, sizeof(int));

lectura.read((char*)&Director, dimD);

Director[dimD] = '\0';

lectura.read((char*)&dimI, sizeof(int));

lectura.read((char*)&Temporadas, dimI);

Temporadas[dimI] = '\0';

lectura.read((char*)&dimF, sizeof(int));

lectura.read((char*)&Fecha, dimF);

Fecha[dimF] = '\0';

```

///Una vez validado que los datos buscados existen en el archivo, hacemos el ingreso de los nuevos datos

```

if(strcmp(NombreBuscadoMod, Nombre) == 0)
{
    system("cls");

    cout << endl;

    cout << "MODIFIQUE LOS NUEVOS VALORES" << endl;

    fflush(stdin);

    cout<<"NOMBRE: ";

    cin.getline(Nombre,TAM_LARGO);

    cout<<"GENERO: ";

    cin.getline(Genero,TAM_LARGO);

    cout<<"DIRECTOR: ";

    cin.getline(Director,TAM_LARGO);

```

```

        cout<<"TEMPORADAS: ";
        cin.getline(Temporadas,TAM_CORTO);
        cout<<"ESTRENO: ";
        cin.getline(Fecha,TAM_MEDIO);

        dimT = strlen(Nombre);
        dimA = strlen(Genero);
        dimD = strlen(Director);
        dimI = strlen(Temporadas);
        dimF = strlen(Fecha);
    }
    if(lectura.eof())
        break;
    temporal.write((char*)&dimT, sizeof(int));
    temporal.write((char*)&Nombre, dimT);
    temporal.write((char*)&dimA, sizeof(int));
    temporal.write((char*)&Genero, dimA);
    temporal.write((char*)&dimD, sizeof(int));
    temporal.write((char*)&Director, dimD);
    temporal.write((char*)&dimI, sizeof(int));
    temporal.write((char*)&Temporadas, dimI);
    temporal.write((char*)&dimF, sizeof(int));
    temporal.write((char*)&Fecha, dimF);

    if(lectura.eof())
        break;
}

```

```

        temporal.close();///Se guardan y se cierra el archivo temporal
        lectura.close(); ///Cerramos el archivo series.txt
        remove("Series.txt");///Eliminamos el archivo original
        rename("temporal.txt", "Series.txt");///El archivo temporal se renombra y supe al
aoriginal
    }
    if (!band)
    {
        ///Mandamos mensaje de error si el nombre no existe
        cout << "NO EXISTE LA SERIE" << endl;
    }
    else
    {
        ///Terminadas las modificaciones, mandamos mensaje de exito
        cout << "SE HAN MODIFICADO LOS DATOS :)";
    }
}
}
}

```

```

void Series_TV::Eliminar()
{
    ///Declaramos bandera y la variable para la busqueda
    int band = 0;
    char NombreEliminar[TAM_LARGO];

    ///Abrimos el archivo en modo lectura
    ifstream lectura("Series.txt");
    if(!lectura.good())

```

```

{
    ///Validamos que no este vacio
    cout<<"\nEl archivo no existe...";
}
else
{

    fflush(stdin);

    cout << "INGRESE LA SERIE QUE DESEA ELIMINAR: ";
    cin.getline(NombreEliminar,TAM_LARGO);

    while(!lectura.eof() && !band)
    {
        lectura.read((char*)&dimT, sizeof(int));
        lectura.read((char*)&Nombre, dimT);
        Nombre[dimT] = '\0';
        lectura.read((char*)&dimA, sizeof(int));
        lectura.read((char*)&Genero, dimA);
        Genero[dimA] = '\0';
        lectura.read((char*)&dimD, sizeof(int));
        lectura.read((char*)&Director, dimD);
        Director[dimD] = '\0';
        lectura.read((char*)&dimI, sizeof(int));
        lectura.read((char*)&Temporadas, dimI);
        Temporadas[dimI] = '\0';
        lectura.read((char*)&dimF, sizeof(int));
        lectura.read((char*)&Fecha, dimF);
    }
}

```

```
Fecha[dimF] = '\0';
```

```
if(strcmp(NombreEliminar, Nombre) == 0)
```

```
{
```

```
    cout << endl;
```

```
    cout << "NOMBRE:" << Nombre << endl;
```

```
    cout << "GENERO: " << Genero << endl;
```

```
    cout << "DIRECTOR: " << Director << endl;
```

```
    cout << "TEMPORADAS: " << Temporadas << endl;
```

```
    cout << "ESTRENO: " << Fecha << endl;
```

```
    band = 1;
```

```
    cout<<"DESEAS ELIMINAR? SI=1 NO=0: "; ///Cuestionamos al usuario si  
    realmente quiere hacer la eliminacion
```

