



Universidad de Guadalajara

Centro Universitario de Ciencias Exactas e Ingeniería
Departamento de Ciencias Computacionales

Seminario de solución de problemas de estructuras de datos II
Manual de proyecto “Blim”



Profesor: Janette Araceli Castellanos Barajas

Tabla de contenido

Tabla de Contenidos	2
Descripcion del proyecto	3
Analisis del proyecto.....	4
Temas que se manejan durante el proyecto	5
Introduccion a archivos	5
Delimitadores.....	6
Campos de dimensión.....	7
Registros de longitud fija.....	8
Índices simples	9
Indice simple	9
Indice secundario	9
Serializacion.....	9
Encriptación y desencriptación	10
Dispersion o Hashing.....	11
Implementacion del código.....	13
Clase Peliculas	13
Clase Series de TV.....	18
Clase Lista de Reproduccion.....	23
Clase Pagos.....	30
Clase Historial.....	37
Clase Usuario y Encriptacion	41
Instrucciones de uso	48
¿Cómo correr el programa?	48
Lado Administrador.....	50
Lado Usuario	67
Recomendaciones para el uso correcto del sistema	79
Acerca de nosotros	80
Conclusiones	81
Bibliografía.....	82

Descripción del proyecto

La empresa comercial de entretenimiento BLIM desea proporcionar mediante internet películas, novelas y series de televisión a una lista de usuarios registrados, los cuales deben cubrir una cuota para gozar del beneficio.

Se requiere que los usuarios se registren y paguen el servicio proporcionando una tarjeta de crédito visa o MasterCard lo cual deberá mantenerse bajo control para permitir la entrada a los usuarios o bien denegarla. cada usuario tendrá su fecha de inicio de servicio y corte de manera individual.

El usuario podrá crear listas de reproducción y listas de favoritos. también deberá permitir restricción de niños y crear máximo 3 cuentas por usuario.

Las categorías que se manejan son acción, romántico, animación, comedia, lo más popular, programas de tv, populares en Facebook, populares en YouTube y nuevos.

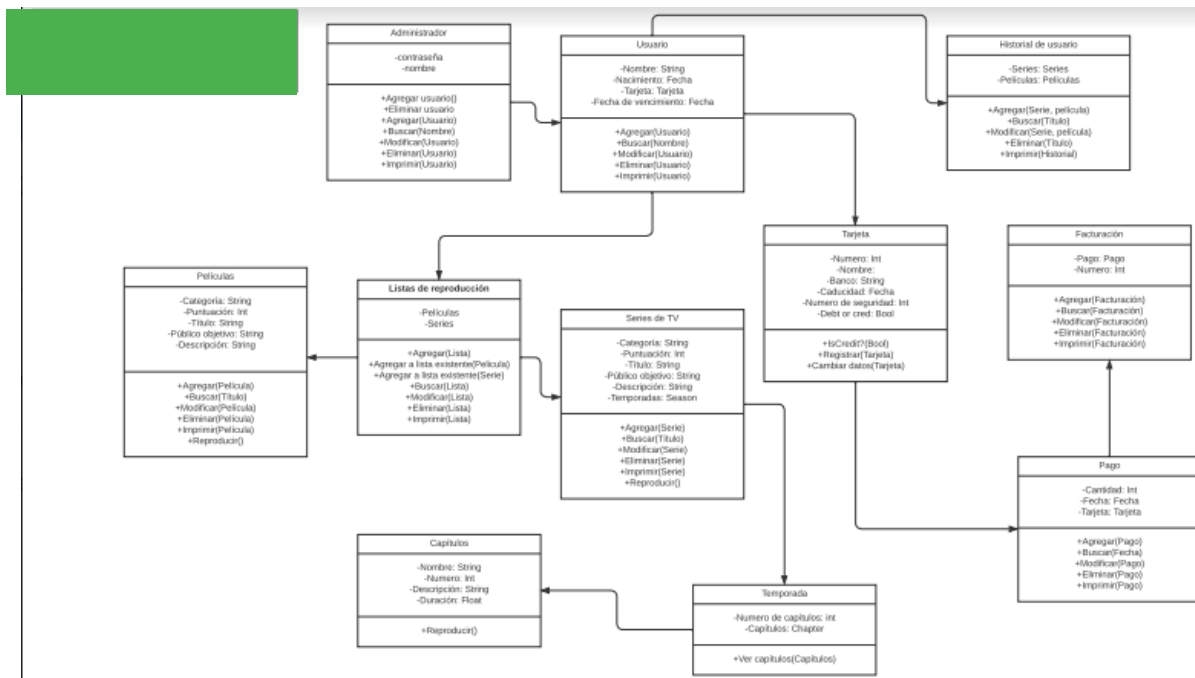
El usuario podrá solicitar facturación del servicio por lo que deberá proporcionar los datos correspondientes.

La contratación del servicio se puede realizar vía internet o vía telefónica, siempre proporcionando el número de tarjeta de crédito.

Análisis del proyecto

Para poder definir bien lo que se iba a estar programando a lo largo del semestre, definimos un esquema UML o de clases donde identificamos el nombre de las clases, sus atributos y métodos que iban a tener.

El diagrama de clases es una técnica central y ampliamente difundida en los distintos métodos orientados a objeto. Cada método incluye sus propias variantes a esta técnica, pero en el presente manual nos vamos a centrar en la visión que se encuentra implementada dentro del lenguaje estándar de modelado UML 1.1



Temas que se manejan durante el proyecto

Con el fin de poder explicar las diferentes técnicas que se usaron para la creación de este proyecto, explicaremos los temas, en que consisten y posteriormente veremos su implementación en código y en el resultado final.

Introducción a archivos

Se pueden manejar gran cantidad de datos del mismo y diferente tipo al mismo tiempo (arrays y arrays de estructuras).

El problema es que el programa retiene los datos mientras esté ejecutándose y se pierden al terminar la ejecución.

La solución para hacer que los datos no se pierdan es almacenarlos en un fichero o archivo.

Los archivos son medios que facilita el lenguaje para almacenar los datos en forma permanente, normalmente en los dispositivos de almacenamiento estándar.

Desde el punto de vista informático, un fichero es una colección de información que almacenamos en un soporte magnético para poder manipularla en cualquier momento.

Esta información se almacena como un conjunto de registros.



Delimitadores

Un delimitador es una secuencia de uno o más caracteres que se utilizan para especificar el límite entre las regiones separadas, independientes en texto o en otros flujos de datos.

Es decir, signos especiales que permiten al compilador separar y reconocer las diferentes unidades sintácticas del lenguaje.

Tenemos dos principales tipos de delimitadores:

Delimitadores de campo: Estos delimitadores separan los distintos campos de un registro, por ejemplo, el campo nombre, edad, teléfono, etc.

Delimitadores de registro: Los delimitadores de registros separan grupos de campos, por ejemplo, en una solicitud de empleo cuyos registros serían datos personales, escolaridad, documentación, etc.

Además, tenemos otros dos tipos:

Delimitadores de soporte: Delimitadores de corchete marcan el inicio y el final de una región de texto.

Delimitadores de choque: Delimitador de colisión es un problema que se produce cuando un autor o programador introduce delimitadores en texto sin realmente la intención que sean interpretados como límites entre regiones separadas.

Campos de dimensión

Hay muchas formas de añadir estructura a los archivos para mantener la entidad de los campos.

Los tres métodos más comunes son:

- Forzar que los campos tengan una longitud predecible;
- Comenzar cada campo con un indicador de longitud, y
- Colocar un delimitador al final de cada campo para separarlo del siguiente.

Comenzar cada campo con un indicador de longitud es otra forma de contar hasta el final de un campo es almacenar su longitud delante del campo, como se muestra en siguiente imagen. Si los campos no son demasiado largos, es posible almacenar la longitud en un solo byte al inicio de cada campo.



04Ames04John09123 Maple 10 Stillwater 020K057407509Mason04Alan1190 Eastgate...

(b)

Registros de longitud fija

Es aquel cuyos registros contienen todo el mismo número de bytes. En este se tiene un número fijo de campos, cada uno con longitud predeterminada. Es uno de los métodos más usados para organizar archivos.

Por ejemplo, imaginemos que tenemos los siguientes datos:

Mary Ames	Alan Mason
123 Maple	90 Eastgate
Stillwater OK 74075	Ada OK 74820

Y para el objeto tenemos definida la siguiente estructura:

```
class Person { public:  
    char last[11];  
    char first [11];  
    char address [16];  
    char city[16];  
    char state[3];  
    char zip[10];  
};
```

Al aplicar la técnica de registros de longitud fija, en el fichero quedara almacenado de la siguiente manera:

Ames	Mary	123 Maple	Stillwater	OK	74075
Mason	Alan	90 Eastgate	Ada	OK	74820

Índices simples

Los libros contienen índices, esto es: Tablas con una lista de temas (llaves) y números de página en donde se puede encontrar dichos temas (campos de referencia).

El índice de un libro nos permite encontrar de forma inmediata un tema, de otra forma se tardaría mucho tiempo en encontrar el tema deseado.

Ahora pasemos a otro tema, ¿Qué es la Indización? Bueno, pues como el índice trabaja por indirección, permite imponer un orden en un archivo sin que realmente se reacomode. Proporciona varios caminos de acceso a un archivo.

Proporciona acceso por llave a archivos de registros de longitud variable.

Índice simple

La estructura del archivo de índices es un archivo de registros de longitud fija, donde cada registro tiene dos campos de longitud fija:

- un campo llave.
- un campo de distancia en bytes.

Para cada registro del archivo de datos hay un registro en el archivo de índices.

Índice secundario

El archivo de índices tiene que reacomodarse cada vez que se agrega un registro nuevo al archivo. Si hay llaves secundarias duplicadas, el campo de llave secundaria se repite para cada entrada. Esto desperdicia espacio, ya que los archivos se agrandan más de lo necesario.

Serialización

La Serialización es un proceso mediante el cual podemos convertir objetos de un programa en ejecución en flujos de bytes capaces de ser almacenados en dispositivos, bases de datos o de ser enviados a través de la red, y ser capaces de reconstruirlos en los equipos donde sea necesario.

Uno de los principales objetivos de la serialización es permitir crear flujos de bytes independientes de la arquitectura de los equipos en los que se utilicen, inclusive, que los objetos puedan ser reconstruidos en otros programas sin importar que el lenguaje con el que son escritos sea diferente al que se usó para crear el objeto originalmente.

Encriptación y desencriptación

La encriptación o cifrado es un mecanismo de seguridad que permite modificar un mensaje de modo que su contenido sea ilegible, salvo para su destinatario. De modo inverso, la desencriptación o descifrado permitirá hacer legible un mensaje que estaba cifrado.

Usando criptografía de clave pública el emisor del mensaje cifrará el mensaje aplicando la clave pública del destinatario. Será por tanto el destinatario, el único que podrá descifrar el mensaje aplicando su clave privada.

El proceso contrario se llama desencriptación el cual consiste en la recuperación de una información que se había protegido mediante un proceso de cifrado.

Dispersión o Hashing

Los algoritmos hash surgen de la necesidad de encontrar una forma más rápida de localizar información. La dispersión o hashing consiste en una transformación matemática de una clave K en una dirección, y una función de dispersión $h(K)$ que da como resultado la posición de K en una tabla.

Este proceso tiene las siguientes características:

- Las direcciones generadas parecen ser aleatorias.
- La dispersión a diferencia de la indización, no requiere almacenamiento adicional para mantener un índice.
- La dispersión facilita la inserción y eliminación muy rápidas de los registros
- La dispersión permite encontrar un registro con muy pocos accesos al disco.
- La dispersión no permite llaves duplicadas.

Entre los algoritmos más comunes para una función Hash encontramos los siguientes:

- Residuo / división
- Medio Cuadrado
- Método de plegamiento
- Truncamiento

A las claves que por dispersión se convierten en la misma dirección se les denomina sinónimos (colisión).

Cuando dos o más claves diferentes tras el proceso de dispersión o transformación de claves tienen el mismo resultado se le conoce como colisión.

Los métodos más populares para manejar colisiones son:

- Algoritmo de dispersión perfecta
- Compartimientos o buckets
- Saturación progresiva
- Encadenamiento
- Doble dirección Hash

Entre las ventajas que podemos obtener usando esta técnica se encuentran:

- Se pueden usar los valores naturales de la llave, puesto que se traducen internamente a direcciones fáciles de localizar.
- Se logra independencia lógica y física, debido a que los valores de las llaves son independientes del espacio de direcciones.
- No se requiere almacenamiento adicional para los índices.

Implementación del código

A continuación, adjuraste capturas del código con una pequeña explicación de que hace cada cosa para mostrar como fue implementado el tema dado y el código que nos brindo la maestra para terminar cada clase con sus métodos completos.

