**UNIVERSIDAD DE GUADALAJARA**

**UA: SEMINARIO DE SOLUCIÓN DE PROBLEMAS DE ESTRUCTURAS DE DATOS II**

**PRÁCTICA NO.3**

**CEBALLOS JIMENEZ CHRISTOPHER**

**MAIN.CPP**

#include <iostream> #include "Pelicula.h" #include "Administrador.h" #include "Serie.h" #include "Usuarios.h"

using namespace std;

int main()

{

system("COLOR 0A");

Administrador a; Series\_TV s; Usuario u;

string opc;

a.Recuperar();

while (true)

{

cout << endl;

cout << "-----BLIM " << endl;

cout << "1) MENU USUARIOS" << endl; cout << "2) MENU PELICULAS" << endl; cout << "3) MENU SERIES" << endl;

cout << "0) SALIR" << endl; cout << " " << endl;

cout << endl;

cout << "Ingresa una opcion: "; getline(cin,opc);

cout << endl;

//MENU PELICULAS

if (opc=="2")

{

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.Respaldar(); 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.Respaldar(); cin.ignore();

}

else if (opc=="4")

{

Pelicula p; a.ModificarIdioma(p); a.Respaldar(); cin.ignore();

}

else if (opc=="5")

{

Pelicula p; a.ModificarTodo(p); a.Respaldar(); 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.Respaldar(); cin.ignore();

}

//Opcion 0 "SALIR" else if (opc=="0")

{

break;

}

}

}

else if (opc=="3")

{

//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=="1")

{

//MENU USUARIOS

cout << "-----USUARIOS " << endl;

cout << "1) AGREGAR USUARIO" << endl; cout << "2) MOSTRAR USUARIOS" << endl; cout << "3) BUSCAR USUARIO" << endl; cout << "4) MODIFICAR USUARIO" << endl; cout << "5) ELIMINAR USUARIO" << endl;

cout << "0) SALIR" << endl; cout << " " << endl;

cout << endl;

cout << "Ingrese una opcion: "; getline(cin,opc);

cout << endl;

if (opc=="1")

{

u.Capturar();

}

else if (opc=="2")

{

u.Mostrar();

}

else if (opc=="3")

{

u.Buscar();

}

else if (opc=="4")

{

u.Modificar();

}

else if (opc=="5")

{

u.Eliminar();

}

else if (opc=="0")

{

return main();

}

}

else if (opc=="0")

{

return 0;

}

}

}

# USUARIOS.H

#ifndef USUARIOS\_H #define USUARIOS\_H

#include <cstdlib> #include <iostream> #include <fstream> #include <cstring> #include "ldl.h"

#define TAM\_LARGO 50

#define TAM\_MEDIO 20

#define TAM\_CORTO 16

using namespace std;

class Usuario

{

public:

char Nombre[TAM\_LARGO], User[TAM\_CORTO], Password[TAM\_MEDIO];

void Capturar();

void Mostrar(); void Buscar(); void Modificar(); void Eliminar(); void Cargar();

} Registro;

LDL<Usuario> lista;

int dimN, dimC, dimPassword;

void Usuario::Capturar()

{

system("cls");

fflush(stdin); cout<<"Nombre: ";

cin.getline(Nombre,TAM\_LARGO); cout<<"Usuario: "; cin.getline(User,TAM\_CORTO); cout<<"Contraseña: ";

cin.getline(Password,TAM\_MEDIO);

ofstream Archivo("Usuarios.txt",ios::app); dimN = strlen(Nombre);

dimC = strlen(User); dimPassword = strlen(Password);

Archivo.write((char\*)&dimN, sizeof(int)); Archivo.write((char\*)&Nombre, dimN); Archivo.write((char\*)&dimC, sizeof(int)); Archivo.write((char\*)&User, dimC); Archivo.write((char\*)&dimPassword, sizeof(int)); Archivo.write((char\*)&Password, dimPassword); Archivo.close();

lista.push\_back(Registro);

cout << "Usuario Creado correctamente";

}

void Usuario::Mostrar()