```
    cin>>opc;
```

```
}
```

```
}
```

```
lectura.close(); ///Cerramos el archivo en modo lectura
```

```
if(opc == 1)
```

```
{
```

```
    ifstream lectura("Series.txt"); ///Abrimos el archivo en modo lectura de nuevo
```

```
    ofstream temporal("temporal.txt", ios::app); ///Creamos un archivo temporal
```

```
    while(!lectura.eof())
```

```
{
```

```
    lectura.read((char*)&dimT, sizeof(int));
```

```
    lectura.read((char*)&Nombre, dimT);
```

```
Nombre[dimT] = '\0';
lectura.read((char*)&dimA, sizeof(int));
lectura.read((char*)&Genero, dimA);
Genero[dimA] = '\0';
lectura.read((char*)&dimD, sizeof(int));
lectura.read((char*)&Director, dimD);
Director[dimD] = '\0';
lectura.read((char*)&dimI, sizeof(int));
lectura.read((char*)&Temporadas, dimI);
Temporadas[dimI] = '\0';
lectura.read((char*)&dimF, sizeof(int));
lectura.read((char*)&Fecha, dimF);
Fecha[dimF] = '\0';
```

```
if(strcmp(NombreEliminar, Nombre) != 0)
{
    if(lectura.eof())
        break;
    temporal.write((char*)&dimT, sizeof(int));
    temporal.write((char*)&Nombre, dimT);
    temporal.write((char*)&dimA, sizeof(int));
    temporal.write((char*)&Genero, dimA);
    temporal.write((char*)&dimD, sizeof(int));
    temporal.write((char*)&Director, dimD);
    temporal.write((char*)&dimI, sizeof(int));
    temporal.write((char*)&Temporadas, dimI);
    temporal.write((char*)&dimF, sizeof(int));
```



```

        temporal.write((char*)&Fecha, dimF);

        if(lectura.eof())
            break;
    }
}

temporal.close();///Cerramos el temporal

lectura.close();///cerramos el original en modo lectura

remove("Series.txt");///Eliminamos el archivo original

rename("temporal.txt", "Series.txt");///renombramos y guardamos el archivo
temporal como el original
}

if(band == 1 && opc == 1)
{
    ///Mandamos mensjae de exito en la eliminacion
    cout << "SERIE ELIMINADA";
}

else if(band == 1 && opc == 2)
{
    ///mandamos este mensaje si el usuario selecciono que no queria eliminar
    cout << "SERIE NO ELIMINADA";
}

else
{
    ///Mensaje de error si la serie no existe
    cout << "SERIE NO ENCONTRADA";
}

if(!cola.empty())
{

```

```
int respuesta;

///Respetando los lineamientos de la cola, mostramos el mensaje si el usuario
quiere eliminar el frente de esta

cout << "NOMBRE: " << cola.front().Nombre << endl;

cout << "GENERO: " << cola.front().Genero << endl;

cout << "DIRECTOR" << cola.front().Director << endl;

cout << "TEMPORADAS: " << cola.front().Temporadas << endl;

cout << "ESTRENO: " << cola.front().Fecha << endl;

cout << "DESEA ELIMAR EL FRENTE DE LA COLA?" << endl;

cout << "1. SI" << endl;

cout << "2. NO" << endl;

cout << "INGRESE UNA OPCION: ";

cin >> respuesta;

if(respuesta == 1)
{
    cola.pop();

    cout << "FRENTE ELIMINADO " << endl;

}
else
{
    cout << "FRENTE DE LA COLA NO ELIMINADO" << endl;

}

}

}
```

## Cola.h

```
#ifndef COLA_H_INCLUDED
#define COLA_H_INCLUDED

#include <iostream>
#include <stdexcept>
#include "ldl.h"

using namespace std;

template<typename T>
class Cola
{
private:
    LDL<T> lista;
public:
    Cola() {}

    bool empty() const;
    size_t size() const;
    const T& front() const;
    const T& back() const;
    void push(const T& element);
    void pop();
};

template<typename T>
bool Cola<T>::empty() const
```

```
{  
    return lista.empty();  
}
```

```
template<typename T>  
size_t Cola<T>::size() const  
{  
    return lista.size();  
}
```

```
template<typename T>  
const T& Cola<T>::front() const  
{  
    if(empty())  
    {  
        throw range_error("Trying front() from an empty queue");  
    }  
    return lista.front();  
}
```

```
template<typename T>  
const T& Cola<T>::back() const  
{  
    if(empty())  
    {  
        throw range_error("Trying back() from an empty queue");  
    }  
}
```