Clase Películas

Declaración de la clase película, en esta primera clase se usaría la técnica de registros de longitud variable o delimitadores, así como nuestro primer acercamiento a los archivos o ficheros.

```
class Movie
{
    private:
        char name[64];
        char description[500];
        char category[15];
        char punctuation[10];

    public:
        Movie();
        Movie(char [64], char [500], char [15], char [10]);

        void setName(char [64]);
        void setDescription(char [500]);
        void setCategory(char [15]);
        void setPunctuation(char [10]);

        char* getName();
        char* getDescription();
        char* getCategory();
        char* getPunctuation();

        void captureData();
        void displayData();
        void findData();
        void deleteData();
        void modifyData();
};
```

Como podemos observar, la clase tiene los atributos de, nombre, descripción, duración, categoría y puntuación, así como una variable de tipo para indicar nuestro delimitador y sus métodos agrega, edita, elimina, imprime, busca.

```
void Movie::captureData()
{
    cout << "Dame el nombre de la pelicula: ";
    cin.ignore();
    cin.getline(name, 64); //EN LUGAR DE FFLUSH X QUE NO SIRVE CON GETLINE PARA LEER CADENAS

    cout << "Dame la descripcion: ";
    cin.getline(description,500);

    cout << "Dame el categoria: ";
    cin.getline(category,15);

    cout << "Dame la puntuacion: ";
    cin.getline(punctuation,10);

    ofstream file("Movie.txt",ios::app); // ofstream crea un objeto para escritura en el archivo

    file << name << '|' << description << '|' << category << '|' << punctuation << '|'; // se e
    file.close(); // cerrar el archivo
}
```

Para capturar la información, abrimos el archivo para crearlo y después de capturar cada dato, le concatenamos el carácter “|” que será nuestro delimitador de campo.

```
void Movie::displayData()
{
    int i;
    ifstream file("Movie.txt"); // abrir archivo en formato de lec
    if(!file.good()) // good ( libreria fstream) nos verifica si h
    {
        cout<<"No existe el archivo\n";
        system("pause");
    }
    else
    {
        while(!file.eof()) // eof = end of file
        {
            //Lectura de la variable user
            i=0;
            do
            {
                file.read((char *)name[i],1); //leo el objeto de
                if(file.eof()) break;
                i++;
            }while(name[i-1]!='|'); // mientras user en la posicio
            name[i-1]='\0'; // si no se cumple el while anterior,
            //Lectura de la variable pass
        }
    }
}
```

Para leer la información con un ciclo while hacemos un recorrido de todo el archivo y usamos la función read para que lea el objeto, y cuando encuentre el primer delimitador, se guardó lo leído en esa variable y así hasta el fin del archivo. Al final solo imprime todo lo leído ya que fue en orden, así como se insertaron.

```
void Movie::findData()
{
    int b = 0; // Bandera
    int i; // Contador
    int iguales;
    char caracter;
    char aux[64];

    ifstream file("Movie.txt"); /* Abrimos el archivo */
    if(!file.good()) /* Verificamos si el archivo existe */
    {
        cout<<"\n NO EXISTE ARCHIVO...\n";
    }

    else{
        cout<<"\n Escribe el nombre del producto a buscar: ";
        cin.ignore();
        cin.getline(aux,64);
    }
}
```

Para buscar, primero se pedía el nombre de la película que queríamos encontrar.

```
while(!file.eof()){
    i = 0;
    do{ // ciclo para leer cada uno de los campos
        file.read((char*)&caracter,1); // Leo 1 caracter y se lo asigno a la variable caracter
        if(caracter!='\n')
        {
            name[i] = caracter; // Le asigna al arreglo en la posicion i, el valor del caracter leído
            i++;
        }
    }while(caracter != '\n');

    name[i] = '\0'; // Finaliza la cadena despues del fin del atributo

    i = 0;
    do
    {
        file.read((char*)&caracter,1);
        if(caracter!='\n')
        {
            description[i] = caracter;
            i++;
        }
    }while(caracter != '\n');

    description[i] = '\0';
}
```

Se hacia un recorrido nuevamente por cada campo hasta el final del archivo y al final teníamos una bandera en donde si coincidía el nombre que nos dieron con alguno de los registros encontrados, significaba que lo habíamos encontrado, así que terminábamos ahí el ciclo e imprimíamos los datos que acabábamos de leer.

```
iguales = strcmp(name,aux);

if(iguales == 0){
    cout<<"\nNombre: " << name <<"\nDescripcion: " <<description <<"\nCategoria: " <<category<<"\nPuntuacion: "<<puntuati
    b = 1; // Activamos la bandera
    break;
}

b == 0){
    cout<<"\n\n NO EXISTE EL REGISTRO... \n\n";
}
```

Para la primera parte de eliminar, hacíamos el mismo algoritmo de buscar, pero cuando encontrábamos el registro que queríamos borrar, creábamos un archivo temporal en donde leíamos todos los datos de nuevo, pero cuando se leía el archivo a eliminar, este no se ponía en el txt, después solo teníamos que renombrarlo y de esta manera se borraba.

```
while(caracter != '|' && !file.eof()); // Validacion que se debe hacer con el ultimo campo
category[i]= '\0';

if(file.eof())
{
    break;
}

iguales = strcmp(name,aux);
if(iguales != 0){
    temporal <<name<<"|"<<description<<"|"<<category<<"|"<<punctuation<<"|";
}
} // while

file.close();
temporal.close();
remove("Movie.txt");
rename("temporal.txt","Movie.txt");
cout<<"\n\n EL REGISTRO FUE ELIMINADO... \n\n";
b = 1;
} // if
```


Por último, para modificar un registro, primero preguntábamos el nombre de la película que queríamos cambiar.

```
void Movie::modifyData()
{
    int i;//contador
    int opcion;
    char auxMovie[64];
    bool b;

    ifstream file("Movie.txt");/* Abrimos el archivo */
    ofstream tempFile("temporal.txt",ios::app);

    if(!file.good())
        cout<<"\nEL REGISTRO NO EXISTE \n ";
    else
    {
        cout<<"\nIntroduce el nombre de la pelicula modificar : ";
        cin.ignore();
        cin.getline(auxMovie,64);

        b = false;
    }
}
```

Leíamos todos los registros existentes y ya que encontráramos el registro que queríamos cambiar, preguntábamos cual campo sería, después de preguntarlo, pedíamos ese dato y en el archivo temporal escribíamos todos esos datos, pero con el que modificamos y solo tocaba renombrar y eliminar el anterior.

```
        cin.getline(category,15);
        break;
    case 4:
        cout<<"\n Puntuacion : ";
        cin.ignore();
        cin.getline(punctuation,10);
        break;
    default: cout<<"\n No se encontro la opcion ";
    }

    if(opcion!=6 && opcion>0 && opcion<7)
        b = true;

    tempFile<<name<<"|"<<description<<"|"<<category<<"|"<<punctuation<<"|";
    }
    file.close();
    tempFile.close();
    if(!b)
        cout<<"\n No se encontro archivo ";

    remove("Movie.txt");
    rename("temporal.txt","Movie.txt");
}
```

Clase Series de TV

Para la clase series, se utilizó la técnica de registros de longitud variable pero esta vez con campos de dimensión.

```
class Series
{
public:
    char codigo[10], nombre[100], temporadas[15], capitulos[15], descripcion[200], publicObj[15];
    char delim = '\n';
    bool checkID(char *);
    void Agregar();
    void Mostrar();
    void Modificar();
    void Eliminar();
    void Buscar();
    string getSerie(char _id[10]);
    void setters(char *, char *, char *, char *, char *, char *);
    void adminSeries();
    // void userSeries(char _id[15]);
}serie;
```

La clase tiene como atributos el código de la serie, nombre, temporadas, capítulos, descripción y publico objetivo, así como un carácter para indicar un delimitador y las funciones principales de agregar, mostrar, modificar, eliminar y buscar.

```
void Series::Agregar()
{
    bool verified = false;
    char _codigo[10], _nombre[100], _temporadas[15], _capitulos[15], _descripcion[200], _publicObj[15];
    do
    {
        cout << "ESCRIBE EL CODIGO DE LA SERIE: ";
        cin.getline(_codigo, 10);
        verified = checkID(_codigo);
        if(verified)
            cout << "Se encontraron coincidencias por favor intente con otro codigo" << endl;
    } while (verified);
    cout << "ESCRIBE EL NOMBRE DE LA SERIE: ";
    cin.getline(_nombre, 100);
    cout << "ESCRIBE LA DESCRIPCION DE LA SERIE: ";
    cin.getline(_descripcion, 200);
    cout << "ESCRIBE EL NUMERO DE TEMPORADAS: ";
    cin.getline(_temporadas, 15);
    cout << "ESCRIBE EL NUMERO DE CAPITULOS: ";
    cin.getline(_capitulos, 15);
    cout << "ESCRIBE EL PUBLICO OBJETIVO: ";
    cin.getline(_publicObj, 15);
    setters(_codigo, _nombre, _temporadas, _capitulos, _descripcion, _publicObj);
    ofstream Archivo("Series.txt", ios::app);
    Archivo.write((char *)&serie, sizeof(serie));
}
```

Para agregar una serie, primero realizamos una validación para que el código que den de la serie no este repetido regresando simplemente un booleano, describimos pedimos los datos de la serie, indicando el numero de caracteres máximo que se leerán en la función getline, todos esos datos los enviamos a una función para settearlos en el objeto y al final solo escribimos un nuevo registro en el txt que tendrá el tamaño dependiendo de la información que se leyó.

```
void Series::Mostrar()
{
    system("cls");

    ifstream archivo("Series.txt");
    if (!archivo.good())
    {
        cout << "\nEl archivo no existe...";
    }
    else
    {
        while (!archivo.eof())
        {
            archivo.read((char *)&serie, sizeof(serie)); // diml contiene el tamaño de la cadena que se quiere leer
            if (archivo.eof())
                break;
            cout << "Codigo: " << codigo << "\nNombre: " << nombre << "\nDescripción: " << descripcion << "\nTemporadas: " << temporadas << "\nCapítulos: "
                << endl;
        }
    }
    archivo.close();
}
```

El algoritmo para mostrar es muy sencillo, solo abrimos nuestro archivo en modo lectura y mientras no sea su final, leemos la cantidad de bytes que mide el objeto en sí, de esta manera el programa ya lo calcula y sabe cuanto debe de leer, por lo que solo toca estar imprimiendo hasta el final.

```
void Series::Modificar()
{
    //int band = 0;
    char codigo2[10];

    ifstream archivo("Series.txt");
    ofstream temporal("TempSeries.txt", ios::app);
    if (!archivo.good())
    {
        cout << "\nEl archivo no existe...";
    }
    else
    {
        cout << "escribe el código del producto: ";
        cin.getline(codigo2, 10);
        while (!archivo.eof())
        {
            archivo.read((char *)&serie, sizeof(serie)); // diml contiene el tamaño de la cadena que se quiere leer
            if (archivo.eof())
                break;
            if (strcmp(codigo2, codigo) == 0)
            {
                cout << "Codigo: " << codigo
                    << "\nNombre: " << nombre
                    << "\nDescripción: " << descripcion
                    << "\nTemporadas: " << temporadas
            }
        }
    }
}
```

Para modificar crearemos un archivo temporal y abriremos el principal, pediremos el código de la serie y leeremos hasta encontrar la serie que coincida con ese código. Imprimiremos para comprobar que esa es la serie que queremos modificar y después nos preguntara el campo que queremos modificar.

```
{
    cout << endl
        << "Codigo : " << codigo << endl
        << "Nombre : " << nombre << endl
        << "Descripcion : " << descripcion << endl
        << "#Temporadas #" << temporadas << endl
        << "#Capitulos #" << capitulos << endl
        << "Clasificacion : " << publicObj << endl
        << "    Que desea realizar ? " << endl
        << "(1) Modificar Nombre: " << endl
        << "(2) Modificar Descripcion: " << endl
        << "(3) Modificar # Temporadas: " << endl
        << "(4) Modificar # Capitulos" << endl
        << "(5) Modificar Clasificacion: " << endl
        << "(6) Regresar: " << endl
        << "Elige la opcion a realizar: ";
    cin >> opcion;
    cin.ignore();
    switch (opcion)
    {
    case 1: //////Nombre
        cout << "ESCRIBE EL NOMBRE DE LA SERIE: ";
        cin.getline(nombre, 100);
        break;
    case 2: /// Descripcion
        cout << "ESCRIBE LA DESCRIPCION DE LA SERIE: ";
        cin.getline(descripcion, 100);
        break;
    case 3: /// Temporadas
        cout << "ESCRIBE EL NUMERO DE TEMPORADAS: ";
        cin.getline(temporadas, 100);
        break;
    case 4: /// Capitulos
        cout << "ESCRIBE EL NUMERO DE CAPITULOS: ";
        cin.getline(capitulos, 100);
        break;
    case 5: /// Clasificacion
        cout << "ESCRIBE LA CLASIFICACION DE LA SERIE: ";
        cin.getline(publicObj, 100);
        break;
    case 6:
        break;
    }
}
```

En el archivo temporal escribiremos todos los datos anteriores y este nuevo modificado, al final solo lo renombramos y borramos el temporal.

```
    }
    } while (opcion != 6);
    }
    }
    temporal.write((char *)&serie, sizeof(serie));
}
temporal.close();
archivo.close();
remove("Series.txt");
rename("TempSeries.txt", "Series.txt");
}
```

Para eliminar nuevamente haremos una búsqueda dado un código

```
void Series::Eliminar()
{
    int band = 0;
    char codigo2[10];

    ifstream archivo("Series.txt");
    if (!archivo.good())
    {
        cout << "\nEl archivo no existe...";
    }
    else
    {
        cout << "CODIGO QUE QUIERES ELIMINAR: ";
        cin.getline(codigo2, 10);
        while (!archivo.eof() && !band)
        {
            archivo.read((char *)&serie, sizeof(serie)); // dimi contiene el tamaño de la cadena que se quiere leer
            if (archivo.eof())
                break;
            if (strcmp(codigo2, codigo) == 0)
            {
                cout << "Codigo: " << codigo << "\nNombre: " << nombre << "\nDescripcion: " << descripcion << "\nTemporadas: " << temporadas << "\nCapitulos: " << capitulos << "\n";
                << endl;
                band = 1;
                cout << "DESEAS ELIMINAR?\n1.- SI\n0.- NO\n> ";
                cin >> opc;
            }
        }
    }
}
```

Cuando coincida el código y el dato en el archivo, lo imprimiremos y preguntaremos si ese es el que queremos eliminar.

```
if (opc == 1)
{
    int i = 0;
    ifstream archivo("Series.txt");
    ofstream temporal("TempSeries.txt", ios::app);
    while (!archivo.eof())
    {
        archivo.read((char *)&serie, sizeof(serie));
        if (archivo.eof())
            break;
        if (strcmp(codigo2, codigo) != 0)
        {
            temporal.write((char *)&serie, sizeof(serie));
        } // condicion
        i++;
    } // ciclo while
    temporal.close();
    archivo.close();
    remove("Series.txt");
    rename("TempSeries.txt", "Series.txt");
} // condicional de la opcion
//
```

En caso de que si, crearemos de nuevo nuestro temporal y leeremos, cuando encontremos el dato a eliminar no haremos nada para que en ese archivo se copien todos excepto ese, después solo renombraremos ese temporal como el principal y eliminamos el otro archivo.

