ALGORITMOS Y ESTRUCTURA DE DATOS

CURSO: K1024

PROFESOR: FEDERICO ANDRES MAIDAN

ASISTE LOS DÍAS: SÁBADOS

EN EL TURNO: MAÑANA

TRABAJO PRÁCTICO: v2

TÍTULO: TRABAJO PRÁCTICO v2

PARTICIPANTES DEL EQUIPO

GRUPO 14

<Achata, Mayerly >	<212.935-8 >	<machatafernandez@frba.utn.edu.ar>
<Cahuaya, Mishel >	<212.949-8 >	<scahuayacuno@frba.utn.edu.ar>
< Gómez, Carla>	<213.750-1 >	<carlgomez@frba.utn.edu.ar>
<Kindrat, Luciano >	<212.994-2 >	<lkindrat@frba.utn.edu.ar>
<Sanna, Ian>	<214.111-5>	<isanna@frba.utn.edu.ar>
<Tumiri, John Jairo>	<214.198-0>	< jtumiricuellar@frba.utn.edu.ar>

Gestor Estudiante

Encargado del sistema: Tumiri Jhon Jairo. Mishel Cahuaya.

Descripción del sistema: Las estructuras creadas son 10 Archivos físicos externos,que son las siguientes:

- **controlador.cpp:** Este archivo fuente, está definiendo las siguientes funciones:

➔ Iniciar sesión: Este código parece estar diseñado para permitir a un usuario iniciar sesión como estudiante al ingresar sus datos, como correo electrónico (o posiblemente otros detalles), y luego verificar si esos datos coinciden con los de un estudiante registrado. Si coinciden, se inicia sesión; de lo contrario, se muestra un mensaje de "CORREO INVALIDO".

```
void INICIARSESION()
{
    system("cls");
    Estudiante dataUser;
    dataUser = IngresarDatos();
    bool condition = false;
    dataUser = existeEstudiante(condition, dataUser);

    if (condition)
    {
        IniciarSesionEstudiante(dataUser);
    }
    else
    {
        cout << "CORREO INVALIDO" << endl;
    }
    system("pause");
}
```

Aquí está la descripción de lo que hace este código:

1) system("cls");: Esta línea utiliza la función system para borrar la pantalla de la consola. Esto proporciona una especie de limpieza visual antes de mostrar nuevos mensajes o resultados.

2) “Estudiante dataUser;”: Se declara una variable llamada “dataUser” del tipo Estudiante. Esto parece ser una instancia de una clase o estructura llamada Estudiante, que probablemente almacena información sobre un estudiante.

3) ”dataUser = IngresarDatos();”: Llama a una función llamada IngresarDatos() para obtener datos de un estudiante y asigna esos datos a la variable dataUser.

4) ”bool condition = false;”: Se declara una variable booleana llamada condition y se inicializa en false. Esta variable se utilizará más adelante para verificar si los datos del estudiante ingresado coinciden con un estudiante registrado.

5) “dataUser = existeEstudiante(condition, dataUser);”: Llama a una función llamada existeEstudiante() y le pasa como argumentos la variable condition y dataUser. Esta función probablemente verifica si los datos ingresados del estudiante coinciden con los de algún estudiante registrado y actualiza la variable

condition en consecuencia. Además, podría devolver la información completa del estudiante encontrado y asignarla nuevamente a dataUser.

6) El código a continuación verifica el valor de condition:
   - ❖ Si condition es true, llama a la función IniciarSesionEstudiante(dataUser) para iniciar sesión con el estudiante cuyos datos coinciden con los ingresados.
   - ❖ Si condition es false, muestra el mensaje "CORREO INVALIDO", lo que indica que los datos ingresados no coinciden con ningún estudiante registrado.

7) system("pause");: Espera a que el usuario presione una tecla antes de continuar. Esto se usa comúnmente para evitar que la consola se cierre inmediatamente después de que se muestren los resultados.

➔ Registrarse:Este código parece estar diseñado para registrar a un estudiante, pero primero verifica si ya existe un estudiante con la misma información antes de realizar el registro. Si ya existe, muestra un mensaje de error; de lo contrario, registra al estudiante y muestra un mensaje de éxito.

```
void INICIARSESION()
{
    system("cls");
    Estudiante dataUser;
    dataUser = IngresarDatos();
    bool condition = false;
    dataUser = existeEstudiante(condition, dataUser);

    if (condition)
    {
        IniciarSesionEstudiante(dataUser);
    }
    else
    {
        cout << "CORREO INVALIDO" << endl;
    }
    system("pause");
}
```

Aquí está la descripción de lo que hace este código:

1) system("cls");: Esta línea utiliza la función system para borrar la pantalla de la consola. Esto proporciona una especie de limpieza visual antes de mostrar nuevos mensajes o resultados.
2) Estudiante dataUser;: Se declara una variable llamada dataUser del tipo Estudiante. Esto parece ser una instancia de una clase o estructura llamada Estudiante, que probablemente almacena información sobre un estudiante.
3) bool condition = false;: Se declara una variable booleana llamada condition y se inicializa en false. Esta variable se usará más adelante para verificar si ya existe un estudiante con la misma información.
4) dataUser = CrearEstudiante();: Llama a una función llamada CrearEstudiante() para obtener datos de un estudiante y asigna esos datos a la variable dataUser.
5) existeEstudiante(condition, dataUser);: Llama a una función llamada existeEstudiante() y le pasa como argumentos la variable condition y dataUser. Esta función probablemente verifica si ya existe un estudiante con la misma información que dataUser y actualiza la variable condition en consecuencia.

6) El código a continuación verifica el valor de condition:
   - ❖ Si condition es true, muestra el mensaje "NO SE PUEDE REGISTRAR UN MISMO USUARIO", lo que indica que ya existe un estudiante con la misma información.
   - ❖ Si condition es false, llama a la función RegistrarEstudiante(dataUser) para registrar al estudiante y muestra el mensaje "ESTUDIANTE REGISTRADO CORRECTAMENTE"
7) system("pause");: Espera a que el usuario presione una tecla antes de continuar. Esto se usa comúnmente para evitar que la consola se cierre inmediatamente después de que se muestren los resultados.

➔ Salida: Este código muestra un mensaje indicando una salida exitosa, espera a que el usuario presione una tecla y luego cambia el valor de la variable condition a false. La modificación de condition a false puede ser utilizada para controlar la lógica de flujo en otras partes del programa, por ejemplo, para indicar que el programa debe salir o detener alguna operación específica.

```
void EXIT(bool &condition)
{
    cout << "SE SALIO EXITOSAMENTE" << endl;
    system("pause");
    condition = false;
}
```

Aquí está la descripción de lo que hace este código:

1) cout << "SE SALIO EXITOSAMENTE" << endl;: Esta línea muestra en la consola el mensaje "SE SALIO EXITOSAMENTE", lo que indica que el programa o la parte del programa está saliendo de manera exitosa. Esto es simplemente un mensaje informativo para el usuario.
2) system("pause");: Espera a que el usuario presione una tecla antes de continuar. Esto se usa comúnmente para evitar que la consola se cierre inmediatamente después de mostrar un mensaje, permitiendo que el usuario vea el mensaje antes de que el programa termine
3) condition = false;: Asigna el valor false a la variable booleana condition. Esto es importante porque es un parámetro pasado por referencia (&condition) a la función. Cambiar su valor a false puede utilizarse para comunicar a otras partes del programa que se debe salir o detener alguna operación.