```

        return lista.back();
    }

template<typename T>
void Cola<T>::push(const T& element)
{
    lista.push_back(element);
}

template<typename T>
void Cola<T>::pop()
{
    if(empty())
    {
        throw range_error("Trying pop() from an empty queue");
    }

    lista.pop_front();
}

#endif // COLA_H_INCLUDED

```

## IDL.h

```

#ifndef LDL_H
#define LDL_H

#include <iostream>
#include <stdexcept>
#include <memory>

```

```
using namespace std;
```

```
template<typename T>
```

```
class LDL
```

```
{
```

```
private:
```

```
    struct NodoLDL
```

```
    {
```

```
        T value;
```

```
        shared_ptr<NodoLDL> prev;
```

```
        shared_ptr<NodoLDL> next;
```

```
        NodoLDL(const T& elem, shared_ptr<NodoLDL> p = nullptr, shared_ptr<NodoLDL> n =  
nullptr) :
```

```
            value(elem), prev(p), next(n)
```

```
        {}
```

```
    };
```

```
    size_t listSize;
```

```
    shared_ptr<NodoLDL> listFront;
```

```
    shared_ptr<NodoLDL> listBack;
```

```
public:
```

```
    LDL()
```

```
    {
```

```
        listSize = 0;
```

```
        listFront = nullptr;
```

```
        listBack = nullptr;
```

```
    }
```

```

~LDL()
{
    clear();
}

bool empty() const;
size_t size() const;
void push_front(const T& elem);
void push_back(const T& elem);
const T& front() const;
const T& back() const;
void pop_front();
void pop_back();
void insert(size_t position, const T& elem);
void erase(size_t position);
void clear();
void remove(const T& value);
T& operator [] (size_t position);
};

```

```

template<typename T>
bool LDL<T>::empty() const
{
    return listSize == 0;
}

```

```

template<typename T>

```

```
size_t LDL<T>::size() const
```

```
{  
    return listSize;  
}
```

```
template<typename T>
```

```
void LDL<T>::push_front(const T &elem)
```

```
{  
    if (empty())  
    {  
        listFront = make_shared<NodoLDL>(elem);  
        listBack = listFront;  
    }  
    else  
    {  
        shared_ptr<NodoLDL> temp = make_shared<NodoLDL>(elem, nullptr, listFront);  
        listFront->prev = temp;  
        listFront = temp;  
    }  
    ++listSize;  
}
```

```
template<typename T>
```

```
void LDL<T>::push_back(const T &elem)
```

```
{  
    if (empty())  
    {
```



```

        listFront = make_shared<NodoLDL>(elem);
        listBack = listFront;
    }
    else
    {
        shared_ptr<NodoLDL> temp = make_shared<NodoLDL>(elem, listBack);
        listBack->next = temp;
        listBack = temp;
    }
    ++listSize;
}

```

```

template<typename T>
const T &LDL<T>::front() const
{
    if (empty())
    {
        throw range_error("Trying front() from empty list");
    }
    return listFront->value;
}

```

```

template<typename T>
const T &LDL<T>::back() const
{
    if (empty())
    {

```

```

        throw range_error("Trying back() from empty list");
    }
    return listBack->value;
}

template<typename T>
void LDL<T>::pop_front()
{
    if (empty())
    {
        throw range_error("Trying pop_front() from empty list");
    }
    if (size() == 1)
    {
        listFront = nullptr;
        listBack = nullptr;
    }
    else
    {
        listFront = listFront->next;
        listFront->prev->next = nullptr;
        listFront->prev = nullptr;
    }

    --listSize;
}

```

```

template<typename T>
void LDL<T>::pop_back()
{
    if (empty())
    {
        throw range_error("Trying pop_back() from empty list");
    }
    if (size() == 1)
    {
        listFront = nullptr;
        listBack = nullptr;
    }
    else
    {
        listBack = listBack->prev;
        listBack->next->prev = nullptr;
        listBack->next = nullptr;
    }
    --listSize;
}

```

```

template<typename T>
void LDL<T>::insert(size_t position, const T &elem)
{
    if (position > size())
    {
        throw range_error("Trying insert() in non valid position");
    }
}

```

```

    }
    if (position == 0)
    {
        push_front(elem);
    }
    else if (position == size())
    {
        push_back(elem);
    }
    else
    {
        shared_ptr<NodoLDL> temp = listFront;
        for (size_t i(0); i < position-1; ++i)
        {
            temp = temp->next;
        }
        shared_ptr<NodoLDL> nuevo = make_shared<NodoLDL>(elem, temp, temp->next);
        temp->next = nuevo;
        nuevo->next->prev = nuevo;
        ++listSize;
    }
}

```