Para finalizar con la serie seguiremos con el de buscar el cual solo consiste en estar leyendo mientras no sea el final de archivo y hacer una comparación cuando el código que pedimos coincida con el de la serie, cuando esto suceda lo imprimiremos.

```
ifstream archivo("Series.txt");

if (!archivo.good())
{
    cout << "\nEl archivo no existe...";
}
else
{
    cout << "escribe el código del producto que quieres buscar: ";
    cin.getline(codigo2, 10);
    while (!archivo.eof() && !band)
    {
        archivo.read((char *)&serie, sizeof(serie)); // dimi contiene el tamaño de la cadena que se quiere leer
        if (archivo.eof())
            break;
        if (strcmp(codigo2, codigo) == 0)
        {
            cout << "Codigo: " << codigo << "\nNombre: " << nombre << "\nDescripcion: " << descripcion << "\nTemporadas: " << temporadas << "\nCapitulo: " << capitulo << endl;
            band = 1;
        }
    }
    if (!band)
```

```
string Series::getSerie(char _id[10])
{
    string N;
    bool band = false;
    ifstream archivo("Series.txt");
    if (!archivo.good())
    {
        cout << "\nEl archivo no existe...";
    }
    else
    {
        while (!archivo.eof() && !band)
        {
            archivo.read((char *)&serie, sizeof(serie)); // dimi contiene el tamaño de la cadena que se quiere leer
            if (archivo.eof())
                break;
            if (strcmp(_id, codigo) == 0)
            {
                N = serie nombre;
                band = true;
            }
        }
    }
    archivo.close();
    return N;
}
```

Clase Lista de Reproducción

Para esta parte se utilizaron dos clases, la de lista de reproducción y la de índice, ya que esta práctica fue con índices simples con TDA.

```
class ListaRepro
{
private:
    int indxContent = 0;
    char idUser[15];
    char idList[15], nameList[35], content[50][10];
    char delim = '\n';

public:
    void setList(char _idUser[15], char _idList[15], char _nameList[35]);
    void setContent(int space);
    void capturar(char _idUser[15]);
    void mostrar();
    void buscar();
    void Modificar();
    void Eliminar();
    void Mostrar_Indice();
    void Recovery();
    void ChangeCont();
    void menuL(char _id[15], bool _ad);
} listR;

class Indice
{
public:
    char idList[14];
    char idUser[15];
    char delim = '\n';
    long int pos;
} y;
```

```
void ListaRepro::capturar(char _idUser[15])
{
    cout << _idUser << endl;
    indxContent = 0;
    int _space;
    char _idList[15], _nameList[35];
    cout << "    AGREGAR LISTA DE REPRODUCCION" << endl;
    cout << "-----" << endl;
    cout << "Id de la lista:" << endl;
    cin.getline(_idList, 14);
    cout << "Dame el Nombre de la ListaRepro" << endl;
    cin.getline(_nameList, 35);
    setList(_idUser, _idList, _nameList);
    cout << "Cuántas id's agregará:" << endl;
    cin >> _space;
    cin.ignore();
    setContent(_space);
    cout << "-----" << endl;
    ofstream a("ListDatabase.txt", ios::app);
    // cout << y.pos << endl;
    if (cont)
    {
        y.pos = y.pos + sizeof(listR);
    }
    else
    {
        y.pos = a.tellp();
    }
    // cout << y.pos << endl;
    a.write((char *)listR, sizeof(listR));
    a.close();
    strcpy(y.idList, idList);
    strcpy(y.idUser, idUser);
}
```

Se pide el id que tendrá la lista, después su nombre y la cantidad de ids se agregaran, con la función de tellp nos vamos posicionando con el puntero dentro del txt para así manejar sus datos.

```
void ListaRepro::mostrar()
{
    ifstream a("ListDatabase.txt");
    if (!a.good())
        cout << "No existe el archivo";
    else
    {
        cout << "LISTAS DE REPRODUCCION EXISTENTES" << endl;
        cout << "-----" << endl;
        while (!a.eof())
        {
            a.read((char *)slistR, sizeof(listR));
            if (!a.eof())
                break;
            if (strcmp(listR.idUser, userlog.id) == 0 || userlog.ad == true)
            {
                if (userlog.ad == true)
                    cout << "UserID: " << listR.idUser << endl;
                cout << "ID:" << listR.idList << endl;
                cout << "Nombre: " << listR.nameList << endl;
                for (int i = 0; i < indrContent; i++)
                {
                    cout << "# " << i + 1 << ": " << listR.content[i] << endl;
                }
                cout << "-----" << endl;
            }
        }
        a.close();
    }
}
```

Para mostrar los datos, después de haber verificado que el archivo existe y se abrió bien, la recorreremos como registro de longitud fija e iremos comparando mediante la lista, ya que encontremos que su índice coincide, este tendrá su id y el nombre, así como la cantidad de ids que se fueron agregando a esa lista.

```
void ListaRepro::bucar()
{
    char idListbus[14];
    int band = 0;
    ifstream b("indexList.txt");
    if (!b.good())
    {
        cout << "No existe el archivo";
    }
    else
    {
        cout << "BUSCAR" << endl;
        cout << "-----" << endl;
        cout << "Vid de lista de Reproduccion a buscar!" << endl;
        cin.getline(idListbus, 14);
        while (!b.eof())
        {
            b.read((char *)y, sizeof(y));
            if (!b.eof())
                break;
            if (strcmp(y.idUser, userlog.id) == 0 || userlog.ad == true)
            {
                if (strcmp(y.idList, idListbus) == 0)
                {
                    ifstream a("ListDatabase.txt");
                    a.seekp(y.pos, ios::beg);
                    a.read((char *)slistR, sizeof(listR));
                    cout << "Lista de Reproduccion encontrada" << endl;
                    cout << "-----" << endl;
                    cout << "ID:" << listR.idList << endl;
                    cout << "Nombre:" << listR.nameList << endl;
                    for (int i = 0; i < indrContent; i++)
                    {
                        cout << "# " << i + 1 << ": " << listR.content[i] << endl;
                    }
                    cout << "-----" << endl;
                    a.close();
                    band = 1;
                    break;
                }
            }
        }
        if (band == 0)
    }
}
```


Para buscar un registro primero abrimos el archivo que trae todos los índices, buscamos el que coincida con el id que nos dieron, ya que encontramos el dato, ahora abrimos la lista que tendrá los datos, y dado el índice buscaremos el dato que coincida e imprimiremos de ahí su contenido.

El proceso para modificar un registro es algo largo, primero tendremos que tener nuestros archivos lógicos del índice y los datos, así como unos temporales para poder apoyarnos. Pediremos un ID, y empezaremos buscándolo en la lista de índices.

```
void ListaRepro::Modificar()
{
    char idListmod[15];
    int band = 0;
    int opc;
    ifstream indice("IndexList.txt");
    ifstream datos("ListDatabase.txt");
    ofstream c("temp.txt", ios::app);
    ofstream d("tempind.txt", ios::app);
    if (!indice.good())
    {
        cout << "No existe el archivo";
    }
    else
    {
        cout << "          MODIFICAR" << endl;
        cout << "-----" << endl;
        cout << "\nID de Lista de Reproduccion a modificar!" << endl;
        cin.getline(idListmod, 14);
        while (!indice.eof())
        {
```

Ya que lo encontramos, ahora dentro de ese ciclo buscaremos el dato dado el índice que obtuvimos.

```
        cin.getline(idListmod, 14);
        while (!indice.eof())
        {
            indice.read((char *)y, sizeof(y));
            datos.read((char *)listR, sizeof(listR));
            if (!indice.eof())
            {
                break;
            }
            if (strcmp(y.idUser, userlog.id) == 0 || userlog.ad == true)
            {
                if (strcmp(y.idList, idListmod) == 0 && band == 0)
                {
                    cout << " Lista de Reproduccion a Modificar" << endl;
                    cout << "-----" << endl;
                    cout << "ID:" << listR.idList << endl;
                    cout << "Nombre: " << listR.nameList << endl;
                    for (int i = 0; i < listR.content; i++)
                    {
                        cout << "i" << i + 1 << ": " << listR.content[i] << endl;
                    }
                    cout << "Deseas modificarlo\n";
                    cout << "1.-Si  2.-No" << endl;
                    cout << "Digite la opcion->";
                    cin >> opc;
```

```

int mod;
cout << "-----" << endl;
cout << "Que quiere modificar:" << endl
    << "1.-Nombre  2.- Contenido(" << indxContent << ")" << endl
    << "Digite la opcion-> ";
cin >> mod;
cout << "-----" << endl;
cin.ignore();
if (mod == 1)
{
    char _nameList[35];
    cout << "Dame el nuevo Nombre de la Lista de Reproduccion" << endl;
    cin.getline(_nameList, 35);
    listR.setList(listR.idUser, listR.idList, _nameList);
}

```

Cuando solo queremos cambiar el nombre de la playlist, simplemente pedimos el nombre y lo seteamos, al final solo reescribiremos.

```

cout << "-----" << endl;
cout << "Que Deseas hacer : " << endl
    << "1.- Insertar  2.- Modificar 3.- Eliminar 4.-Salir" << endl
    << "Digite la opcion-> ";
cin >> mod;
cout << "-----" << endl;
cin.ignore();
switch (mod)
{
    case 1:
    {
        char temp[10];
        if (indxContent == 10)
        {
            cout << "No se pueden ingresar mas ID's" << endl;
        }
        else
        {
            cout << "Cual es el Nuevo ID: ";
            cin.getline(temp, 10);
            strcpy(content[indxContent], temp);
            indxContent++;
        }
    }
    break;
    case 2:
    {
        if (!indxContent)
        {
            cout << "NO HAY DATOS" << endl;
        }
        else

```

Si queremos modificar parte del contenido, tendremos que usar otras opciones, tenemos un limite de 10 así que primero verificamos eso, podemos insertar, modificar o eliminar el contenido, así que pediremos ese ID.

Con otros ciclos for, volveremos a recorrer la lista de índices y después la de datos para poder modificarlos.

```

if (!indxContent)
{
    cout << "NO HAY DATOS" << endl;
}
else
{
    int line;
    for (int i = 0; i < indxContent; i++)
    {
        cout << "pos[" << i << "] = " << listR.content[i] << endl;
    }
    cout << "-----" << endl;
    cout << "Que posicion desea Eliminar: ";
    cin >> line;
    cin.ignore();
    if (indxContent < 10)
    {
        for (int i = line; i < indxContent; i++)
        {
            strcpy(content[i], content[i + 1]);
        }
    }
    indxContent--;
}

```

Por último, ya que se hicieron todos esos cambios, escribiremos en los archivos temporales la nueva información, y apoyándonos en estos los renombraremos y eliminaremos los registros que no se modificaron para solo tener los que tuvieron cambios.

```

c.write((char *)&listR, sizeof(listR));
strcpy(y.idList, idList);
d.write((char *)&y, sizeof(y));
} // while archivo
if (band == 0)
{
    cout << "\n NO EXISTE REGISTRO.....";
}
} // else
d.close();
c.close();
indice.close();
datos.close();
cout << "*****" << endl;
cout << "          GUARDANDO CAMBIOS..... " << endl;
cout << "    POR FAVOR NO CIERRE EL PROGRAMA " << endl;
cout << "*****" << endl;
remove("IndexList.txt");
rename("tempind.txt", "IndexList.txt");
remove("ListDatabase.txt");
rename("temp.txt", "ListDatabase.txt");
}

```

El eliminar es muy similar al modificar, empezaremos pidiendo el id de la lista a eliminar y la buscaremos primero en el archivo de índices, ya que lo encontramos, leeremos el archivo de datos en base a ese ID y mostraremos el dato para ver si es ese el que quiere eliminar.

```

cout << "                ELIMINAR                " << endl
<< "-----" << endl;
cout << " ID de Lista de Reproduccion a eliminar!" << endl;
cin.getline(idListeli, 14);
while (!b.eof())
{
    b.read((char *)sy, sizeof(y));
    contl++;
    if (b.eof())
    {
        break;
    }
    if (strcmp(y.idUser, userlog.id) == 0 || userlog.ad == true)
    {
        if (strcmp(y.idList, idListeli) == 0)
        {
            ifstream a("ListDatabase.txt");
            a.seekg(y.pos, ios::beg);
            a.read((char *)listR, sizeof(listR));
            if (a.eof())
            {
                break;
            }
            cout << " Lista de Reproduccion a Eliminar: " << endl
            << "-----" << endl;
            cout << "ID:" << listR.idList << endl
            << "Nombre: " << listR.nameList << endl;
            for (int i = 0; i < indkContent; i++)
            {
                cout << "# " << i + 1 << ": " << listR.content[i] << endl;
            }
            cout << "Deseas eliminarla" << endl
            << "1.-Si   2.-No" << endl
            << "Digite la opcion-> ";

```

Ya que nos aseguramos de que ese es la lista que queremos eliminar, copiamos la información dentro de los archivos temporales, pero cuando una lista coincida con el ID que le dimos, ese no se copiara, por último, renombramos esos archivos y eliminamos los anteriores.

```

void ListaRepro::Mostrar_Indice()
{
    ifstream a("IndexList.txt");
    if (!a.good())
        cout << "No existe el archivo";
    else
    {
        while (!a.eof())
        {
            a.read((char *)sy, sizeof(y));
            if (a.eof())
            {
                break;
            }
            cout << "                Lista de Indices                " << endl
            << "-----" << endl;
            cout << "Indice:" << y.idList << endl
            << "Posicion en elArchivo: " << y.pos << endl
            << endl;
        }
    }
    a.close();
}