➔ por defecto: Este código muestra un mensaje indicando que la opción seleccionada no existe o no es válida, espera a que el usuario presione una tecla y luego cambia el valor de la variable condition a false. La modificación de condition a false puede ser utilizada para controlar la lógica de flujo en otras partes del programa, por ejemplo, para manejar casos en los que una opción seleccionada no es válida.

```
void DEFAULT(bool &condition)
{
    cout << "ESTA OPCION NO EXISTE" << endl;
    system("pause");
    condition = false;
}
```

Aquí está la descripción de lo que hace este código:

1) cout << "ESTA OPCION NO EXISTE" << endl;: Esta línea muestra en la consola el mensaje "ESTA OPCION NO EXISTE". Este mensaje informa al usuario que la opción seleccionada no es válida o no existe en el programa.
2) system("pause");: Espera a que el usuario presione una tecla antes de continuar. Esto se usa comúnmente para evitar que la consola se cierre inmediatamente después de mostrar un mensaje, permitiendo que el usuario vea el mensaje antes de que el programa continúe.
3) condition = false;: Asigna el valor false a la variable booleana condition. Similar al ejemplo anterior, condition es un parámetro pasado por referencia (&condition) a la función. Cambiar su valor a false puede utilizarse para comunicar a otras partes del programa que ocurrió un error o que la opción seleccionada no es válida.

- **controlador.hpp:** Este archivo de encabezado establece las declaraciones de varias funciones, pero sus implementaciones deben estar en otro lugar (como en el archivo de código fuente "controlador.cpp"). Además, incluye el archivo "controlador.cpp" y utiliza #pragma once para evitar la inclusión múltiple del archivo de encabezado. Estas funciones parecen estar relacionadas con el control de sesiones y opciones en algún programa.

```
1    #pragma once
2    #include "controlador.cpp"
3
4    void INICIARSESION();
5    void REGISTRARSE();
6    void EXIT(bool &);
7    void DEFAULT(bool &);
```

Aquí está la descripción de lo que hace este código:

1) #pragma once: Esta directiva #pragma once se utiliza para prevenir la inclusión múltiple del archivo de encabezado en un programa. Cuando un archivo se incluye más de una vez en un programa, pueden surgir problemas de redefinición y aumento innecesario en el tiempo de compilación. #pragma once asegura que el archivo de encabezado se incluye solo una vez, lo que evita estos problemas.
2) #include "controlador.cpp: Esta línea incluye el archivo de código fuente "controlador.cpp" en el programa. Es importante destacar que es inusual incluir archivos de código fuente (.cpp) en archivos de encabezado (.h). Lo

común es incluir archivos de encabezado en otros archivos de código fuente para acceder a declaraciones de funciones y estructuras de datos.

3) void INICIARSESION();: Declaración de una función llamada INICIARSESION que no toma argumentos y no devuelve un valor (void). Esta declaración indica que existe una función llamada INICIARSESION en algún lugar del código, pero su definición (implementación) debe estar en otro lugar, como en el archivo de código fuente "controlador.cpp".
4) void REGISTRARSE();: Declaración de una función llamada REGISTRARSE que no toma argumentos y no devuelve un valor (void). Al igual que la función anterior, esta declaración indica que existe una función llamada REGISTRARSE en algún lugar del código, pero su definición debe estar en otro lugar.
5) void EXIT(bool &);: Declaración de una función llamada EXIT que toma un argumento por referencia, una variable booleana llamada condición. Esta función no devuelve un valor (void). Esta declaración indica que existe una función llamada EXIT que puede modificar la variable condition pasada como argumento.
6) void DEFAULT(bool &);: Declaración de una función llamada DEFAULT que toma un argumento por referencia, una variable booleana llamada condition. Esta función no devuelve un valor (void). Similar a la función EXIT, esta declaración indica que existe una función llamada DEFAULT que puede modificar la variable condition pasada como argumento.

- **interfaz.cpp:**Este código proporciona funcionalidades para que un usuario (posiblemente un estudiante) inicie sesión o se registre en una aplicación. Las funciones capturan y validan los datos del usuario, brindan la opción de corrección y devuelven un objeto Estudiante con los datos ingresados. Las opciones de menú se gestionan a través de la función NavegacionMain().

```cpp
#include "interfaz.hpp"
#include <iostream>
#include <string>
#include <string.h>
#include <conio.h>
using namespace std;
struct Estudiante
{
    char name[25];
    char lastname[25];
    char email[60];
    char password[60];
    int creditos;
    int Id;
};

int NavegacionMain()
{
    int number;
    cout << "====================" << endl;
    cout << "1: Iniciar Sesion" << endl;
    cout << "2: Registrarse" << endl;
    cout << "3: cerrar" << endl;
    cout << "--------------------" << endl;
    cout << "Para realizar una tarea ingresar el numero correspondiente" << endl;
    cout << "====================" << endl;
    cin >> number;
    return number;
};
```

1) Función NavegacionMain(): Esta función muestra un menú en la consola y permite al usuario seleccionar una opción ingresando un número. Las opciones incluyen Iniciar Sesión, Registrarse y Cerrar la aplicación. La función devuelve el número de opción seleccionado.

```
Estudiante IngresarDatos()
{

    Estudiante user;
    bool condition = true;
    int validacionOption;
    cout << "----------------------------------------" << endl;
    cout << "--------------- INICIAR SESION ----------" << endl;
    cout << "----------------------------------------" << endl;
    do
    {
        fflush(stdin);
        cout << "ingresar tu email: ";
        cin.getline(user.email, 60, '\n');

        cout << "ingresar tu contrasenia: ";
        cin.getline(user.password, 60, '\n');

        cout << "los datos son correctos?" << endl;
        cout << "1: SI" << endl;
        cout << "2: NO" << endl;
        cout << "----------------------------------------------------" << endl;
        cin >> validacionOption;

        switch (validacionOption)
        {
        case 1:
            cout << "CARGANDO.........." << endl;
            condition = false;
            break;
        case 2:
            cout << "RESET FORMULARIO" << endl;
            break;
cinco   default:
            cout << "ESTA OPCION NO EXISTE \n SE RESETEARA LOS DATOS INGRESADOS" << endl;
            break;
        }
    } while (condition);

    return user;
}
```