{

ifstream lectura("Usuarios.txt");

if(!lectura.good())

{

cout<<"\nEl archivo no existe...";

}

else

{

cout << left;

cout << setw(18) << "NOMBRE "; cout << setw(18) << "USUARIO "; cout << setw(18) << "PASWORD "; cout << endl;

while(!lectura.eof())

{

lectura.read((char\*)&dimN, sizeof(int)); lectura.read((char\*)&Nombre, dimN); Nombre[dimN] = '\0'; lectura.read((char\*)&dimC, sizeof(int)); lectura.read((char\*)&User, dimC); User[dimC] = '\0';

lectura.read((char\*)&dimPassword, sizeof(int)); lectura.read((char\*)&Password, dimPassword); Password[dimPassword] = '\0';

if(lectura.eof()) break;

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

cout << setw(18) << Password; cout << endl;

if(lectura.eof()) break;

}

}

lectura.close();

}

void Usuario::Buscar()

{

char NombreBuscado[TAM\_LARGO]; int band = 0;

system("cls");

ifstream lectura("Usuarios.txt"); if(!lectura.good())

{

cout<<"\nEl archivo no existe...";

}

else

{

cout << endl << endl;

cout << "Ingrese el Nombre a buscar: "; cin.getline(NombreBuscado,TAM\_LARGO);

while(!lectura.eof() && !band)

{

lectura.read((char\*)&dimN, sizeof(int)); lectura.read((char\*)&Nombre, dimN);

Nombre[dimN] = '\0'; lectura.read((char\*)&dimC, sizeof(int)); lectura.read((char\*)&User, dimC); User[dimC] = '\0';

lectura.read((char\*)&dimPassword, sizeof(int)); lectura.read((char\*)&Password, dimPassword); Password[dimPassword] = '\0';

if( strcmp(NombreBuscado, Nombre) == 0)

{

cout << endl;

cout << "Nombre: " << Nombre << endl; cout << "User: " << User << endl;

cout << "Password: " << Password << endl; band = 1;

}

}

if (!band)

{

cout << "NO SE ENCUENTRA EL USUARIO..." << endl;

}

}

lectura.close();

}

void Usuario::Modificar()

{

int band = 0;

char NombreBuscadoMod[TAM\_LARGO];

ifstream lectura("Usuarios.txt"); if(!lectura.good())

{

cout<<"\nEl archivo no existe...";

}

else

{

cout << endl << endl;

fflush(stdin);

cout << "Ingrese el Nombre a buscar y posteriormente a Modificar: "; cin.getline(NombreBuscadoMod, TAM\_LARGO);

while(!lectura.eof() && !band)

{

lectura.read((char\*)&dimN, sizeof(int)); lectura.read((char\*)&Nombre, dimN); Nombre[dimN] = '\0'; lectura.read((char\*)&dimC, sizeof(int)); lectura.read((char\*)&User, dimC); User[dimC] = '\0';

lectura.read((char\*)&dimPassword, sizeof(int)); lectura.read((char\*)&Password, dimPassword); Password[dimPassword] = '\0';

if(strcmp(NombreBuscadoMod, Nombre) == 0)

{

cout << endl;

cout << "Nombre : " << Nombre << endl; cout << "User : " << User << endl;

cout << "Password : " << Password << endl; band = 1;

cout<<"DESEA MODIFICAR? SI=1 NO=0: ";

cin>>opc;

}

}

lectura.close();

if(opc == 1)

{

ifstream lectura("Usuarios.txt");

ofstream temporal("temporal.txt", ios::app); while(!lectura.eof())

{

lectura.read((char\*)&dimN, sizeof(int)); lectura.read((char\*)&Nombre, dimN); Nombre[dimN] = '\0'; lectura.read((char\*)&dimC, sizeof(int)); lectura.read((char\*)&User, dimC); User[dimC] = '\0';

lectura.read((char\*)&dimPassword, sizeof(int)); lectura.read((char\*)&Password, dimPassword); Password[dimPassword] = '\0';

if(strcmp(NombreBuscadoMod, Nombre) == 0)

{

system("cls"); cout << endl;