```

```

while (!indice.eof())
{
    indice.read((char *)&y, sizeof(y));
    datos.read((char *)&listR, sizeof(listR));
    if (indice.eof())
    {
        break;
    }
    if (strcmp(y.idUser, userlog.id) == 0 || userlog.ad == true)
    {
        if (strcmp(y.idList, idListeli))
        {
            cont2++;
            ofstream c("temp.txt", ios::app);
            c.write((char *)&listR, sizeof(listR));
            strcpy(y.idList, idList);
            if (tempPos < y.pos)
            {
                y.pos = y.pos - sizeof(listR);
            }
            ofstream d("tempind.txt", ios::app);
            d.write((char *)&y, sizeof(y));
            d.close();
            c.close();
        }
    }
} // while archive
indice.close();
datos.close();
remove("IndexList.txt");
rename("tempind.txt", "IndexList.txt");
remove("ListDatabase.txt");

```

```

void ListaRepro::menuL(char _id[15], bool _ad)
{
    cout << _id << endl;
    userlog.submit(_id, _ad);
    int opc;
    do
    {
        system("cls");
        cout << "    Menu de Lista de Reproduccion    " << endl
             << "-----" << endl;
        cout << "Selecciona una de la siguientes opciones:" << endl
             << "1.- Agregar" << endl
             << "2.- Buscar" << endl
             << "3.- Mostrar todas listas" << endl
             << "4.- Modificar" << endl
             << "5.- Eliminar" << endl
             << "6.- Mostrar Indice" << endl
             << "7.- Salir" << endl
             << "Digite la opcion-> ";
        cin >> opc;
        cin.ignore();
        cout << "-----" << endl;
        switch (opc)
        {

```

Clase Pagos

Para la clase pago tendremos 4 atributos: ID, numero de tarjeta, fecha y cantidad a pagar, así como un número que será el índice después de hacer dispersión o hashing.

```
class Pago
{
private:
    char id[12];
    char tarjeta[35];
    char fecha[35];
    char cantidad[10];

    int dispersion(const char llave[12]);
    long int buscarId(const string &);

public:
    Pago(void);
    Pago(const Pago &);
    void setId(const string &);
    void setTarjeta(const string &);
    void setFecha(const string &);
    void setCantidad(const string &);
    friend ostream &operator<<(ostream &, const Pago &);

    bool agregar(const Pago &);
    void mostrar(void);
    bool buscar(const string &, Pago &);
    bool eliminar(const string &, Pago &);
    bool modificar(const string &, const Pago &);
    void mostrarIndice(void);
    Pago pedirRegistro(void);
    bool contiene(const string &);
    void genera(void);
    void AdminPay();
};
```

Para agregar un pago, pediremos el ID y verificaremos que sea valido o que exista, después pediremos los demás datos y usaremos sus setters para irlos agregando a un objeto que retornaremos con toda esta información.

```

Pago Pago::pedirRegistro(void)
{
    Pago registroARetornar;
    string cadena;

    cout << endl
         << "ID: ";
    fflush(stdin);
    getline(cin, cadena);
    while (contiene(cadena) || cadena.size() == 0)
    {
        cout << "Ese ID ya existe o la cadena es invalida: ";
        fflush(stdin);
        getline(cin, cadena);
    }
    registroARetornar.setId(cadena);

    cout << "Tarjeta: ";
    fflush(stdin);
    getline(cin, cadena);
    registroARetornar.setTarjeta(cadena);

    cout << "Fecha: ";
    fflush(stdin);
    getline(cin, cadena);
    registroARetornar.setFecha(cadena);

    cout << "Cantidad: ";
    fflush(stdin);
    getline(cin, cadena);
    registroARetornar.setCantidad(cadena);
}

```

Este es el algoritmo que realiza la dispersión, recibe una cadena y retornara la posición en donde se almacenara ese dato.

```

int Pago::dispersion(const char llave[12])
{
    // llena la el sobrante de la llave con espacios
    char llaveCopia[12];
    strcpy(llaveCopia, llave);
    if (strlen(llaveCopia) < 12)
        for (int i = strlen(llaveCopia); i < 12; i++)
            llaveCopia[i] = ' ';
    llaveCopia[12] = '\0';

    // realiza el algoritmo
    long sum = 0;
    int j = 0;
    while (j < 12)
    {
        sum = (sum + 100 * llaveCopia[j] + llaveCopia[j + 1]) % 20000;
        j += 2;
    }
    return (sum % 99); // retorna la posición donde se guardará el registros.
}

```

```

bool Pago::agregar(const Pago &nuevoPago)
{
    Pago pago;
    string cadena;
    int posIndice, contador;
    long int posByte;
    string idString = nuevoPago.id;

    if (contiene(idString))
        return false;

    fstream archivo("Payments.txt", ios::in | ios::out);
    posIndice = dispersion(nuevoPago.id);
    cout << "Se guardara en la posicion indice: " << posIndice << endl;
    posByte = posIndice * (sizeof(Pago) * CONTENEDOR + sizeof(int));
    archivo.seekg(posByte, ios::beg);
    archivo.read((char *)&contador, sizeof(int)); // lee el numero de registros en el contador
    if (contador < CONTENEDOR) // si el contenedor no esta lleno
    {

```

Tendremos una función booleana para saber si se agrego un pago, tenemos primero que verificar que el id que se recibió no exista, después con un contador verificamos el numero de registros y si el contador no está lleno.

Con la función seekg nos estaremos posicionando dentro del txt de pagos, así como regresar un falso cuando no haya más espacio para más.

```

        if (contador < CONTENEDOR) // si el contenedor no esta lleno
        {
            // aumenta el contador y lo escribe
            contador++;
            archivo.seekg(posByte, ios::beg);
            archivo.write((char *)&contador, sizeof(int));

            // escribe el nuevo registro en el contenedor
            for (int i = 0; i < CONTENEDOR; i++)
            {
                archivo.read((char *)&pago, sizeof(Pago));
                if (pago.id[0] == '\\0')
                {
                    archivo.seekg((long)archivo.tellg() - sizeof(Pago), ios::beg);
                    archivo.write((char *)&nuevoPago, sizeof(Pago));
                    archivo.close();
                    return true;
                }
            }
        }
        else
            cout << endl
                << "No hay mas espacio para este registro" << endl;
        archivo.close();
        return false;
    }
}

```



```

ifstream archivo("Payments.txt", ios::in);
if (!archivo)
    cout << "No existe el archivo" << endl;
else
{
    cout << endl;
    for (int i = 0; i < NUMREGISTROS; i++)
    {
        archivo.read((char *)&contador, sizeof(int));
        if (contador > 0)
        {
            for (int j = 0; j < CONTENEDOR; j++)
            {
                archivo.read((char *)&pago, sizeof(Pago));
                if (pago.id[0] != '\0'){
                    cout << "ID: " << pago.id << endl
                        << "Tarjeta: " << pago.tarjeta << endl
                        << "Fecha vencimiento: " << pago.fecha << endl
                        << "Monto: " << pago.cantidad << endl;
                }
            }
        }
    }
}

```

Para mostrar solo abrimos el archivo en modo lectura, leemos hasta el máximo numero de registros que tengamos y después al contenedor, de ahí vamos leyendo por la longitud del registro y vamos imprimiendo los datos.

```

bool Pago::buscar(const string &idABuscar, Pago &pagoEncontrado)
{
    long int posByte;

    if (!contiene(idABuscar))
        return false;

    ifstream archivo("Payments.txt", ios::in);
    if (!archivo)
    {
        cout << "El archivo no existe" << endl;
        archivo.close();
        return false;
    }

    posByte = buscarId(idABuscar);
    archivo.seekg(posByte, ios::beg);
    archivo.read((char *)&pagoEncontrado, sizeof(Pago));
    archivo.close();
    return true;
}

```

Para buscar un pago, tendremos que enviar el id y el objeto si es que lo encontró, si este existe, abrimos el archivo en modo lectura y la función de buscarID nos regresa la posición en donde se encuentra después del hash, así que solo nos posicionamos ahí y leemos el registro que haya en esa posición.

```

bool Pago::modificar(const string &idAModificar, const Pago &pagoNuevo)
{
    Pago registroLimpio, pago;
    int posIndiceAntiguo, posIndiceNuevo, contador;
    long int posByteAntiguo, posByteNuevo;

    if (!contiene(idAModificar))
        return false;

    fstream archivo("Payments.txt", ios::in | ios::out);
    if (!archivo)
    {
        cout << "El archivo no existe" << endl;
        archivo.close();
        return false;
    }

    posIndiceAntiguo = dispersion(idAModificar.c_str());
    posByteAntiguo = buscarId(idAModificar);
    posIndiceNuevo = dispersion(pagoNuevo.id);
    posByteNuevo = posIndiceNuevo * (sizeof(Pago) * CONTENEDOR + sizeof(int));
}

```

Para modificar un registro tendremos que crear varias variables para almacenar la posición del índice y del byte en donde estará el registro a modificar.

```

if (posByteAntiguo == posByteNuevo)
{
    archivo.seekg(posByteAntiguo, ios::beg);
    archivo.write((char *)&pagoNuevo, sizeof(Pago));
}
else
{
    // quita el registro antiguo y resta uno al contador del contenedor
    archivo.seekg(posByteAntiguo, ios::beg);
    archivo.write((char *)&registroLimpio, sizeof(Pago));
    posByteAntiguo = posIndiceAntiguo * (sizeof(Pago) * CONTENEDOR + sizeof(int));
    archivo.seekg(posByteAntiguo, ios::beg);
    archivo.read((char *)&contador, sizeof(int));
    contador--;
    archivo.seekg(posByteAntiguo, ios::beg);
    archivo.write((char *)&contador, sizeof(int));

    // intenta meter el nuevo registro en la nueva posicion
    archivo.seekg(posByteNuevo, ios::beg);
    archivo.read((char *)&contador, sizeof(int));
}

```

Cuando el registro arroja la misma posición, es muy sencillo porque solo sobrescribimos sobre ese, pero cuando no lo es, tendremos que quitarlo y auxiliarnos de un contador del contenedor.

Todo esto para intentar meter el nuevo registro dentro de la nueva posición.

```
if (contador < CONTENEDOR)
{
    // aumenta el contador y lo escribe
    contador++;
    archivo.seekg(posByteNuevo, ios::beg);
    archivo.write((char *)&contador, sizeof(int));

    // escribe el nuevo registro en el contenedor
    for (int i = 0; i < CONTENEDOR; i++)
    {
        archivo.read((char *)&pago, sizeof(Pago));
        if (pago.id[0] == '\0') // si el lugar no esta ocupado
        {
            archivo.seekg((long)archivo.tellg() - sizeof(Pago), ios::beg);
            archivo.write((char *)&pagoNuevo, sizeof(Pago));
            archivo.close();
            return true;
        }
    }
}
else // el contenedor esta lleno
    return false;
}
```

Así que, si el contador es menor que el máximo del contenedor, vamos posicionándonos para poder escribir el índice y leyendo la cantidad de bytes del objeto para así estar modificando los registros, si el contenedor esta lleno al calcular de nuevo la posición, retornaremos un false como fracaso al momento de agregarlo.

```
bool Pago::eliminar(const string &idAEliminar, Pago &pagoEliminado)
{
    Pago pago;
    int posIndice, posByte, contador;

    if (!contiene(idAEliminar))
        return false;

    fstream archivo("Payments.txt", ios::in | ios::out);
    if (!archivo)
    {
        cout << "El archivo no existe" << endl;
        archivo.close();
        return false;
    }

    posIndice = dispersion(idAEliminar.c_str());
    posByte = buscarId(idAEliminar);

    archivo.seekg(posByte, ios::beg);
    archivo.read((char *)&pagoEliminado, sizeof(Pago));
    archivo.seekg(posByte, ios::beg);
    archivo.write((char *)&pago, sizeof(Pago));

    posByte = posIndice * (sizeof(Pago) * CONTENEDOR + sizeof(int));
    archivo.seekg(posByte, ios::beg);
    archivo.read((char *)&contador, sizeof(int));
    contador--;
    archivo.seekg(posByte, ios::beg);
    archivo.write((char *)&contador, sizeof(int));
}
```

Si el pago a eliminar registro, posicionaremos nuestros enteros posIndice y posByte en la posición en el registro para de ahí leer y sobrescribir sobre los txts, auxiliándonos del contador y calculando la cantidad exacta de el objeto por el entero y el contenedor.

Y así se vería el menú para administrar los pagos:

```
void Pago::AdminPay()
{
    Pago pago, pagoBuscar, pagoModificar, pagoEliminar, registroAgregar;
    string idABuscar, idAModificar, idAEliminar;
    int opcion, op;
    system("cls");

    do
    {
        cout << endl << "\t\tBienvenido al menu de Pagos" << endl;
        cout << "Seleccione una opcion" << endl
             << "1. Agregar" << endl
             << "2. Mostrar" << endl
             << "3. Buscar" << endl
             << "4. Modificar" << endl
             << "5. Eliminar" << endl
             << "6. Salir" << endl
             << endl;
        cin >> opcion;
        switch (opcion)
        {
            case '1':
```

Clase Historial

```
class Historial
{
public:
    int indxContent = 0;
    char userID[15];
    char tipo[50];
    char content[50][10];
    char delimitador = '\n';

    void setList(char _userID[15]);
    void setContent(char _id[15], char _content[10], char tipo);
    void capturar();
    void mostrar();
    void getHistorial(char _id[15]);
    void buscar();
    void Modificar();
    void Eliminar();
    int hAdmin();
    void hUser(char _id[15], bool _ad);
} historial;
```

La clase historial posee como atributos un índice, el id del usuario, el tipo de historial, un contenido y un delimitador, así como los métodos que considere importante y los requeridos como capturar, mostrar, buscar, modificar y eliminar. En esta clase se hará uso de la serialización, es por esa razón que se trabaja con archivos binarios.