2) Función IngresarDatos(): Esta función solicita al usuario que ingrese su correo electrónico y contraseña para iniciar sesión. Luego, le pregunta al usuario si los datos son correctos y permite confirmar o volver a ingresarlos. La función devuelve un objeto Estudiante con los datos ingresados.

```
Estudiante CrearEstudiante()
{
    Estudiante newUser;
    bool condition = true;

    int option;
    cout << "-----------------------------------------------------" << endl;
    cout << "Para registrarte ingresa tus datos en la parte de abajo" << endl;
    do
    {
        // funcion que sirve para liberar espacio en el buffer (buffer: espacio de memoria en el que se almacenan datos de manera temporal.)
        fflush(stdin);
        cout << "ingresar solo tu nombre: ";
        cin.getline(newUser.name, 25, '\n');

        cout << "ingresar solo tu apellido: ";
        cin.getline(newUser.lastname, 25, '\n');

        cout << "ingresar tu email: ";
        cin.getline(newUser.email, 60, '\n');

        cout << "ingresar tu contrasenia: ";
        cin.getline(newUser.password, 60, '\n');

        cout << "----------------" << endl;
        cout << "nombre:" << newUser.name << endl;
        cout << "apellido:" << newUser.lastname << endl;
        cout << "email:" << newUser.email << endl;
        cout << "password:" << newUser.password << endl;
        cout << "----------------" << endl;

        cout << "los datos son correctos?" << endl;
        cout << "1: SI" << endl;
        cout << "2: NO" << endl;
        cout << "-----------------------------------------------------" << endl;
        cin >> option;
```

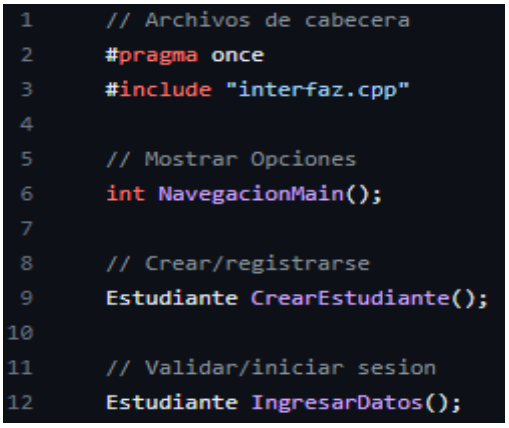
3) Función CrearEstudiante(): Esta función guía al usuario a través del proceso de registro. El usuario debe ingresar su nombre, apellido, correo electrónico y contraseña. Luego, se le muestra una confirmación de los datos ingresados y se le pregunta si son correctos. La función devuelve un objeto Estudiante con los datos ingresados.

```
        switch (option)
        {
        case 1:
            condition = false;
            break;
        case 2:
            cout << "RESET FORMULARIO" << endl;
            break;
        default:
            cout << "ESTA OPCION NO EXISTE \n SE RESETEARA LOS DATOS INGRESADOS" << endl;
            break;
        }
    } while (condition);

    return newUser;
};
```

Ambas funciones IngresarDatos() y CrearEstudiante() tienen un bucle do-while que permite al usuario corregir los datos si comete un error o no está seguro de que sean correctos. También manejan la opción de confirmar o restablecer los datos ingresados.

- **interfaz.hpp:** Este archivo de encabezado define las interfaces de las funciones que se utilizarán en el programa, incluye un archivo de código fuente y proporciona las declaraciones necesarias para que las funciones sean utilizadas en otros archivos de código fuente correspondiente (en este caso, "interfaz.cpp").



```
1   // Archivos de cabecera
2   #pragma once
3   #include "interfaz.cpp"
4
5   // Mostrar Opciones
6   int NavegacionMain();
7
8   // Crear/registrarse
9   Estudiante CrearEstudiante();
10
11  // Validar/iniciar sesion
12  Estudiante IngresarDatos();
```

En este archivo, se realizan las siguientes acciones:

1) #pragma once: Esta directiva se utiliza para prevenir la inclusión múltiple del archivo de encabezado en un programa. Evita que el mismo archivo de encabezado se incluya varias veces, lo que puede causar errores de redefinición. Es una práctica común en archivos de encabezado.
2) "#include "interfaz.cpp": Incluye el archivo de código fuente "interfaz.cpp" en este archivo de encabezado. Esto es inusual, ya que generalmente se incluyen archivos de encabezado (.h o .hpp) en lugar de archivos de código fuente (.cpp) en archivos de encabezado. Puede llevar a problemas de compilación si se abusa de esta práctica.
3) int NavegacionMain();: Declaración de una función llamada NavegacionMain(), que devuelve un entero (int). Esta función posiblemente se encargue de mostrar opciones al usuario y obtener la selección del usuario
4) Estudiante CrearEstudiante();: Declaración de una función llamada CrearEstudiante() que devuelve un objeto del tipo Estudiante. Esta función probablemente se utiliza para recopilar información del usuario y crear una instancia de la estructura Estudiante.
5) Estudiante IngresarDatos();: Declaración de una función llamada IngresarDatos() que devuelve un objeto del tipo Estudiante. Esta función probablemente se utiliza para recopilar información del usuario que está iniciando sesión y validarla.

- **lógica.cpp:** Este código proporciona funciones para verificar si un estudiante existe en un archivo, registrar nuevos estudiantes en el archivo y mostrar información de estudiantes que han iniciado sesión en el sistema. Las funciones trabajan con un archivo llamado "estudiantes.dat" para almacenar y recuperar información de estudiantes.Aquí está una descripción de las funciones y su funcionamiento:

```cpp
#include <iostream>
#include <conio.h>
#include <cstdio>
using namespace std;
#include "logica.hpp"

/*
Se define la funcion "ExisteEstudiante" y se manda como parametros:
    . "existencia" - dato bool mandado por referencia, para que se verifique la existencia del alumno en el archivo.
    . "datosEstudiante" - data Estudiante, contenedor de los datos a verificar.
*/
Estudiante existeEstudiante(bool &exist, Estudiante data)
{
    fflush(stdin);
    FILE *archivo = fopen("estudiantes.dat", "rb");
    if (archivo != NULL)
    {
        Estudiante validar;
        while (fread(&validar, sizeof(Estudiante), 1, archivo) == 1)
        {
            if (strcmp(validar.email, data.email) == 0)
            {
                exist = true;
                if (strcmp(validar.password, data.password) == 0)
                {
                    exist = true;
                    return validar;
                }
            }
        }
        fclose(archivo);
    }
    return data;
}
```

1) Bibliotecas y Declaraciones: Se incluyen varias bibliotecas, como <iostream>, <conio.h>, <cstdio>, y se utiliza using namespace std; para habilitar el uso del espacio de nombres estándar de C++. Luego, se incluye el archivo de encabezado "logica.hpp" que probablemente contiene las declaraciones de estructuras y funciones utilizadas en este archivo.