```

template<typename T>
void LDL<T>::erase(size_t position)
{
    if (empty())

```

```

{
    throw range_error("Trying erase() from empty list");
}
if (position >= size())
{
    throw range_error("Trying erase() in non valid position");
}
if (position == 0)
{
    pop_front();
}
else if (position == size()-1)
{
    pop_back();
}
else
{
    shared_ptr<NodoLDL> temp = listFront;
    for (size_t i(0); i < position-1; ++i)
    {
        temp = temp->next;
    }
    shared_ptr<NodoLDL> eliminar = temp->next;
    temp->next = eliminar->next;
    eliminar->next->prev = temp;
    eliminar->next = nullptr;
    eliminar->prev = nullptr;
}

```

```
        --listSize;
    }
}
```

```
template<typename T>
void LDL<T>::clear()
{
    for (size_t i(0), j(size()); i < j; ++i)
    {
        pop_front();
    }
}
```

```
template<typename T>
void LDL<T>::remove(const T &value)
{
    if (empty())
    {
        throw range_error("Trying remove() from empty list");
    }

    T dato;
    size_t i(0);
    shared_ptr<NodoLDL> temp = listFront;
    while(temp != nullptr)
    {
        dato = temp->value;
        temp = temp->next;
```

```

        if (dato == value)
        {
            erase(i);

            --i;
        }

        ++i;
    }
}

```

```

template<typename T>
T &LDL<T>::operator [] (size_t position)
{
    if (empty())
    {
        throw range_error("Trying [] from empty list");
    }

    if (position >= size())
    {
        throw range_error("Trying [] in non valid position");
    }

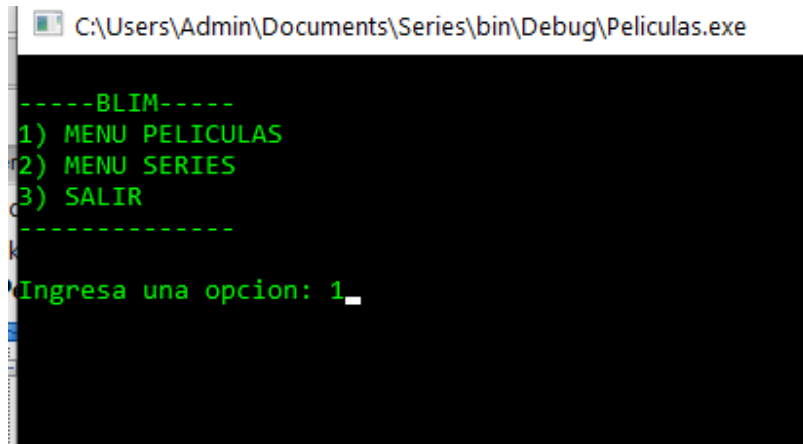
    shared_ptr<NodoLDL> temp = listFront;
    for (size_t i(0); i < position; ++i)
    {
        temp = temp->next;
    }

    return temp->value;
}

```

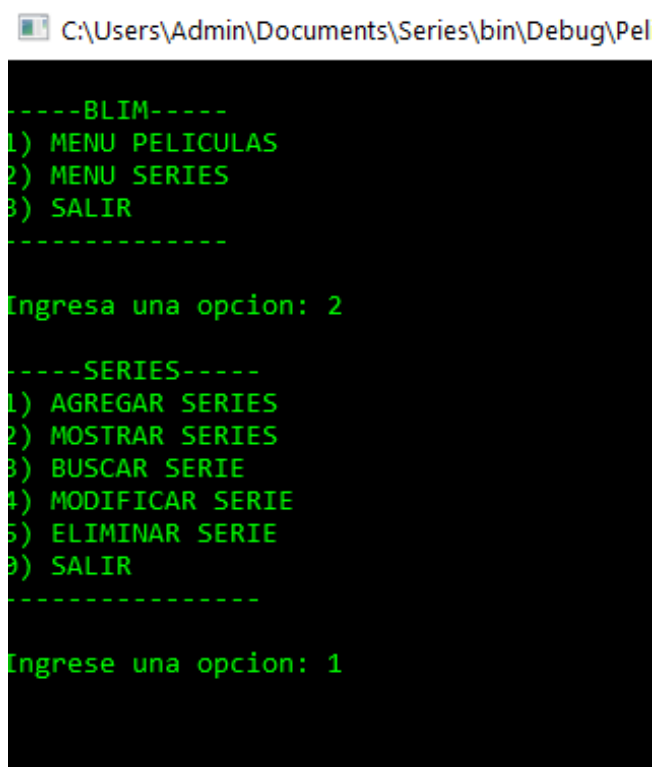
```
#endif // LDL_H
```

## Capturad de ejecución:



```
C:\Users\Admin\Documents\Series\bin\Debug\Peliculas.exe

-----BLIM-----
1) MENU PELICULAS
2) MENU SERIES
3) SALIR
-----
Ingresa una opcion: 1_
```



```
C:\Users\Admin\Documents\Series\bin\Debug\Pel

-----BLIM-----
1) MENU PELICULAS
2) MENU SERIES
3) SALIR
-----
Ingresa una opcion: 2

-----SERIES-----
1) AGREGAR SERIES
2) MOSTRAR SERIES
3) BUSCAR SERIE
4) MODIFICAR SERIE
5) ELIMINAR SERIE
6) SALIR
-----
Ingrese una opcion: 1
```



C:\Users\Admin\Documents\Series\bin\Debug\Peliculas.exe

```
NOMBRE: Batman
GENERO: Accion
DIRECTOR: Javier Olivares
TEMPORADAS: 5
ESTRENO: 17/08/2022
SE AGREGO LA SERIE CON EXITO
```

C:\Users\Admin\Documents\Series\bin\Debug\Peliculas.exe

-----BLIM-----

- 1) MENU PELICULAS
- 2) MENU SERIES
- 3) SALIR

-----

Ingresa una opcion: 2

-----SERIES-----

- 1) AGREGAR SERIES
- 2) MOSTRAR SERIES
- 3) BUSCAR SERIE
- 4) MODIFICAR SERIE
- 5) ELIMINAR SERIE
- 0) SALIR

-----

Ingresa una opcion: 2

NOMBRE	GENERO	DIRECTOR	TEMPORADAS	ESTRENO
Bob	Animacion	Na	8	2001
Sherlock	Crimen	NA	4	2012

-----SERIES-----

- 1) AGREGAR SERIES
- 2) MOSTRAR SERIES
- 3) BUSCAR SERIE
- 4) MODIFICAR SERIE
- 5) ELIMINAR SERIE
- 0) SALIR

-----

Ingresa una opcion: 3

C:\Users\Admin\Documents\Series\bin\Debug\Peliculas.exe

INGRESE EL NOMBRE DE LA SERIE: Bob

NOMBRE: Bob  
GENERO: Animacion  
DIRECTOR: Na  
TEMPORADAS: 8  
ESTRENO: 2001

-----BLIM-----  
1) MENU PELICULAS  
2) MENU SERIES  
3) SALIR

-----  
Ingresa una opcion: \_

-----BLIM-----  
1) MENU PELICULAS  
2) MENU SERIES  
3) SALIR  
-----

Ingresa una opcion: 2

-----SERIES-----  
1) AGREGAR SERIES  
2) MOSTRAR SERIES  
3) BUSCAR SERIE  
4) MODIFICAR SERIE  
5) ELIMINAR SERIE  
0) SALIR  
-----

Ingresa una opcion: 4

-----MODIFICAR SERIE-----  
INGRESE LA SERIE QUE DESEA MODIFICAR: Bob

C:\Users\Admin\Documents\Series\bin\Debug\Pe

```
MODIFIQUE LOS NUEVOS VALORES
NOMBRE: Batman
GENERO: Accion
DIRECTOR: Javier Lopez Murillo
TEMPORADAS: 2
ESTRENO: 2023
SE HAN MODIFICADO LOS DATOS :)
-----BLIM-----
1) MENU PELICULAS
2) MENU SERIES
3) SALIR
-----
Ingresa una opcion: _
```

C:\Users\Admin\Documents\Series\bin\Debug\Peliculas.exe

```
Ingresa una opcion: 2
-----SERIES-----
1) AGREGAR SERIES
2) MOSTRAR SERIES
3) BUSCAR SERIE
4) MODIFICAR SERIE
5) ELIMINAR SERIE
0) SALIR
-----
Ingresa una opcion: 5
INGRESE LA SERIE QUE DESEA ELIMINAR: Batman
NOMBRE:Batman
GENERO: Accion
DIRECTOR: Javier Lopez Murillo
TEMPORADAS: 2
ESTRENO: 2023
DESEAS ELIMINAR? SI=1 NO=0: 1
SERIE ELIMINADANOMBRE:
GENERO:
DIRECTOR
TEMPORADAS:
ESTRENO:
DESEA ELIMAR EL FRENTE DE LA COLA?
1. SI
2. NO
INGRESE UNA OPCION: 1_
```