Como el historial trabaja con lo que se supone que vio el usuario, para agregar solo tenemos que escribir en el archivo ese contenido.

```
void Historial::capturar()
{
    ofstream b("History.bin", ios::binary | ios::app);
    indxContent = 0;
    char _userID[15];

    cout << "    AGREGAR HISTORIAL" << endl;
    cout << "-----" << endl;
    cout << "Id del usuario:" << endl;
    cin.getline(_userID, 14);
    setList(_userID);
    cout << "-----" << endl;
    b.write((char *)historial, sizeof(historial));
    b.close();
    cout << "*****" << endl;
    cout << "    GUARDANDO CAMBIOS..... " << endl;
    cout << "    POR FAVOR NO CIERRE EL PROGRAMA " << endl;
    cout << "*****" << endl;
}
```

Para mostrar los historiales tenemos que abrir el archivo binario en modo lectura y hacemos un ciclo hasta que sea el fin del archivo.

```
{
    ifstream a("History.bin");
    if (!a.good())
        cout << "No existe el archivo" << endl;
    else
    {
        cout << "HISTORIALES EXISTENTES" << endl
            << "-----" << endl;
        while (!a.eof())
        {
            a.read((char *)historial, sizeof(historial));
            if (a.eof())
                break;
            cout << "ID:" << historial.userID << endl;
            cout << "N : Tipo / ID" << endl;
            for (int i = 0; i < indxContent; i++)
            {
                cout << "#" << i + 1 << ": " << historial.tipo[i] << " / " << historial.content[i] << endl;
            }
            cout << "-----" << endl;
        }
    }
    a.close();
}
```

En la búsqueda tenemos que leer hasta el final del archivo y pedir el ID del historial que en este caso será el mismo que el del usuario y ya que coincida, imprimimos los datos leídos y su contenido.

```
{
    char userIDbus[14];
    int band = 0;
    ifstream a("History.bin");
    if (!a.good())
    {
        cout << "NO EXISTE EL ARCHIVO" << endl;
    }
    else
    {
        cout << "\tBUSCAR" << endl
            << "-----" << endl;
        cout << "\nID DE HISTORIAL (*ES EL MISMO DEL USUARIO*)" << endl;
        cin.getline(userIDbus, 14);
        while (!a.eof())
        {
            a.read((char *)historial, sizeof(historial));
            if (a.eof())
            {
                break;
            }
            if (strcmp(historial.userID, userIDbus) == 0)
            {
                cout << " HISTORIAL DEL ID #" << historial.userID << " ENCONTRADO" << endl
                    << "-----" << endl;
                cout << "N : Tipo / ID" << endl;
                for (int i = 0; i < indxContent; i++)
                {
                    cout << "#" << i + 1 << ": " << historial.content[i] << endl;
                }
                cout << "-----" << endl;
                band = 1;
            }
        }
    }
}
```

```

        {
            cout << "CUAL ES EL NUEVO ID: ";
            cin.getline(temp, 10);
            strcpy(content[indxContent], temp);
            indxContent++;
        }
    }
    break;
    case 2:
    {
        if (!indxContent)
        {
            cout << "NO HAY DATOS" << endl;
        }
        else
        {
            int line;
            char temp[10];
            for (int i = 0; i < indxContent; i++)
            {
                cout << "POS[" << i << "] = " << historial.content[i] << endl;
            }
            cout << "-----" << endl;
            cout << "QUE POSICION DESEA MODIFICAR: ";
            cin >> line;
            cin.ignore();

```

El modificar consiste en pedir el ID del historial a modificar, ya sea para insertar un nuevo id del programa, eliminarlo o solo cambiarlo, cuando esto se hace, se copian los datos nuevos y se sobrescriben auxiliándonos de un archivo temporal.

```

        }
        c.write((char *)&historial, sizeof(historial));
    } // while archivo
    if (band == 0)
    {
        cout << "\n NO EXISTE REGISTRO....." << endl;
    }
} // else
c.close();
datos.close();
cout << "*****" << endl;
cout << "      GUARDANDO CAMBIOS..... " << endl;
cout << "    POR FAVOR NO CIERRE EL PROGRAMA " << endl;
cout << "*****" << endl;
remove("History.bin");
rename("temp.bin", "History.bin");

```

Para eliminar se pedirá nuevamente el ID, de ahí haremos una búsqueda por longitud fija dentro del archivo, y ya que encontremos el dato que coincide preguntaremos si ese es el que queremos eliminar, Si es así, se activara una bandera en donde escribiremos los datos leídos en un archivo temporal

exceptuando el que quisimos eliminar, después solo lo renombramos y eliminamos el primer archivo.

```
        << "Digite la opcion-> ";
        cin >> opc;
        band = true;

        } // if de comparacion
        if (opc == 1)
        {
            // cont--;
        }
        else
        {
            temp.write((char *)&historial, sizeof(historial));
        }
    } // while

    if (!band)
    {
        cout << "\n NO EXISTE REGISTRO....." << endl;
    }
} // else
a.close();
temp.close();
remove("History.bin");
rename("TempHistory.bin", "History.bin");
```

Por último, así se vería el menú:

```
int Historial::hAdmin()
{
    int opc;
    do
    {
        system("cls");
        cout << "    MENU DEL HISTORIAL    " << endl
            << "-----" << endl;
        cout << "Selecciona una de la siguientes opciones:" << endl
            << "1.- Agregar" << endl
            << "2.- Buscar" << endl
            << "3.- Mostrar todas listas" << endl
            << "4.- Modificar" << endl
            << "5.- Eliminar" << endl
            << "6.- Salir" << endl
            << "Digite la opcion-> ";
        cin >> opc;
        cin.ignore();
        cout << "-----" << endl;
```


Clase Usuario y Encriptación

Decidí explicar esta clase hasta el final ya que es la que necesita todas las clases anteriores, también es la que maneja más métodos además de los obligatorios.

```
class Users
{
private:
    char id[15];
    char userName[30];
    char password[35];
    char delim = '\n';
    string adminPassword = "admin"; //Contraseña para logearse como admin
```

Como podemos observar, tenemos de atributos un ID, el nombre de usuario, su contraseña, un carácter para identificar el delimitador y una contraseña que será manejada por el administrador.

```
public:
    bool admin = false;
    void setters(char *_id, char *_user, char *_password);
    void setUser(char *_user);
    void setPass(char *_password);
    void setID(char *_id);
    string getUser();
    bool checkUser(char *_user);
    bool checkID(char *_id);
    void Agregar();
    void Mostrar();
    void Modificar();
    void Buscar();
    void Eliminar();
    bool checkLog();
    bool checkAdminPassword(const string&);
    void LogIn();
    void ver(bool tipo);
    void userMovies();
    void userSeries();
    void userPagos();
    void menuAdmin();
    void menuUser();
    void getHistorial(char _id[15]);
    void showHistorial();
    char *_cifrar(char *, int);
    char *_descifrar(char *, int);
} OrdCom;
```

Estos son sus métodos, algunos son para validar las contraseñas, otros para los diferentes menús que se manejarán y al final vemos los que nos cifrarán y descifrarán los datos.

Para la parte de encriptación, decidimos implementar un método cesar, en donde recibe una key que será el número de desplazamientos que se le harán a la cadena y de esta manera queda encriptada.

```
char *Users::cifrar(char *t, int clave)
{
    int size = strlen(t);

    for (int i = 0; i < size; i++)
    {
        t[i] += clave;
    }

    return t;
}
```

Para descifrar el mensaje es hacer el mismo proceso, pero a la inversa, es decir, recibimos la cadena cifrada, y la recorreremos, pero ahora a la izquierda, solo recibiendo la key, pero de manera negativa.

```
char *Users::descifrar(char *t, int clave)
{
    return cifrar(t, -clave);
}
```

Ya entrando en la parte de usuario, primero es importante validar que el usuario a ingresar no exista, entonces con una función booleana y una bandera revisaremos en el archivo que no existan coincidencias, de esta manera validamos que si pueda ser ingresado. Este mismo proceso se realizará con el ID.

```

bool Users::checkUser(char *user)
{
    bool band = false;
    string buffer1;
    string buffer2;
    buffer1 = user;
    ifstream archivo("Users.txt", ios::in);
    if (!archivo.good())
    {
        cout << "No se encontro el archivo" << endl;
    }
    else
    {
        while (!archivo.eof())
        {
            archivo.read((char *)OrdCom, sizeof(OrdCom));
            descifrar(userName, c1);
            buffer2 = userName;
            if (archivo.eof())
            {
                break;
            }
            if (buffer1 == buffer2)
            {
                cout << "Se encontraron coincidencias" << endl;
                band = true;
            }
        }
    }
    archivo.close();
    return band;
}

```

```

bool Users::checkID(char *_id)
{
    bool band = false;
    string buffer1;
    string buffer2;
    buffer1 = _id;
    ifstream archivo("Users.txt", ios::in);
    if (!archivo.good())
    {
        cout << "No se encontro el archivo" << endl;
    }
    else
    {
        while (!archivo.eof())
        {
            archivo.read((char *)OrdCom, sizeof(OrdCom));
            descifrar(id, c1);
            buffer2 = id;
            if (archivo.eof())
            {
                break;
            }
            if (buffer1 == buffer2)
            {
                cout << "Se encontraron coincidencias" << endl;
                band = true;
            }
        }
    }
    archivo.close();
    return band;
}

```

Para agregar un usuario, se pide el ID, el nombre y se realizan las validaciones anteriores, finalmente se le pide una contraseña y esta será cifrada, todos estos datos los almacenaremos en el objeto mediante una función setter y se escribirán en el archivo mediante registros de longitud fija.

```

cin.ignore();
cin.getline(buffer, 15);
verifyA = checkID(buffer);
if (verifyA)
{
    cout << "ID repetido intente con otro..." << endl;
}
while (verifyA);
cifrar(buffer, c1);
do
{
    cout << "Username: ";
    cin.getline(buffer1, 30);
    verifyB = checkUser(buffer1);
    if (verifyB)
    {
        cout << "Username repetido intente con otro..." << endl;
    }
} while (verifyB);
cifrar(buffer1, c1);
cout << "Password: ";
cin.getline(buffer2, 35);
cifrar(buffer2, c1);
setters(buffer, buffer1, buffer2);
OrdCom.admin = false;
ofstream archivo;
archivo.open("Users.txt", ios::app);
archivo.write((char *)OrdCom, sizeof(OrdCom));
archivo.close();

```

Para mostrar a los usuarios abriremos el archivo en modo lectura y con un ciclo estaremos leyendo cada registro, pero para poder mostrarlo en la pantalla de manera correcta, tendremos que mandar a llamar a la función de descifrar desde antes para que en consola se vean como se escribieron.

```
if (!archivo.good())
{
    cout << "\n El archivo no existe....." << endl;
}
else
{
    cout << "USUARIOS EXISTENTES" << endl;
    while (!archivo.eof())
    {
        archivo.read((char *)sOrdCom, sizeof(OrdCom));
        if (archivo.eof())
        {
            break;
        }
        descifrar(id, cl);
        cout << "ID: " << id << endl;
        descifrar(userName, cl);
        cout << "Usuario: " << userName << endl;
        descifrar(password, cl);
        cout << "Password: " << password << endl;
        cout << "" << endl;
    }
    archivo.close();
}
```

En la parte de modificar solo se podrá cambiar el nombre de usuario y la contraseña, después de haber leído el archivo y encontrar el registro a modificar dado el ID, se pedirán los datos que se quieren cambiar.

```
    cout << "Nuevo Username: ";
    cin.getline(aux, 30);
    verify = checkUser(aux);
    if (verify)
    {
        cout << "Username repetido intente con otro..." << endl;
    }
    while (verify);
    setUser(cifrar(aux, cl));
    modic = true;
    modicl = true;
}
break;
case 2:
{
    char aux[35];
    char aux2[35];
    bool verify = false;
    do
    {
        cout << "Nuevo Password: ";
        cin.getline(aux, 35);
        cout << "Confirmar Password: ";
        cin.getline(aux2, 35);
        if (strcmp(aux, aux2) == 0)
        {
            break;
        }
    } while (verify);
    setPassword(cifrar(aux, cl));
    modic = true;
    modicl = true;
}
break;
```

En caso de ser la contraseña, se pedirá confirmarla y si se equivoca después de varios intentos, tendrá que regresar e intentarlo de nuevo, si la modificación pudo proceder, entonces se escribirá ese dato y los anteriores en un archivo temporal, después se renombrará y eliminaremos el archivo anterior.

```
    }  
    }  
    else  
    {  
        cout << "Se acabaron los intentos" << endl;  
        archivo2.close();  
        remove("TempUser.txt");  
        return;  
    }  
    cout << "Debug: " << userName << " / " << password << endl;  
    ofstream archivo("Users.txt", ios::app);  
    archivo2.write((char *)&OrdCom, sizeof(OrdCom));  
    }  
    else  
    {  
        setID(cifrar(id, cl));  
        setUser(cifrar(userName, cl));  
        setPass(cifrar(password, cl));  
        archivo2.write((char *)&OrdCom, sizeof(OrdCom));  
    }  
    }  
    archivo.close();  
    archivo2.close();  
    remove("Users.txt");  
    rename("TempUser.txt", "Users.txt");  
}
```

Para eliminar un usuario pediremos el id del usuario a eliminar, después de recorrer el archivo, imprimiremos el usuario que coincida con ese id para verificar que ese es el que queremos borrar.

```
    }  
    else  
    {  
        cout << "ELIMINAR USUARIO" << endl;  
        cout << "Ingrese el ID a eliminar: ";  
        cin.getline(valor, 15);  
        ofstream archivo2("TempUser.txt", ios::app);  
        while (!archivo.eof())  
        {  
            archivo.read((char *)&OrdCom, sizeof(OrdCom));  
            descifrar(id, cl);  
            if (archivo.eof())  
            {  
                break;  
            }  
            if (strcmp(id, valor) == 0)  
            {  
                cout << "\tUsuario Encontrada" << endl;  
                cout << "ID: " << id << endl;  
                descifrar(userName, cl);  
                cout << "Fecha: " << userName << endl;  
                descifrar(password, cl);  
                cout << "Password: " << password << endl;  
                cout << "DESEAS ELIMINAR?\n1.- SI\n0.- NO\n>: ";  
                cin >> opcion;  
            }  
        }  
    }  
}
```

```

    }
    else
    {
        cifrar(id, cl);
        cifrar(userName, cl);
        cifrar(password, cl);
        archivo2.write((char *)&OrdCom, sizeof(OrdCom));
    }
}
else
{
    cifrar(id, cl);
    archivo2.write((char *)&OrdCom, sizeof(OrdCom));
}
}
archivo.close();
archivo2.close();
remove("Users.txt");
char oldname[] = "TempUser.txt";
char newname[] = "Users.txt";
rename(oldname, newname);

```

Ya que estemos sobre el dato a eliminar, escribiremos los demás en un archivo temporal, eliminamos el archivo principal, renombramos el temporal como el principal y así nos quedamos con un solo archivo.

```

ifstream archivo("Users.txt", ios::in);
if (!archivo.good())
{
    cout << "\n El archivo no existe....." << endl;
}
else
{
    cout << "BUSCAR USUARIO" << endl;
    cout << "Ingrese el ID a buscar: ";
    cin.getline(valor, 15);
    while (!archivo.eof())
    {
        archivo.read((char *)&OrdCom, sizeof(OrdCom));
        descifrar(id, cl);
        if (archivo.eof())
        {
            break;
        }
        if (strcmp(id, valor) == 0)
        {
            find = true;
            cout << "\tUsuario Encontrada" << endl;
            cout << "ID: " << id << endl;
            descifrar(userName, cl);
            cout << "Fecha: " << userName << endl;

```

Al momento de buscar abriremos el archivo en modo lectura, ingresaremos el ID del usuario a buscar y estaremos leyendo los datos, inmediatamente desciframos el ID y cuando este coincida con el que enviamos, descifraremos los demás campos y los mostraremos en consola.