2) Función existeEstudiante(bool &exist, Estudiante data): Esta función verifica si un estudiante con las mismas credenciales (correo electrónico y contraseña) proporcionadas en el argumento data existe en un archivo llamado "estudiantes.dat". Toma un argumento booleano exist por referencia para indicar si el estudiante existe o no. Si se encuentra un estudiante con las mismas credenciales, establece exist en true y devuelve la información completa del estudiante encontrado. Si no se encuentra un estudiante con las mismas credenciales, se mantiene exist en false y se devuelve el objeto data que se pasó como argumento.

```
void RegistrarEstudiante(Estudiante dataStudent)
{
    FILE *archivo = fopen("estudiantes.dat", "rb");
    if (archivo != NULL)
    {
        Estudiante temp;
        fseek(archivo, sizeof(Estudiante), SEEK_END);
        fread(&temp, sizeof(Estudiante), 1, archivo);
        dataStudent.Id = temp.Id + 1;
    }
    else
    {
        dataStudent.Id = 1;
    }
    dataStudent.creditos = 1000;
    archivo = fopen("estudiantes.dat", "a+b");
    if (archivo != NULL)
    {
        fwrite(&dataStudent, sizeof(Estudiante), 1, archivo);
        fclose(archivo);
    }
    else
    {
        cout << "=========================" << endl;
        cout << "NO SE PUDO LEER EL ARCHIVO" << endl;
        cout << "=========================" << endl;
    }
}
```

3) Función RegistrarEstudiante(Estudiante dataStudent): Esta función se encarga de registrar un nuevo estudiante. Primero, abre el archivo "estudiantes.dat" para verificar si ya hay estudiantes registrados. Luego, asigna un nuevo ID al estudiante basado en el último ID en el archivo o lo establece en 1 si es el primer estudiante. A continuación, se establece el número de créditos en 1000 y se abre el archivo "estudiantes.dat" en modo de escritura binaria (a+b) para agregar el nuevo estudiante al archivo.

```
void IniciarSesionEstudiante(Estudiante student)
{
    cout << "========================================" << endl;
    cout << "========================================" << endl;
    cout << "ESTOS SON DATOS DE: " << student.name << endl;
    cout << "========================================" << endl;
    cout << "== ID: " << student.Id << endl;
    cout << "== Nombre: " << student.name << endl;
    cout << "== Apellido: " << student.lastname << endl;
    cout << "== Email: " << student.email << endl;
    cout << "== Contrasenia: " << student.password << endl;
    cout << "== Creditos: " << student.creditos << endl;
    cout << "========================================" << endl;
    cout << "========================================" << endl;
}
```

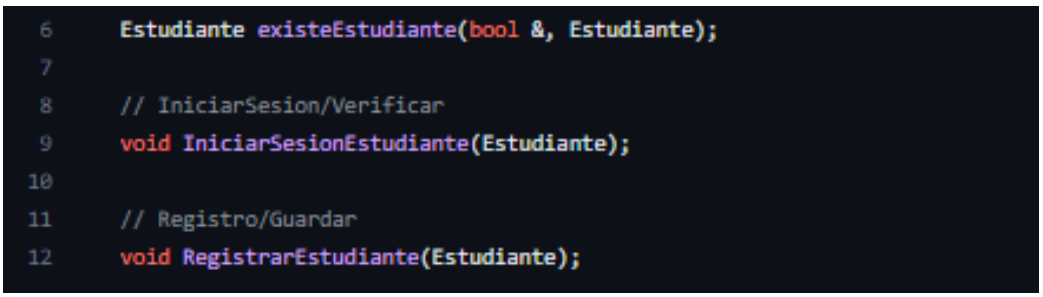
4) Función IniciarSesionEstudiante(Estudiante student): Esta función muestra los datos del estudiante que ha iniciado sesión en la consola. Imprime el ID, nombre, apellido, correo electrónico, contraseña y créditos del estudiante.

- **lógica.hpp**: Este archivo de encabezado proporciona declaraciones de funciones que se utilizan en el programa, pero no contiene las implementaciones de estas funciones. Las implementaciones de estas funciones deben encontrarse en el archivo de código fuente correspondiente (en este caso, "logica.cpp"). Nuevamente, la práctica de incluir archivos de código fuente en archivos de encabezado no es recomendable, ya que puede causar problemas de compilación y mantenimiento del código. Los archivos de encabezado deben utilizarse para declarar las funciones y estructuras que se utilizan en el código, mientras que las implementaciones deben estar en archivos de código fuente separados.

```
#pragma once
#include "logica.cpp"
```

En este archivo, se realizan las siguientes acciones:

1) #pragma once: Esta directiva se utiliza para prevenir la inclusión múltiple del archivo de encabezado en un programa, lo que evita problemas de redefinición. Es una práctica común en archivos de encabezado.
2) #include "logica.cpp": Incluye el archivo de código fuente "logica.cpp" en este archivo de encabezado. Al igual que en el código anterior, es inusual incluir archivos de código fuente (.cpp) en archivos de encabezado. Los archivos de encabezado (.h o .hpp) generalmente se utilizan para declarar funciones y estructuras, mientras que los archivos de código fuente (.cpp) se utilizan para implementar esas funciones. Esta práctica podría llevar a problemas de compilación y no se recomienda.



```
6     Estudiante existeEstudiante(bool &, Estudiante);
7
8     // IniciarSesion/Verificar
9     void IniciarSesionEstudiante(Estudiante);
10
11    // Registro/Guardar
12    void RegistrarEstudiante(Estudiante);
```

3) Declaraciones de funciones:
   - **Estudiante existeEstudiante(bool &, Estudiante)**;: Esta declaración de función describe una función llamada existeEstudiante. Esta función toma dos parámetros, un booleano por referencia (bool &) y un objeto del tipo Estudiante. La función probablemente se utiliza para verificar la existencia de un estudiante en algún contexto y modificar el booleano por referencia para indicar si el estudiante existe o no.
   - **void IniciarSesionEstudiante(Estudiante);**: Esta declaración de función describe una función llamada IniciarSesionEstudiante que toma un objeto del tipo Estudiante. La función parece estar destinada a iniciar sesión con un estudiante y posiblemente mostrar información relacionada con la sesión.
   - **void RegistrarEstudiante(Estudiante);**: Esta declaración de función describe una función llamada RegistrarEstudiante que toma un objeto del tipo Estudiante. La función probablemente se utiliza para registrar un nuevo estudiante en algún contexto, como guardar los datos del estudiante en algún lugar.

- **principal.cpp:** Este código representa un programa interactivo que muestra un menú de opciones en la consola y permite al usuario realizar acciones como iniciar sesión, registrarse, salir del programa o manejar opciones no válidas. El programa continuará ejecutándose hasta que el usuario elija salir.

```
// llamando al archivo cabecera
#include "interfaz.hpp"
#include "controlador.hpp"
#include <iostream>
using namespace std;

int main()
{
    bool condition = true;
    do
    {
        system("cls");
        cout << "SELECIONAR UNA OPCION" << endl;
        switch (NavegacionMain())
        {
        case 1:
            INICIARSESION();
            break;
        case 2:
            REGISTRARSE();
            break;
        case 3:
            EXIT(condition);
            break;
        default:
            DEFAULT(condition);
            break;
        };

    } while (condition);

    return 0;
};
```

Aquí está la descripción de lo que hace este código:

1) Incluye los archivos de cabecera:
   - ➢ "interfaz.hpp": Incluye un archivo de cabecera llamado "interfaz.hpp".
   - ➢ "controlador.hpp": Incluye otro archivo de cabecera llamado "controlador.hpp".
   - ➢ <iostream>: Incluye la biblioteca estándar de C++ para entrada/salida.

2) Declara la función principal main(). Esta es la función que se ejecuta cuando se inicia el programa.