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

fflush(stdin); cout<<"Nombre: ";

cin.getline(Nombre,TAM\_LARGO); cout<<"User: "; cin.getline(User,TAM\_LARGO); cout<<"Password: "; cin.getline(Password,TAM\_LARGO);

dimN = strlen(Nombre); dimC = strlen(User);

dimPassword = strlen(Password);

}

if(lectura.eof()) break;

temporal.write((char\*)&dimN, sizeof(int)); temporal.write((char\*)&Nombre, dimN); temporal.write((char\*)&dimC, sizeof(int)); temporal.write((char\*)&User, dimC); temporal.write((char\*)&dimPassword, sizeof(int)); temporal.write((char\*)&Password, dimPassword);

if(lectura.eof())

break;

}

temporal.close(); lectura.close(); remove("Usuarios.txt");

rename("temporal.txt", "Usuarios.txt");

}

}

cout << "USUARIO MODIFICADO CON EXITO";

lista.clear(); Cargar();

}

void Usuario::Eliminar()

{

int band = 0;

char NombreEliminar[TAM\_LARGO];

ifstream lectura("Usuarios.txt"); if(!lectura.good())

{

cout<<"ARCHIVO NO ENCONTRADO";

}

else

{

cout << endl << endl; fflush(stdin);

cout << "[?] Ingrese el Nombre para Eliminar: "; cin.getline(NombreEliminar,TAM\_LARGO);

while(!lectura.eof() && !band)

{

lectura.read((char\*)&dimN, sizeof(int)); lectura.read((char\*)&Nombre, dimN); Nombre[dimN] = '\0'; lectura.read((char\*)&dimC, sizeof(int)); lectura.read((char\*)&User, dimC); User[dimC] = '\0';

lectura.read((char\*)&dimPassword, sizeof(int)); lectura.read((char\*)&Password, dimPassword); Password[dimPassword] = '\0';

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

{

cout << endl;

cout << "Nombre : " << Nombre << endl; cout << "User : " << User << endl;

cout << "Password: " << Password << endl; band = 1;

cout<<"DESEA ELIMINAR? SI=1 NO=0: ";

cin>>opc;

}

}

lectura.close();

if(opc == 1)

{

ifstream lectura("Usuarios.txt");

ofstream temporal("temporal.txt", ios::app); while(!lectura.eof())

{

lectura.read((char\*)&dimN, sizeof(int)); lectura.read((char\*)&Nombre, dimN); Nombre[dimN] = '\0'; lectura.read((char\*)&dimC, sizeof(int)); lectura.read((char\*)&User, dimC); User[dimC] = '\0';

lectura.read((char\*)&dimPassword, sizeof(int)); lectura.read((char\*)&Password, dimPassword); Password[dimPassword] = '\0';

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

{

if(lectura.eof()) break;

temporal.write((char\*)&dimN, sizeof(int)); temporal.write((char\*)&Nombre, dimN); temporal.write((char\*)&dimC, sizeof(int)); temporal.write((char\*)&User, dimC); temporal.write((char\*)&dimPassword, sizeof(int)); temporal.write((char\*)&Password, dimPassword); if(lectura.eof())

break;

}

}

temporal.close(); lectura.close(); remove("Usuarios.txt");

rename("temporal.txt", "Usuarios.txt");

}

}

lista.clear(); Cargar();

}

void Usuario::Cargar()

{

ifstream lectura("Usuarios.txt");

if(lectura.good())

{

while(!lectura.eof())

{

lectura.read((char\*)&dimN, sizeof(int)); lectura.read((char\*)&Nombre, dimN); Nombre[dimN] = '\0'; lectura.read((char\*)&dimC, sizeof(int)); lectura.read((char\*)&User, dimC); User[dimC] = '\0';

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

lectura.read((char\*)&Password, dimPassword); Password[dimPassword] = '\0';

if(lectura.eof()) break;

lista.push\_back(Registro);

if(lectura.eof()) break;

}

}

lectura.close();

}

#endif // USUARIOS\_H

# 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

# CAPTURAS DE EJECUCIÓN



 