Estas son las opciones que tendría el menú cuando nos loggemos como admin.

```
void Users::menuAdmin()
{
    Pago adminPay;
    int op;
    do
    {
        system("cls");
        cout << "MENU DE ADMIN" << endl;
        cout << "1.-Agregar" << endl;
        cout << "2.-Imprimir Usuarios" << endl;
        cout << "3.-Modificar " << endl;
        cout << "4.-Eliminar" << endl;
        cout << "5.-Buscar " << endl;
        cout << "6.-Películas" << endl;
        cout << "7.-Series" << endl;
        cout << "8.-Listas de reproduccion" << endl;
        cout << "9.-Historiales de usuario" << endl;
        cout << "10.-Pagos" << endl;
        cout << "11.-Cerrar Sesión " << endl;
        cout << "Ingresa la opción->";
        cin >> op;
        cin.ignore();
        switch (op)
```

Y el menú del usuario normal tendrá las siguientes opciones, como podemos ver, está más restringido.

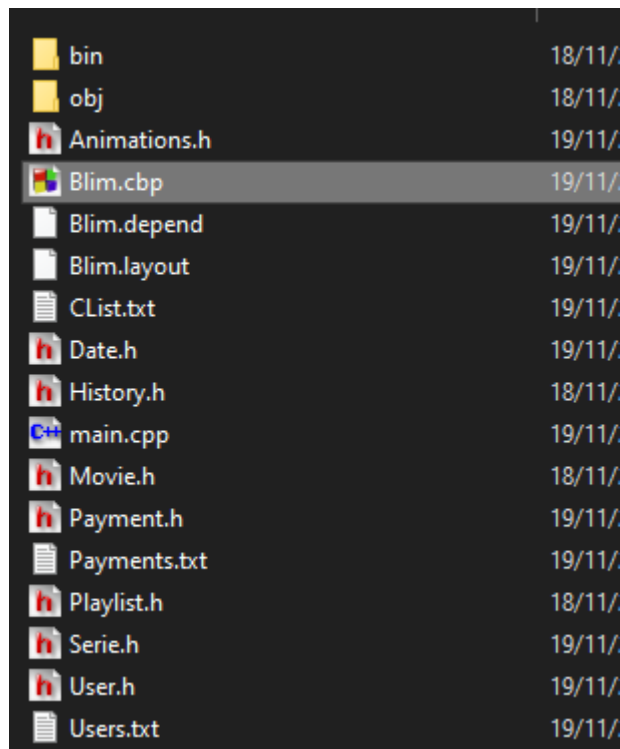
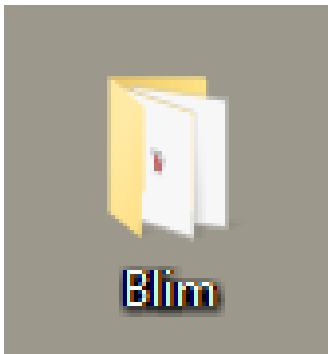
```
void Users::menuUser()
{
    Pago userPay;
    int op;
    do
    {
        system("cls");
        cout << "BIENVENIDO " << userLog.userName << endl;
        cout << "1.-Ver Películas" << endl;
        cout << "2.-Ver Series" << endl;
        cout << "3.-Listas de Reproduccion " << endl;
        cout << "4.-Historial" << endl;
        cout << "5.-Buscar " << endl;
        cout << "6.-Pago" << endl;
        cout << "7.-Configuración cuenta" << endl;
        cout << "8.-Cerrar Sesión " << endl;
        cout << "Ingresa la opción->";
        cin >> op;
        cin.ignore();
        switch (op)
```

Instrucciones de uso

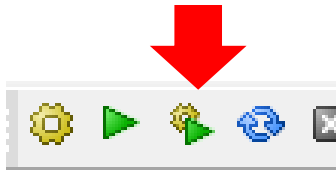
A continuación, explicare cómo funciona este programa, así como las funciones que realiza y como es el correcto funcionamiento para evitar problemas y/o errores durante su uso.

¿Cómo correr el programa?

Usted recibirá un archivo .zip que traerá todo el programa comprimido, una vez descomprimido tendrá la siguiente carpeta, dentro de ella abrirá el archivo con nombre “Blim.cbp”, es importante que usted tenga ya descargado e instalado el programa CodeBlocks versión 20.03. Si no lo tiene, puede comunicarse con Yazman para que le pueda brindar el archivo, dejaremos su información de contacto en el apartado de “Acerca de nosotros”.



Ya que abra el programa, diríjase a la parte superior, aquí encontrará 4 botones, el que se usará para correr el programa será el tercero (indicado con una flecha)



Y de la siguiente manera

Si ha seguido las indicaciones hasta el momento, se abrirá una ventana en donde se presenta el programa, la fecha, la hora y tendrá que esperar a que cargue.



A continuación, tendrá 2 ventanas, en donde podrá seleccionar la opción para poder entrar, ya sea como administrador o usuario.



A continuación, explicare el funcionamiento de las dos partes del programa

Lado Administrador

Si seleccionamos la primera opción, nos pedirá inmediatamente que ingresemos la contraseña, por default nosotros configuramos la contraseña como admin, pero si usted desea cambiarla puede comunicarse con nosotros para realizar la modificación, es importante que escriba la contraseña correcta porque si no será imposible entrar a esta parte.



Al entrar, estas serán las opciones que tenemos disponibles siendo administrador, básicamente es el control total de la aplicación y sus usuarios.

```
MENU DE ADMIN
1.-Agregar
2.-Imprimir Usuarios
3.-Modificar
4.-Eliminar
5.-Buscar
6.-Películas
7.-Series
8.-Listas de reproduccion
9.-Historiales de usuario
10.-Pagos
11.-Cerrar Sesion
Ingresa la opcion->
```

La parte de agregar se refiere a agregar un nuevo usuario, pero desde el menú de administrador, se solicitará ID, nombre y contraseña de el usuario que queremos agregar.

```
GENERAR USUARIO NUEVO
ID: 123
Username: Juanito
Password: contra
Presione una tecla para continuar . . .
```

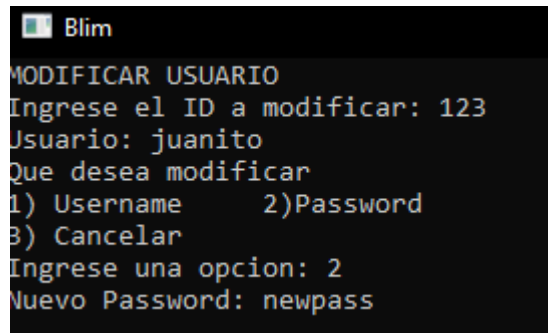
Con la segunda opción que es la de mostrar usuarios, podremos visualizar todos los usuarios suscritos, ya sea los que se agregaron desde el menú de admin o los que se registraron.

```
USUARIOS EXISTENTES
ID: 123
Usuario: juanito
Password: pass

ID: 321
Usuario: pepe
Password: word

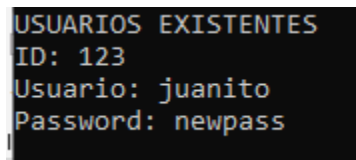
Presione una tecla para continuar . . .
```

La opción de modificar nos pedirá el ID del usuario, después imprimirá su nombre y nos preguntara que queremos modificar, en este caso seleccionare la contraseña.



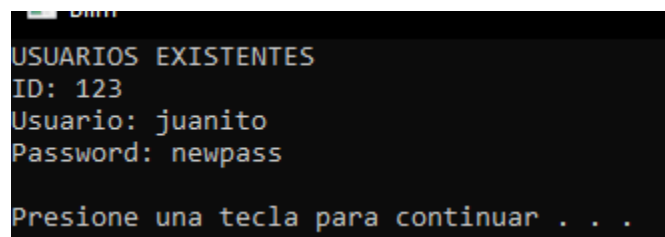
```
Blim
MODIFICAR USUARIO
Ingrese el ID a modificar: 123
Usuario: juanito
Que desea modificar
1) Username      2) Password
3) Cancelar
Ingrese una opcion: 2
Nuevo Password: newpass
```

Y al imprimir de nuevo, vemos que se modificó correctamente ese campo.



```
USUARIOS EXISTENTES
ID: 123
Usuario: juanito
Password: newpass
```

Al eliminar un usuario se nos pedirá el ID, después un mensaje para confirmar si realmente ese es el que queremos eliminar, y al seleccionar que si e imprimir, vemos que se eliminó correctamente.



```
Blim
USUARIOS EXISTENTES
ID: 123
Usuario: juanito
Password: newpass
Presione una tecla para continuar . . .
```

Finalmente para buscar un usuario, escribimos su ID, y si es que existe, nos imprimirá todos sus campos.

```
BUSCAR USUARIO
Ingrese el ID a buscar: 123
      Usuario Encontrada
ID: 123
Fecha: juanito
Password: newpass
Presione una tecla para continuar . . .
```

Con la opción número 6, entraremos a un submenú que será el de películas, aquí podremos realizar las mismas acciones.

```
Ingresa el numero de la opcion:
1.Capturar
2.Imprimir todo
3.Buscar
4.Eliminar
5.Modificar
6.Salir
1

Codigo: 9050

Nombre: Evangelion

Descripcion: Robots peleando

Time: 1:20

Clasificacion: R
```

La opción de capturar nos pedirá el código, el nombre, una pequeña descripción, duración y la clasificación de la película.

Con la opción de imprimir todo, se verán todos los registros de las películas existentes.

```

PELICULAS EXISTENTES

Codigo: 9050
Nombre: Evangelion
Descripcion: Robots peleando
Duracion: 1:20
Clasificacion: R

Codigo: 2022
Nombre: One Piece
Descripcion: Morra cantando
Duracion: 1:50
Clasificacion: A
Presione una tecla para continuar . . .

```

Para buscar una película solo tenemos que poner su código y nos imprimirá sus campos en caso de que esta si exista.

```

Escribe el codigo de la Pelicula a buscar
9050

Codigo: 9050
Nombre: Evangelion
Descripcion: Robots peleando
Duracion: 1:20
Clasificaion: R
Presione una tecla para continuar . . .

```

Para eliminar tendremos que escribir nuevamente el código, imprimirá sus datos si existe y nos preguntara si realmente queremos eliminar, al dar que sí, e imprimir de nuevo veremos que el registro fue eliminado.

```

Introduce el codigo a Eliminar :
2022
Se encontro una coincidencia

Codigo: 2022
Nombre: One Piece
Descripcion: Morra cantando
Duracion: 1:50
Clasificaion: A
Quiere eliminarlo?
(1) SI (2) NO
->

```

```

2
PELICULAS EXISTENTES

Codigo: 9050
Nombre: Evangelion
Descripcion: Robots peleando
Duracion: 1:20
Clasificacion: R
Presione una tecla para continuar . . .

```

Para modificar escribiremos el ID de la película a modificar, después lo buscaremos e imprimiremos el registro de esa película, a continuación se nos pregunta el campo que queremos modificar, escribimos el nuevo campo y al momento de imprimir podemos verificar que los cambios se realizaron de manera correcta.

```

PELICULAS EXISTENTES

Codigo: 9050
Nombre: Evangelion
Descripcion: Fin del mundo
Duracion: 1:20
Clasificacion: R
Presione una tecla para continuar . . .

```

Ahora pasaremos a la parte de series, para agregar una nueva serie tendremos que escribir los siguientes datos.

```

                                MENU SERIES ADMIN

                                SELECCIONE LA OPCION DESEADA
1.- AGREGAR
2.- MOSTRAR
3.- BUSCAR
4.- MODIFICAR
5.- ELIMINAR
6.- SALIR
>: 1
ESCRIBE EL CODIGO DE LA SERIE: 3015
ESCRIBE EL NOMBRE DE LA SERIE: The Office
ESCRIBE LA DESCRIPCION DE LA SERIE: Oficina chistosa
ESCRIBE EL NUMERO DE TEMPORADAS: 4
ESCRIBE EL NUMERO DE CAPITULOS: 10
ESCRIBE EL PUBLICO OBJETIVO: Adolescentes
Presione una tecla para continuar . . .