3) Se declara una variable booleana llamada condition e inicializa en true. Esta variable se utilizará para controlar el bucle do-while que mantiene el programa en funcionamiento hasta que condition sea false.

4) Entra en un bucle do-while que continuará ejecutándose mientras condition sea true.
5) Dentro del bucle, se borra la pantalla de la consola usando system("cls") para proporcionar una especie de limpieza visual.

6) Se muestra un mensaje "SELECIONAR UNA OPCIÓN" en la consola.

7) Utiliza una sentencia switch para gestionar las opciones ingresadas por el usuario. La función NavegacionMain() se llama para obtener la opción seleccionada por el usuario, y luego se ejecuta una de las siguientes acciones según la opción:

- ➢ case 1: Llama a la función INICIARSESION() para iniciar sesión.
- ➢ case 2:Llama a la función REGISTRARSE() para registrarse como estudiante.
- ➢ case 3: Llama a la función EXIT(condition) para salir del programa. La variable condition se pasa por referencia y se establecerá en false para finalizar el bucle do-while.
- ➢ default: Llama a la función DEFAULT(condition) para manejar cualquier otra opción que no esté en los casos anteriores.

8) El bucle do-while continúa ejecutándose mientras condition sea true, lo que permite al usuario seleccionar diferentes opciones o salir del programa.

9)Cuando condition se establece en false, el bucle se detiene y el programa termina con return 0; en la función main().

Gestor de Beneficios

Encargado de el sistema: Tumiri Jhon Jairo, Kindrat Luciano, Carla Gomez

Descripción del sistema:

- **controlador.cpp:** Este sería el código fuente, en el que vemos todas las funciones que puede realizar el sistema, eso son:

- Listar beneficios: Se define una función con el nombre de void LISTAR_BENEFICIOS() en la que definimos la limpieza de la pantalla con system("cls") para windows y system("clear") para linux, llamamos a la función mostrarBeneficios() que esta definida en el código interfaz.cpp y por último pausamos el código system("pause").

```
void LISTAR_BENEFICIOS()
{
    system("cls");
    mostrarBeneficios();
    system("pause");
};
```

- Agregar más beneficios: En esta función, se declaró newBenefit se usó el tipo de dato beneficio que es un struct que está definido en el código interfaz.cpp, se definió el la variable bool exist = false. En newBenefit se llama a la función IngresarDatosBeneficios() que esta definida en el código interfaz.cpp y en la línea de abajo se llama a la función existeBeneficio(exist, newBenefit) definida en el código logica.cpp.
  En el if si el beneficio que se desea agregar ya existe se le comunica al usuario "NO SE PUEDE AGREGAR UN BENEFICIO CON UN MISMO NOMBRE", en caso contrario en el else se llama a la función agregarBeneficio(newBenefit) también definida en logica.cpp y se comunica "SE AGREGO CORRECTAMENTE".
  Para finalizar con esta función se pausa el código.
  Al ingresar un beneficio automáticamente está activo.

```
void AGREGAR_BENEFICIOS()
{
    system("cls");
    Beneficio newBenefit;
    bool exist = false;
    newBenefit = IngresarDatosBeneficios();
    existeBeneficio(exist, newBenefit);
    if (exist)
    {
        cout << "NO SE PUEDE AGREGAR UN BEENEFICIO CON UN MISMO NOMBRE" <<
    }
    else
    {
        agregarBeneficio(newBenefit);
        cout << "Se AGREGO CORRECTAMENTE" << endl;
    }
    system("pause");
};
```

- **Modificar beneficio**: En esta parte de el código definimos la función void

- **MODIFICAR_BENEFICIOS()**, en la que desarrollamos una variable: nombre_buscado de tipo string, después se le pide al usuario que ingrese el nombre del beneficio que desea buscar, lo que ingrese se guarda en nombre_buscado , y es el valor que ingresa a la función modificarBeneficio(nombre_buscar), como último se pausa el código.

```
void MODIFICAR_BENEFICIOS()
{
    system("cls");
    string nombre_buscar;
    cin.ignore(10000, '\n');
    cout << "INGRESAR EL NOMBRE DEL BENEFICIOS QUE DESEAS BUSCAR" << endl;
    getline(cin, nombre_buscar);
    modificarBeneficio(nombre_buscar);
    system("pause");
};
```

- Eliminar beneficio: En esta función le ingresa int countBenefit que es la cantidad de beneficios. dentro de la función, se definió la variable exist como bool exist = false, después se llamó a la función eliminarBeneficio(countBenefit).

```
void ELIMINAR_BENEFICIOS(int countBenefit)
{
    system("cls");
    bool exist = false;
    eliminarBeneficio(countBenefit);
    system("pause");
};
```

- Exit: La función void EXIT(bool &condition)es utilizada para finalizar con el sistema.
- Default: Esta función es para cuando el usuario ingresa una opción que no existe en el código.

```
void DEFAULT(bool &condition)
{
    cout << "NO EXISTE ESTA OPCION" << endl;
    system("pause");
    condition = false;
};
```

- **interfaz.cpp:** Es la parte del código trabajada como la superficie de contacto con el usuario y dentro de esta se define las funciones declaradas en el archivo cabecera **interfaz.hpp**.

- Struct Beneficio: Esta estructura "Beneficio" se utiliza para organizar información sobre un beneficio, incluyendo su identificador con el tipo de dato **int**, nombre con el tipo de dato **char**, estado de activación con el tipo de dato **bool true(**si sigue activo**) o false(**si está desactivado**)** y costo con el tipo de dato **int**.

```cpp
struct Beneficio
{
    int id;
    char nombre[60];
    bool activo;
    int costo;
};
```

- Navegación: Es la función que muestra la navegación del inicio, cual sus case están definidas en el **main** con un switch.

```cpp
int Navegacion()
{
    int number;
    cout << "==================================" << endl;
    cout << "=========== BENEFICIOS ==========" << endl;
    cout << "==================================" << endl;
    cout << "( 1 ) LISTAR BENEFICIOS" << endl;
    cout << "( 2 ) AGREGAR BENEFICIOS" << endl;
    cout << "( 3 ) ELIMINAR BENEFICIOS" << endl;
    cout << "( 4 ) MODIFICAR BENEFICIOS" << endl;
    cout << "( 5 ) SALIR" << endl;
    cout << "==================================" << endl;
    cin >> number;
    return number;
};
```

```cpp
switch (Navegacion())
{
case 1:
    LISTAR_BENEFICIOS();
    break;
case 2:
    AGREGAR_BENEFICIOS();
    break;
case 3:
    ELIMINAR_BENEFICIOS(countBenefit);
    break;
case 4:
    MODIFICAR_BENEFICIOS();
    break;
case 5:
    EXIT(condition);
    break;
default:
    DEFAULT(condition);
    break;
};
```

La segunda foto es el código que esta en el **main**.

- Ingresar datos de beneficios: como su nombre lo dice es el ingreso de el nombre y el costo del beneficio, estos datos los ingresa el usuario, esta función después es usada en el controlador.cpp en “agregar más beneficios”.
  Luego se preguntará al usuario si los datos son correctos, si la opción es si (1) la variable condition va a ser igual a false y sale de el do-while. Si la opción elegida es no(2) o otra opción que no existe en el código, se resetearon los datos y vuelve a correr el do-while hasta que condition = false.

```cpp
    cout << "==========================================" << endl;
    cout << "====== INGRESAR EL NOMBRE: ";
    cin.getline(benefit.nombre, 60, '\n');
    cout << "====== INGRESAR EL COSTO: ";
    cin >> benefit.costo;
    // AL INGRESAR UN BENEFICIO AUTOMATICAMENTE ESTA ACTIVO
    benefit.activo = true;
    cout << "==========================================" << endl;
    cout << "LOS DATOS SON CORRECTOS" << endl;
    cout << "( 1 ) SI " << endl;
    cout << "( 2 ) NO " << endl;
    cin >> number;
    switch (number)
    {
    case 1:
        condition = false;
        break;
    case 2:
        cout << "RESET DE LOS DATOS" << endl;
        break;
    default:
        cout << "ESTA OPCION NO EXISTE, SE HARA UN RESET DE LOS DATOS"
        break;
    }

} while (condition);
```

- MostrarOpcionesModificarBeneficio(): En esta función se **elige con las opciones** qué dato de el beneficio se desea modificar sea el nombre, costo, estado del beneficio (activado o desactivado) o salir de la función.

  Esta función es usada en Modificar beneficio del controlador.cpp.

```
int MostrarOpcionesModificarBeneficio()
{
    int number;
    cout << "==================================" << endl;
    cout << "============ MODIFICAR ===========" << endl;
    cout << "==================================" << endl;
    cout << "( 1 ) NOMBRE" << endl;
    cout << "( 2 ) COSTO" << endl;
    cout << "( 3 ) ESTADO" << endl;
    cout << "( 4 ) SALIR" << endl;
    cout << "==================================" << endl;
    cin >> number;
    return number;
};
```

- Beneficio MostrarOpcionesModificacion(int option, Beneficio dataBenefit): En esta otra función, ya sabiendo que dato se eligió en la función anterior cambiar se modifica el dato elegido a cambiar y el usuario ingresa el nuevo dato del beneficio.

```
switch (option)
{
case 1:
    cout << "AGREGAR EL NUEVO NOMBRE AL BENEFICIO" << endl;
    char updateName[60];
    cin.ignore(10000, '\n');
    cin.getline(updateName, 60, '\n');
    strcpy(updateBenefit.nombre, updateName);
    break;
case 2:
    cout << "AGREGAR EL NUEVO COSTO AL BENEFICIO" << endl;
    int costo;
    cin >> costo;
    updateBenefit.costo = costo;
    break;
case 3:
    cout << "AGREGAR NUEVO ESTADO" << endl;
    cout << "( 0 ) desactivado | ( 1 ) activo" << endl;
    int state;
    cin >> state;
    switch (state)
    {
    case 0:
        cout << "DESACTIVADO" << endl;
        updateBenefit.activo = false;
        break;
    case 1:
        cout << "ACTIVO" << endl;
        updateBenefit.activo = true;
        break;
    default:
        cout << "NO EXISTE ESTA OPCION" << endl;
        cout << "por temas de seguridad se guardara ACTIVO" << endl;
        break;
```

- **logica.cpp:** Se encarga de las operaciones internas en el archivo se definió lo que se declaró en el archivo cabecera **lógica.hp**p.
  - Beneficio: Esta función busca un puntero específico en el archivo beneficios.dat y determina si este está en el archivo.

```
Beneficio existeBeneficio(bool &exist, Beneficio dataBeneficio)
{
    Beneficio itemBeneficio;
    FILE *archivo = fopen("beneficios.dat", "rb");
    if (archivo != NULL)
    {
        while (fread(&itemBeneficio, sizeof(Beneficio), 1, archivo) == 1)
        {
            if (strcmp(itemBeneficio.nombre, dataBeneficio.nombre) == 0)
            {
                exist = true;
            }
        };
        fclose(archivo);
    };

    return dataBeneficio;
};
```

Primero, se declara la variable exist a través de referencia dentro la función Beneficio, para poder luego modificarlo fuera de esta. Se declara la variable **itemBeneficio** que se va a usar para leer beneficios del archivo.
Se abre el archivo **beneficio.dat** en modo lectura binaria y se almacena en el puntero archivo, y luego se comprueba si el archivo se abrió correctamente. En caso de que se haya abierto correctamente, se usa el bucle while que va a leer los beneficios uno por uno utilizando fread, hasta que se encuentre exitosamente el beneficio buscado comparado por el strcmp(), en cuyo caso se cierra el archivo y se devuelve el beneficio encontrado.

- Agregar beneficio: Esta sección se encarga de añadir los beneficios al archivo beneficios.dat..

```
void agregarBeneficio(Beneficio dataBenefit)
{
    FILE *archivo = fopen("beneficios.dat", "rb");
    if (archivo != NULL)
    {
        Beneficio temporal;
        fseek(archivo, sizeof(Beneficio), SEEK_END);
        fread(&temporal, sizeof(Beneficio), 1, archivo);
        dataBenefit.id = temporal.id + 1;
    }
    else
    {
        dataBenefit.id = 1;
    }
    archivo = fopen("beneficios.dat", "a+b");
    if (archivo != NULL)
    {
        MostrarBeneficios(dataBenefit);
        fwrite(&dataBenefit, sizeof(Beneficio), 1, archivo);
        fclose(archivo);
    }
    else
    {
        cout << "NO SE PUDO AGREGAR EL BENEFICIO" << endl;
    };
};
```

Aquí, primero se abre el archivo beneficios.dat en modo lectura binaria y se almacena en el puntero “archivo”, para luego verificar si este se abrió correctamente, y si el archivo existe. Luego se crea la variable “temporal” para leer el último beneficio del archivo y obtener su id. Se mueve el puntero al final del archivo para poder leer el último beneficio, y una vez leído, se almacena en la variable temporal en el “fread”. Por último se le asigna el nuevo id al siguiente beneficio.
Si el archivo no existe o no se puede abrir, se le otorga un id a beneficio de 1.

Luego, se vuelve a abrir beneficios.dat, pero en modo apertura y escritura binaria, para poder agregar el nuevo beneficio al final del archivo, o crearlo en caso de que este no exista. Se llama a la función MostrarBeneficios() definida en interfaz.hpp para mostrar el beneficio a agregar y finalmente se escribe el dataBenefit en el archivo en fwrite.