```

Agregare otra serie para poder observar que al momento de seleccionar la opción de mostrar se imprimirán todos los registros.

```
Codigo: 3015
Nombre: The Office
Descripcion: Oficina chistosa
Temporadas: 4
Capitulos: 10
Publico Objetivo: Adolescentes

Codigo: 8567
Nombre: Chainsaw Man
Descripcion: Pibe motosierra
Temporadas: 1
Capitulos: 12
Publico Objetivo: Adolescentes

Presione una tecla para continuar . . .
```

Para buscar escribimos el código de la serie e imprime el registro que coincida con ese código.

```
escribe el codigo del producto que quieres buscar: 3015
Codigo: 3015
Nombre: The Office
Descripcion: Oficina chistosa
Temporadas: 4
Capitulos: 10
Publico Objetivo: Adolescentes

Presione una tecla para continuar . . .
```

Al modificar, pasamos el código de la serie, imprime el registro que tenga ese código y nos preguntará el campo que queremos modificar, el programa seguirá así hasta que decidamos salir de ese menú.


```

>: 4
Escribe el codigo de la serie: 8567
Codigo: 8567
Nombre: Chainsaw Man
Descripcio: Pibe motosierra
Temporadas: 1
Capitulos: 12
Publico Objetivo: Adolescentes
DESEA MODIFICAR?
1.SI
0.NO
>: 1

Codigo : 8567
Nombre : Chainsaw Man
Descripcion : Pibe motosierra
#Temporadas #1
#Capitulos #12
Clasificacion : Adolescentes
Que desea realizar ?
(1) Modificar Nombre:
(2) Modificar Descripcion:
(3) Modificar # Temporadas:
(4) Modificar # Capitulos
(5) Modificar Clasificacion:
(6) Regresar:
Elige la opcion a realizar: 2
ESCRIBE LA DESCRIPCION DE LA SERIE: Power>>Makima

```

Al imprimir vemos como se realizó correctamente la modificación.

```

Codigo: 8567
Nombre: Chainsaw Man
Descripcio: Power>>Makima
Temporadas: 1
Capitulos: 12
Publico Objetivo: Adolescentes

```

Por último, para eliminar, escribimos el código de la serie que queremos borrar, imprimirá el registro para asegurarnos de que ese sea y seleccionamos que sí.

```

CODIGO QUE QUIERES ELIMINAR: 3015
Codigo: 3015
Nombre: The Office
Descripcio: Oficina chistosa
Temporadas: 4
Capitulos: 10
Publico Objetivo: Adolescentes

DESEAS ELIMINAR?
1.- SI
0.- NO
>: 1
Presione una tecla para continuar . . .

```

Aquí se observa que al imprimir de nuevo todos los registros, ya no esta el que acabamos de eliminar.

```
Codigo: 8567
Nombre: Chainsaw Man
Descripcio: Power>>Makima
Temporadas: 1
Capitulos: 12
Publico Objetivo: Adolescentes

Presione una tecla para continuar . . .
```

Seguimos ahora con la opción numero 8 que es la lista de reproducción, aquí al agregar algo a la lista de reproducción se pedirá el ID de la lista, después el nombre que tendrá la lista y por ultimo la cantidad de IDs de programas que se almacenaran, dependiendo de la cantidad seleccionada, preguntara el ID para cada una.

```
-----
                AGREGAR LISTA DE REPRODUCCION
-----
Id de la Lista:
5
Dame el Nombre de la ListaRepro
Comedia

Cuantas id's agregara:
2
Id #1: 2040
Id #2: 5090
-----
*****
                GUARDANDO CAMBIOS.....
                POR FAVOR NO CIERRE EL PROGRAMA
*****
Presione una tecla para continuar . . .
```

Para mostrar, elegimos la opción 2 y se verán los datos.

```
BUSCAR
-----
ID de Lista de Reproduccion a buscar!
5
  Lista de Reproduccion encontrada
-----
ID:5
Nombre: Comedia
#1: 2040
#2: 5090
-----
Presione una tecla para continuar . . .
```

A la hora de modificar, solicitara el ID de la lista, de ahí, podemos modificar ya sea el nombre de la playlist o su contenido, como podemos ver hay 2, en este ejemplo modificare el nombre.

```
MODIFICAR
-----
ID de Lista de Reproduccion a modificar!
5
  Lista de Reproduccion a Modificar
-----
ID:5
Nombre: Comedia
#1: 2040
#2: 5090
Deseas modificarlo
1.-Si  2.-No
Digite la opcion->1
-----
Que quiere modificar:
1.-Nombre  2.- Contenido(2)
Digite la opcion-> 1
-----
Dame el nuevo Nombre de la Lista de Reproduccion
Gracioso
```

Y al imprimir vemos que se ha hecho la modificación.

```
-----  
LISTAS DE REPRODUCCION EXISTENTES  
-----  
UserID:  
ID:5  
Nombre: Gracioso  
#1: 2040  
#2: 5090  
-----  
Presione una tecla para continuar . . .
```

Agregue otra lista de reproducción para que el ejemplo de eliminar sea más claro.

```
ID de Lista de Reproduccion a buscar!  
1  
  Lista de Reproduccion encontrada  
-----  
ID:1  
Nombre: 1  
#1: 1  
-----  
Desea eliminar?  
1)Si  
2)No
```

Escribimos el ID de la lista a eliminar y escribimos 1 para eliminarla, al imprimir de nuevo todas las listas, ya no aparecerá esa lista.

```
LISTAS DE REPRODUCCION EXISTENTES  
-----  
UserID:  
ID:5  
Nombre: Gracioso  
#1: 2040  
#2: 5090  
-----  
UserID:  
ID:99  
Nombre: Accion  
#1: 8321  
-----  
Presione una tecla para continuar . . .
```

Por último, hay que recordar que este programa tenía una función extra y era la de mostrar índices.

```
-----  
                Lista de Indices  
-----  
Indice:5  
Posicion en elArchivo: 607  
  
                Lista de Indices  
-----  
Indice:99  
Posicion en elArchivo: 1179  
  
Presione una tecla para continuar . . .
```

Como podemos observar, el índice es básicamente el ID que le dimos a esa lista, y después de aplicarle hashing, genero una posición que es donde se guardo el archivo, esta parte del proyecto se uso manejando 2 archivos txt.

Seguimos con la opción 9 que consiste en el menú del historial.

La función de agregar historial en si solo actualiza lo que un usuario ha visto, se ingresa el ID del usuario y se guarda.

```
-----  
                AGREGAR HISTORIAL  
-----  
Id del usuario:  
123  
-----  
*****  
                GUARDANDO CAMBIOS.....  
                POR FAVOR NO CIERRE EL PROGRAMA  
*****  
Presione una tecla para continuar . . .
```

```

BUSCAR
-----

ID DE HISTORIAL (*ES EL MISMO DEL USUARIO*)
123
  HISTORIAL DEL ID #123 ENCONTRADO
-----
N : Tipo / ID
#1: 9050
-----
Presione una tecla para continuar . . .

```

Para buscar el historial de un usuario, escribimos su ID e imprimirá el código del contenido que haya visto, si el usuario no ha visto nada, no aparecerá nada, esta función se encuentra el en lado del usuario que estará en el siguiente apartado.

Si mostramos todas las listas, ahí saldrá el tipo de contenido y el ID que ha visto. M se refiere a Movie.

```

-----
MODIFICAR
-----

ID DE HISTORIAL A MODIFICAR
123
  HISTORIAL ENCONTRADO
-----
ID:123
#1: 9050
DESEAS MODIFICARLO?
1.-SI  2.-NO
DIGITE LA OPCION->1
-----

QUE DESEAS HACER :
1.- INSERTAR  2.- MODIFICAR 3.- ELIMINAR 4.-SALIR
Digite la opcion-> 2
-----
POS[0] = 9050
-----
QUE POSICION DESEA MODIFICAR: 0
ID NUEVA:

```

Al modificar tendremos que escribir el ID del usuario, después aparecerán los IDs del contenido que haya visto, de ahí podemos elegir si insertar otro contenido, modificar uno existente o eliminarlo, como podemos ver aquí solo hay un elemento visto que está en la posición 0, entonces esa será la que modifiquemos. Y a continuación podemos ver como se ha modificado el ID.

```
HISTORIALES EXISTENTES
-----
ID:123
N : Tipo / ID --> Nombre
#1: M / 2022 -->
```

No imprime el nombre porque no hay ninguna película que tenga ese ID ya que solo teníamos una registrada, pero si lo hubiera, ahí se imprimiría.

Por último, el eliminar borra todo el historial, si quisiéramos eliminar solo un contenido, entonces tendríamos que irnos al apartado de modificar.

```
ELIMINAR
-----
ID de Lista de Reproduccion a eliminar!
123
Lista de Reproduccion a Eliminar
-----
ID:123
#1: 2022
Deseas eliminarla
1.-Si 2.-No
Digite la opcion->
```

Y al mostrar las listas, vemos que ya no hay ninguna.

```
HISTORIALES EXISTENTES
-----
Presione una tecla para continuar . . .
```

Para finalizar con el lado administrador del programa, tenemos el menú de pagos para poder controlar las suscripciones de cada usuario.

Este es el menú principal y para agregar un pago solo tenemos que ingresar el ID del usuario, su numero de tarjeta, la fecha y la cantidad, como este programa se realizo usando hashing o dispersión, arrojará la posición en donde se guardó el registro.

```

                                Bienvenido al menu de Pagos
Seleccione una opcion
1. Agregar
2. Mostrar
3. Buscar
4. Modificar
5. Eliminar
6. Salir
1
ID: 123
Tarjeta: 5022 4080
Fecha: 12/05/24
Cantidad: 300
Se guardara en la posicion indice: 40
Pago agregado con exito
```

Para mostrar la segunda opción, agregue otro pago y al momento de mostrar todos sale de la siguiente manera.

```

ID: 123
Tarjeta: 5022 4080
Fecha vencimiento: 12/05/24
Monto: 300
ID: 321
Tarjeta: 4156 6043
Fecha vencimiento: 05/07/22
Monto: 250
```


Para buscar un pago solo tenemos que escribir su ID y nos imprimirá los datos que le correspondan.

```
Ingrese el ID del pago a buscar: 321  
ID:      321  
Producto: 4156 6043  
Proveedor: 05/07/22  
Precio:   250
```

Para modificar un pago, escribimos el ID del pago a modificar, imprimirá el pago y nos preguntara si ese es el registro que queremos modificar, de ahí solo basta con darle a que sí y nos pedirá los nuevos datos que tendrá ese pago.

```
4  
Ingrese el ID del pago a buscar: 321  
ID:      321  
Producto: 5055 3030  
Proveedor: 17/09/24  
Precio:   250  
  
Desea modificarlo?  
1) Si  
2) No  
1  
  
ID: 555  
Tarjeta: 2022 5040  
Fecha: 15/10/23  
Cantidad: 150
```

Y al imprimir vemos que el cambio se realizó con éxito.

```
ID: 555  
Tarjeta: 2022 5040  
Fecha vencimiento: 15/10/23  
Monto: 150  
  
Presione una tecla para continuar . . .
```

Al eliminar se nos solicitara el ID del pago, después lo imprimirá y preguntara si realmente queremos eliminar ese registro.

```
5
Ingrese el ID del pago a eliminar: 555

ID:      555
Producto: 2022 5040
Proveedor: 15/10/23
Precio:   150

Desea eliminarlo?
1) Si
2) No
```

```
Blim

Presione una tecla para continuar . . .
```

Y como podemos observar, al imprimir los pagos ya no hay ninguno ya que fue eliminado el único que teníamos.

La última opción que tenemos es la numero 11 que básicamente consiste en cerrar la sesión y salir de ese menú, una vez realizada esta acción, regresaremos al menú principal, si queremos volver a entrar al menú de Amin tendremos que escribir la contraseña o aquí podemos irnos al menú de usuarios que es el que explicare a continuación.

Lado Usuario

Para entrar a este apartado no necesitaremos contraseña la primera vez, al entrar tendremos dos opciones, Iniciar sesión que es para usuarios registrados y registrarse, que es para los usuarios nuevos que quieran suscribirse al servicio.

Empecemos registrando un nuevo usuario, para ello seleccionaremos la opción número 2.

```
MENU DE USUARIOS
-----
Que desea realizar
1)Iniciar sesion
2)Registrarse
3)Salir
Seleccione una opcion -> 2
GENERAR USUARIO NUEVO
ID: 420
Username: cesarin
Password: hola

USUARIO AGREGADO CON EXITO !
Presione una tecla para continuar . . .
```

Con la opción de registrarse empezaremos creando un nuevo usuario, a este le asignaremos un ID, su nombre de usuario y la contraseña que tendrá su cuenta.