- Eliminar beneficio: Esta función elimina un beneficio específico del archivo beneficios.dat.

```
void eliminarBeneficio(int cantidadBeneficios)
{
    string nombre_buscado;
    cout << "INGRESAR EL NOMBRE DEL BENEFICIO A ELIMINAR" << endl;
    cin.ignore(10000, '\n');
    getline(cin, nombre_buscado);
    Beneficio aux[cantidadBeneficios];
    int count = 0;
    FILE *archivo = fopen("beneficios.dat", "rb");
    if (archivo != NULL)
    {
        Beneficio temp;
        while (fread(&temp, sizeof(Beneficio), 1, archivo) == 1)
        {
            if (temp.nombre != nombre_buscado)
            {
                aux[count] = temp;
                count++;
            }
        }
        fclose(archivo);
    }
    else
    {
        cout << "NO SE PUDO ABRIR EL ARCHIVO" << endl;
    }
    archivo = fopen("beneficios.dat", "wb");
    archivo = fopen("beneficios.dat", "a+b");
    if (archivo != NULL)
    {
        for (int i = 0; i < count; i++)
        {
            fwrite(&aux[i], sizeof(Beneficio), 1, archivo);
        }
        fclose(archivo);
    }
}
```

Primero se declara una variable string nombre_buscado, para guardar el nombre del beneficio que queremos eliminar. También se crea un array aux[cantidadBeneficios]
que se va a usar para almacenar temporalmente los beneficios que no se desean eliminar, y una variable count. Luego se verifica si el archivo se abrió correctamente (en caso que no la función devolverá por un cout “NO SE PUDO ABRIR EL ARCHIVO”) y se declara una variable temp. Se usa el bucle while para leer los beneficios del archivo uno por uno y se compara el nombre del beneficio buscado o nombre_buscado con el beneficio actual o temp.nombre, y mientras que no sean iguales, este se guardara en el array aux[count], mientras que a la vez, incrementa el count. Una vez están todos los beneficios que no se desean eliminar copiados en aux, se cierra el archivo, y se vuelve a abrir en modo escritura binaria y luego en modo apertura y escritura binaria, para poder reescribir mediante fwrite todos los beneficios almacenados en aux devuelta al archivo.

- Modificar beneficio: Esta función busca beneficios por su nombre en el archivo beneficios.dat y los modifica y sobreescribe dependiendo de las entradas del usuario.

```
void modificarBeneficio(string nombre_buscado)
{
    FILE *archivo = fopen("beneficios.dat", "r+b");
    if (archivo != NULL)
    {
        Beneficio dataBenefit;
        bool encontrado = false;
        int count = 0;

        while (!encontrado && fread(&dataBenefit, sizeof(Beneficio), 1, archivo) == 1)
        {
            if (dataBenefit.nombre == nombre_buscado)
            {
                encontrado = true;
                int option;
                MostrarBeneficios(dataBenefit);
                option = MostrarOpcionesModificarBeneficio();
                dataBenefit = MostrarOpcionesModificacion(option, dataBenefit);

                fseek(archivo, sizeof(Beneficio) * count, SEEK_SET);
                fwrite(&dataBenefit, sizeof(Beneficio), 1, archivo);
                cout << "SE ACTUALIZO" << endl;
                count = 0;
            }
            count++;
        }
        if (!encontrado)
        {
            system("cls");
            cout << "NO SE ENCONTRO EL NOMBRE DEL BENEFICIO" << endl;
        }
        fclose(archivo);
    }
    else
    {
        cout << "No se pudo abrir el archivo para lectura y escritura." << endl;
    }
};
```

Primero se abre el archivo beneficios.dat en modo lectura y escritura binaria y se almacena en el puntero archivo, y se verifica si el archivo se abrió correctamente. Luego se declaran las variables dataBenefit, encontrado que inicializa en false, y count inicializada en 0 que se van a usar para saber si se encontró el beneficio buscado, y llevar cuenta de los registros ya leídos.
Se utiliza el bucle while que se continuará ejecutando hasta que se encuentre el beneficio buscado o se recorran todos.En caso de que el fread encuentre el beneficio buscado, se llamará a las funciones Mostrar Beneficios(), MostrarOpcionesModificarBeneficios() y MostrarOpcionesModificacion() para que se realicen las modificaciones deseadas por el usuario. Se finaliza sobrescribiendo estos beneficios, y notificando al usuario mediante un cout que se ha actualizado.

- Mostrar beneficios: Esta función se encarga de leer los registros de beneficios uno por uno y mostrarlos en la pantalla, y muestra una descripción visual del estado de desactivación/activación de los beneficios.

```
void mostrarBeneficios()
{
    Beneficio dataBenefit;
    FILE *archivo = fopen("beneficios.dat", "rb");
    if (archivo != NULL)
    {
        cout << "================================" << endl;
        cout << "( 0 ) desactivado | ( 1 ) activo" << endl;
        cout << "================================" << endl;
        while (fread(&dataBenefit, sizeof(dataBenefit), 1, archivo) == 1)
        {
            MostrarBeneficios(dataBenefit);
        };
    }
}
```

Primero declara una variable dataBenefit, que se usa para leer los registros de los beneficios durante el proceso de lectura. Luego se abre el archivo beneficios.dat en modo lectura binaria y se almacena en un puntero archivo. Después se imprime por pantalla una interfaz para informar al usuario del significado de los 0 y 1 como desactivados o activados respectivamente, y se utiliza un bucle while que se llamará a sí misma para mostrar los detalles del beneficio actual en pantalla.

- Contar Beneficios: Esta función se encarga de abrir el archivo beneficios.dat y contar la cantidad de beneficios en este para luego mostrarlos en pantalla.

```
void countBeneficios(int &countBenefit)
{
    Beneficio dataBenefit;
    countBenefit = 0;
    FILE *archivo = fopen("beneficios.dat", "rb");
    if (archivo != NULL)
    {
        while (fread(&dataBenefit, sizeof(dataBenefit), 1, archivo) == 1)
        {
            countBenefit++;
        };
    }
    cout << "=========" << endl;
    cout << "Cantidad de beneficios: " << countBenefit << endl;
    cout << "=========" << endl;
}
```

Primero se declaran las variables dataBenefit, que se va a usar para leer los registros de beneficios mientras se cuentan, y countBenefit que se inicializa en 0, y se va a usar para guardar el número de beneficios contados.
Luego se abre el archivo beneficios.dat en modo lectura binaria y se almacena en el puntero archivo. Luego se verifica si el archivo se abrió correctamente y se inicializa un bucle while, que se va a ejecutar mientras que fread pueda leer correctamente el registro sizeof(dataBenefit), y dentro de este, se

incrementara countBenefit en 1. Al terminar de contar los beneficios, se imprimirá por pantalla una interfaz que mostrará la cantidad de estos.

- Patrones algorítmicos:

Consumidor Beneficios

Encargado del sistema: Ian Gabriel Sanna, Mayerly Romina Achata Fernandez

Descripción del sistema:

- **beneficio.cpp:**

Parte encargada del manejo de beneficios del código.

```
void consultarBeneficios(int creditos){ //Obtiene los beneficios que puede comprar el estudiante y los devuelve en un array

	FILE* archivo = fopen("beneficios.dat", "rb");
	Beneficio beneficio;

	cout << "\nBeneficios(Id, Nombre, Costo): \n" << endl;

	if (archivo != NULL){
		while (fread(&beneficio, sizeof(Beneficio), 1, archivo) == 1){//Busca cuantos elemento puede comprar el estudiante o los muestra todos
			if (beneficio.costo < creditos or creditos == -1){
				cout << beneficio.id << "\t" << beneficio.nombre << "\t" << beneficio.costo << endl;
			}
		}
	}

	fclose(archivo);
}
```

- Consultar beneficios:

  Tiene el propósito de obtener y mostrar los beneficios que un estudiante puede comprar, basándose en la cantidad de créditos disponibles que el estudiante tiene.