Ahora seleccionaremos la opción 1, que será con la que iniciaremos sesión.

```
MENU DE USUARIOS
-----
Que desea realizar
1)Iniciar sesion
2)Registrarse
3)Salir
Seleccione una opcion -> 1
INICIAR SESION
Username: cesarin
Password: hola
Presione una tecla para continuar . . .
```

Si nuestros datos fueron correctos, entonces veremos el siguiente menú que corresponde a las acciones que puede realizar el usuario.

```
Blim
BIENVENIDO cesarin
1.-Ver Peliculas
2.-Ver Series
3.-Listas de Reproduccion
4.-Historial
5.-Buscar
6.-Pago
7.-Configuracion cuenta
8.-Cerrar Sesion
Ingresa la opcion->
```

En la opción número 1 que es ver películas, si la seleccionamos tendremos otro submenú.

```
MENU PELICULAS
[1] CATALOGO
[2] BUSCAR
[3] VER
[4] SALIR
->
```

Si seleccionamos catálogo, podremos ver todas las películas que hay registradas (las que registro el administrador anteriormente)

```
-> 1
PELICULAS EXISTENTES

Codigo: 9050
Nombre: Evangelion
Descripcion: Fin del mundo
Duracion: 1:20
Clasificacion: R

Codigo: 592
Nombre: Avengers
Descripcion: Superheroes se unen
Duracion: 2:20
Clasificacion: ninios
Presione una tecla para continuar . . .
```

Si conocemos el código de la película, podemos usar la opción numero 2, anotar su código y nos imprimirá los datos correspondientes.

```
-> 2

Escribe el codigo de la Pelicula a buscar
9050

Codigo: 9050
Nombre: Evangelion
Descripcion: Fin del mundo
Duracion: 1:20
Clasificaion: R
Presione una tecla para continuar . . .
```

En la tercera opción, podremos ver algún contenido para ir generando nuestro historial, aquí podemos escribir el código de la película que vamos a ver o mostrar el catálogo completo y después escribir el código.

```
-> 3
Que deseas hacer?
[ 1 ] Ingresar ID a ver
[ 2 ] Mostrar catalogo
[ 3 ] Salir
-> 2
PELICULAS EXISTENTES

Codigo: 9050
Nombre: Evangelion
Descripcion: Fin del mundo
Duracion: 1:20
Clasificacion: R

Codigo: 592
Nombre: Avengers
Descripcion: Superheroes se unen
Duracion: 2:20
Clasificacion: ninios
Ingresa ID: 592
```

En la opción numero 2 podremos realizar lo mismo, pero con series de televisión.

```

                                MENU SERIES

                                SELECCIONE LA OPCION DESEADA
[1] CATALOGO
[2] BUSCAR
[3] VER
[4] SALIR
->
```

```

Codigo: 8654
Nombre: Chainsaw Man
Descripcion: power>>makima
Temporadas: 1
Capitulos: 12
Publico Objetivo: adolescentes

Codigo: 1523
Nombre: Ergo Proxy
Descripcion: proxys atacan
Temporadas: 1
Capitulos: 26
Publico Objetivo: adultos

Presione una tecla para continuar . . .
```

```

escribe el codigo del producto que quieres buscar: 8654
Codigo: 8654
Nombre: Chainsaw Man
Descripcion: power>>makima
Temporadas: 1
Capitulos: 12
Publico Objetivo: adolescentes

Presione una tecla para continuar . . .
```

```

Codigo: 8654
Nombre: Chainsaw Man
Descripcio: power>>makima
Temporadas: 1
Capitulos: 12
Publico Objetivo: adolescentes

Codigo: 1523
Nombre: Ergo Proxy
Descripcio: proxys atacan
Temporadas: 1
Capitulos: 26
Publico Objetivo: adultos

Ingresa ID: 1523

```

Aquí decidimos usar la opción de ver y estamos ingresando el ID después de mostrar el catálogo.

Los usuarios también tendrán la opción de crear sus propias listas de reproducción, esto lo realizaremos con la opción 3, al elegirla aparecerán las siguientes opciones.

```

Menu de Lista de Reproduccion
-----
Selecciona una de la siguientes opciones:
1.- Agregar
2.- Buscar
3.- Mostrar todas listas
4.- Modificar
5.- Eliminar
6.- Mostrar Indice
7.- Salir
Digite la opcion->

```

Empecemos agregando nuestra primera lista.

```

Digite la opcion-> 1
-----
420
      AGREGAR LISTA DE REPRODUCCION
-----
Id de la Lista:
777
Dame el Nombre de la ListaRepro
pendientes
420
420
Cuantas id's agregara:
1
Id #1: 8564

```

El numero que aparece en la esquina es la posición que tenemos reservada para guardar nuestros datos, le asignaremos un ID a la lista y le pondremos un nombre, también indicaremos la cantidad de ids o programas agregaremos.

Con la función de buscar, le pasaremos el ID que le pusimos a la lista y la imprimirá.

```
-----
BUSCAR
-----
ID de Lista de Reproduccion a buscar!
777
  Lista de Reproduccion encontrada
-----
ID:777
Nombre: pendientes
#1: 8564
-----
Presione una tecla para continuar . . .
```

La opción de mostrar todas, imprimirá todas las listas de reproducción que hayamos creado, como solo tenemos una, solo imprimirá esa.

```
LISTAS DE REPRODUCCION EXISTENTES
-----
ID:777
Nombre: pendientes
#1: 8564
-----
Presione una tecla para continuar . . .
```

Con la opción de modificar, se nos pedirá el ID de la lista, después de encontrarla, la imprimirá y nos dirá que parte queremos modificar si el nombre o el contenido, en el contenido hay un número, este indica la cantidad de IDs

que metimos a esa lista de reproducción, en este caso yo solo modificare el nombre.

```
-----  
MODIFICAR  
-----  
ID de Lista de Reproduccion a modificar!  
777  
Lista de Reproduccion a Modificar  
-----  
ID:777  
Nombre: pendientes  
#1: 8564  
Deseas modificarlo  
1.-Si 2.-No  
Digite la opcion->1  
-----  
Que quiere modificar:  
1.-Nombre 2.- Contenido(1)  
Digite la opcion-> 1  
-----  
Dame el nuevo Nombre de la Lista de Reroduccion  
para el rato
```

Y al imprimirla de nuevo, la modificación habrá sido realizada.

```
-----  
LISTAS DE REPRODUCCION EXISTENTES  
-----  
ID:777  
Nombre: para el rato  
#1: 8564  
-----  
Presione una tecla para continuar . . .
```

La opción de mostrar índice, nos imprimirá el ID que le dimos a la lista y la posición del índice en el que se guardó después de aplicarle hashing.

```
-----  
Lista de Indices  
-----  
Indice:5  
Posicion en elArchivo: 607  
  
Lista de Indices  
-----  
Indice:99  
Posicion en elArchivo: 1179  
  
Lista de Indices  
-----  
Indice:777  
Posicion en elArchivo: 1751
```

Para la opción de eliminar tenemos que escribir el ID de la lista que queremos borrar, después la imprimirá y nos preguntara si ese es el registro que queremos borrar, si es el caso, escribimos 1, y veremos que al imprimir ya se habrá eliminado.

```
ELIMINAR
-----
ID de Lista de Reproduccion a buscar!
777
  Lista de Reproduccion encontrada
-----
ID:777
Nombre: para el rato
#1: 8564
-----
Desea eliminar?
1)Si
2)No
1
```

```
7.- Salir
Digite la opcion-> 3
-----
No existe el archivoP
```

El archivo si existe, solo que ese es el mensaje cuando esta vacío

¿Recuerdan que en la primera parte podíamos ver películas o series? Pues esto era para poder ir generando nuestro historial, al escribir 4, se imprimirá los programas que hayamos visto anteriormente.

```
HISTORIAL DEL ID #420
-----
N : Tipo / ID --> Nombre
#1: M / 592 --> Avengers
#2: S / 1523 --> Ergo Proxy
-----
```

Aquí podemos observar que vimos una película (Tipo M) que fue Avengers, además vimos una serie (Tipo S) que fue Ergo Proxy, cada uno con sus respectivos IDs.

En la opción 5 seremos capaces de mostrar los pagos, para agregar un pago solo tenemos que ingresar el ID del usuario, su número de tarjeta, la fecha y la cantidad, como este programa se realizó usando hashing o dispersión, arrojará la posición en donde se guardó el registro.

```

                               Bienvenido al menu de Pagos
Seleccione una opcion
1. Agregar
2. Mostrar
3. Buscar
4. Modificar
5. Eliminar
6. Salir

1

ID: 123
Tarjeta: 5022 4080
Fecha: 12/05/24
Cantidad: 300
Se guardara en la posicion indice: 40
Pago agregado con exito
```

Para mostrar la segunda opción, agregue otro pago y al momento de mostrar todos sale de la siguiente manera.

Para buscar un pago solo tenemos que escribir su ID y nos imprimirá los datos que le correspondan.

```
Ingrese el ID del pago a buscar: 321
ID:      321
Producto: 4156 6043
Proveedor: 05/07/22
Precio:   250
```

Para modificar un pago, escribimos el ID del pago a modificar, imprimirá el pago y nos preguntara si ese es el registro que queremos modificar, de ahí solo basta con darle a que sí y nos pedirá los nuevos datos que tendrá ese pago.

```
4
Ingrese el ID del pago a buscar: 321
ID:      321
Producto: 5055 3030
Proveedor: 17/09/24
Precio:   250
Desea modificarlo?
1) Si
2) No
1
ID: 555
Tarjeta: 2022 5040
Fecha: 15/10/23
Cantidad: 150
```

```
ID: 123
Tarjeta: 5022 4080
Fecha vencimiento: 12/05/24
Monto: 300
ID: 321
Tarjeta: 4156 6043
Fecha vencimiento: 05/07/22
Monto: 250
```

Y al imprimir vemos que el cambio se realizó con éxito.

```
ID: 555
Tarjeta: 2022 5040
Fecha vencimiento: 15/10/23
Monto: 150
Presione una tecla para continuar . . .
```

Al eliminar se nos solicitara el ID del pago, después lo imprimirá y preguntara si realmente queremos eliminar ese registro.

```
5
Ingrese el ID del pago a eliminar: 555

ID:      555
Producto: 2022 5040
Proveedor: 15/10/23
Precio:   150

Desea eliminarlo?
1) Si
2) No
```

```
Blim

Presione una tecla para continuar . . .
```

Y como podemos observar, al imprimir los pagos ya no hay ninguno ya que fue eliminado el único que teníamos.

Tenemos una última opción en donde podemos configurar nuestra cuenta, en donde podemos cambiar nuestro username o nuestra contraseña.

Para este ejemplo optare por cambiar el nombre de usuario.

```
MODIFICAR USUARIO
Usuario: cesarin
Ingrese su password: hola
Que desea modificar
1) Username      2) Password
3) Cancelar
Ingrese una opcion: 1
Nuevo Username: acerrin
```

```
MENU DE USUARIOS
-----
Que desea realizar
1) Iniciar sesion
2) Registrarse
3) Salir
Seleccione una opcion -> 1
INICIAR SESION
Username: acerrin
Password: hola
```

Ahora intentare iniciar sesión con mi nuevo username y podemos ver que entre de manera correcta.

```
Blim
BIENVENIDO acerrin
1.-Ver Peliculas
2.-Ver Series
3.-Listas de Reproduccion
4.-Historial
5.-Pago
6.-Configuracion cuenta
7.-Cerrar sesion
Ingresa la opcion->
```

Recomendaciones para el uso correcto del sistema

- Evite escribir cadenas o títulos muy largos, como cada atributo fue realizado con arreglos de caracteres de un tamaño predeterminado, el superar la cantidad recomendada puede llegar a corromper los archivos o en su defecto, escribir datos basura
- La aplicación fue programada en Windows, para poder brindar una mejor experiencia de usuario se utilizó una librería exclusiva del sistema operativo, por lo que es probable que en otro SO el funcionamiento del programa y su apariencia se vea afectado
- Utilizar el IDE CodeBlocks versión 20.03 con GNU y el lenguaje C++
- Es probable que durante algunos procesos se creen archivos temporales dentro de la carpeta, es importante no abrirlas o borrarlos durante este proceso, además de no cambiar nombre o ubicación de los mismos para evitar posible pérdida de información

Acerca de nosotros

Nuestra misión es conseguir que cada cliente pueda conseguir una experiencia agradable a la hora de usar nuestro programa. ¿Quiénes son nuestros clientes? Los usuarios finales de nuestro software son de vital importancia para este proyecto de Blim, pero no son nuestros únicos clientes, además tenemos a los administrativos que serán los que manejen todas las demás funciones. Cada miembro de nuestra comunidad colectiva son referencias claves para nosotros, cuando ofrecemos los mejores resultados, no podemos estar equivocados.

Si hubiera algún problema con el programa puede contactarnos con los siguientes datos:

Conclusiones

La realización de este proyecto comenzó desde el inicio del semestre, en donde se nos propuso la idea y dimos el primer paso creando el diagrama UML, de ahí durante cada semana íbamos viendo un tema e implementando la clase correspondiente con la técnica que se pedía, algunos códigos fueron mas complicados que otros ya que la realización de estos era diferente y teníamos que tomar en cuenta algunas consideraciones. Hacer que las clases se conectaran entre si fue de las partes más difíciles como con el historial o las listas de reproducción, además en varias partes del código tuvimos que realizar ciertas validaciones, ya sea para evitar tener códigos repetidos o para prevenir colisiones. Personalmente, con lo que aprendimos en el semestre anterior pudimos realizar algunas pequeñas animaciones y diseños para hacer el programa mas intuitivo, si bien algunas partes se pudieron mejorar, consideramos que tenemos un programa sencillo e intuitivo con el usuario final.

Fue interesante y entretenida toda la creación de este proyecto, si bien, fue todo un reto, se pudo sacar adelante, esperemos que el resultado final sea el esperado por la maestra y podamos recibir una calificación gratificante, de momento nos quedamos con todo el conocimiento que nos brindo y con un proyecto mas que ira a nuestro portafolio de evidencias.

Bibliografía

- Rational Software Corporation et al. "UML 1.1 Documentation Set". <http://www.rational.com/uml>. 1 September 1997.
- García Peñalvo, Francisco José y Pardo Aguilar, Carlos. "UML 1.1. Un lenguaje de modelado estándar para los métodos de ADOO". RPP, No36, pp. 57-61. Enero, 1998.
- García Peñalvo, Francisco José y Pardo Aguilar, Carlos. "Introducción al Análisis y Diseño Orientado a Objetos". RPP, No37. Febrero, 1998.
- <http://centrodeartigos.com/articulos-para-saber-mas>
- [http://recursos.udgvirtual.udg.mx/biblioteca/bitstream/123456789/1726/1/Conceptos fundamentales de estructuras de archivos.pdf](http://recursos.udgvirtual.udg.mx/biblioteca/bitstream/123456789/1726/1/Conceptos_fundamentales_de_estructuras_de_archivos.pdf)
- Estructuras de archivos. - Michael J. Folk; Addison-Wesley Iberoamericana
- Estructuras de datos. - Osvaldo Cairo, Silvia Guardati; Mc Graw Hill