  La función es de tipo void, lo que significa que no retorna ningún valor. Recibe el argumento creditos, que es la cantidad de créditos disponibles que tiene el estudiante. Luego se abre el archivo beneficios.dat. Este archivo contiene información sobre los beneficios que pueden ser adquiridos por los estudiantes. Se declara la variable beneficio. Se verifica si el archivo se abrió correctamente con un if.

  Empezamos a leer el archivo
  El bucle while lee cada elemento del archivo archivo, utilizamos la estructura if para verificar si el costo del beneficio beneficio.costo es menor que la cantidad de créditos disponibles creditos. Además, se verifica si creditos es igual a -1 como indicador para expresar que muestre todos los beneficios existentes

Si se cumple la primera condicion, significa que el estudiante puede adquirir este beneficio y se lleva a cabo la impresión de los mismos. La segunda condición es por si se desea ver todos los beneficios sin tener en cuenta los creditos

Una vez completado el bucle, se cierra el archivo.

Se utiliza la estructuras de control: el bucle while, la estructura if. Se realiza un patrón recorrido de corte de control.

- Coste de beneficio:

Verifica que exista un beneficio y este tenga costo menor a la cantidad de créditos actuales.

```
int costeBeneficio(int id, int creditos)
{
        FILE* archivo = fopen("beneficios.dat", "a+b");
        fseek(archivo, 0, SEEK_SET);

        if(archivo != NULL)
        {
                Beneficio beneficio;
                while (fread(&beneficio, sizeof(Beneficio), 1, archivo) == 1) // Verifica que el beneficios exista
                {
                        if(beneficio.id == id and beneficio.costo < creditos)
                        {
                                return beneficio.costo;
                        }
                        if(beneficio.id > id)
                        {
                                break;
                        }
                }
        }

        return -1;
```

Luego de abrir el archivo, agregamos la función seek para hacer una búsqueda.

Dentro del while donde tenemos la lectura del archivo

Utilizamos la estructura if y nos preguntamos si beneficio.id es igual a el id que el usuario escribió, al igual que vemos si el costo del beneficio es menor de lo que tiene en créditos. Retornamos beneficio.costo y luego realizamos otro if para que cuando el id sea mayor al que se está buscando sea interrumpido.

No se encuentra ningún patrón de ordenamiento debido a que el id es automático, pero sí podemos mencionar el patrón de recorrido con corte de control ya que surge un corte cuando deja de buscar al no cumplir su condición
Retorna return -1; si no se encontró un beneficio válido.

Dentro del bucle while que itera sobre los registros de beneficios en el archivo. Encontramos if(beneficio.id == id and beneficio.costo < creditos) y if(beneficio.id > id) son condiciones de corte que determinan cuándo detener el recorrido. En síntesis, se implementa un patrón de búsqueda secuencial y de recorrido de con corte de control

- **beneficio.h:**

```
struct Beneficio
{
    int id;
    char nombre[60];
    bool activo;
    int costo;
};

void consultarBeneficios(int creditos);

int costeBeneficio(int id, int creditos);
```

Tenemos el struct Beneficio contiene un identificador (id) con el tipo de dato integer, nombre (nombre) con el tipo de dato char, variable booleana(activo), y también declaramos int costo.

El código define las funciones consultarBeneficios y costeBeneficio

- **estudiante.cpp:**

  Valida que el mail y contraseña existan en estudiante.dat

- Validar Estudiante:

```
#include <iostream>
#include <cstring>
#include "estudiante.h"

using namespace std;


Estudiante validarEstudiante(const char* mail, const char* contraseña){
    //Valida que el mail y la contraseña ingresadas existan en el archivo estudiantes.dat"
    FILE* archivo = fopen("estudiantes.dat", "rb");
    Estudiante estudiante;
    bool encontrado = false;
    if (archivo != NULL){
        //busqueda secuencial
            while (fread(&estudiante, sizeof(Estudiante), 1, archivo) == 1){
            if (strcmp(estudiante.email, mail) == 0 and strcmp(estudiante.password, contraseña) == 0){
                encontrado = true;
                break;
            }
        }
        fclose(archivo);
    }
    if (!encontrado){
        estudiante.creditos = -1;//Validado con exito
    }
    return estudiante;
}
```

  Tenemos Estudiante validarEstudiante(const char* mail, const char* contraseña) donde sus parámetros son punteros que representan los atributos de un estudiante introducidos por el usuario

  Se inicia con la apertura del archivo “estudiante.dat” en el modo de apertura “rb” (lectura binaria)

Se declara una variable de tipo Estudiante llamada estudiante, previamente declarada en el main dentro de struct Estudiante

Se desea encontrar esta variable para eso se declara una variable booleana encontrado e inicializada como false. Esta variable se usará para indicar si se encuentra o no el estudiante con el mail y contraseña proporcionados. Validando mail y contraseña del estudiante.

Se verifica si el archivo se abrió correctamente (archivo != NULL).

Se visualiza un bucle while, que intenta leer datos del archivo usando la función fread

Volvemos a utilizar la estructura de control if para comparar el email y password del estudiante leído con los proporcionados como argumentos de la función, utilizamos strcmp

Una vez que se encontró un estudiante que cumple con las condiciones, break; detiene el bucle. Cerramos el archivo luego de terminar el bucle, usamos fclose(archivo)

Si se encuentra un estudiante con el mail y contraseña proporcionados, encontrado será true, en caso contrario, se asigna -1 a estudiante.creditos e indica que la validación fallo

Finalmente, se retorna la estructura estudiante, que contendrá la información del estudiante encontrado

El bucle se detiene cuando se encuentra un estudiante que coincide con los datos proporcionados, lo cual es una condición específica que interrumpe el recorrido. Por tal motivo, mencionare al patrón de recorrido de corte de control. Además, utilizamos estructuras de control. Trabajamos con un archivo binario, es por eso que la búsqueda lineal es la opción más viable

- Modificar Estudiante:

```
void modificarEstudiante(Estudiante estudiante)
{
    FILE* archivo = fopen("estudiantes.dat", "r+b");

    if(archivo != NULL)
    {
        fseek(archivo, (estudiante.id - 1) * sizeof(Estudiante), SEEK_SET); // Desplaza el puntero hasta la posicion del estudiante a modificar

        fwrite(&estudiante, sizeof(Estudiante), 1, archivo); // Modifica al estudiante objetivo
    }
    else
    {
        cout << "Error en la creacion / lectura del archivo" << endl;
    }

    fclose(archivo);
}
```

Esta función permite modificar un registro de estudiante en el archivo binario "estudiantes.dat". Recibe estudiante

Para su busqueda utilizamos fseek(archivo, (estudiante.id - 1) * sizeof(Estudiante), SEEK_SET); se encarga de mover el puntero de lectura/escritura en el archivo a la posición correspondiente al estudiante que queremos modificar, basado en su id.

Luego sobrescribe el registro del estudiante con los nuevos datos

Si no, manda un mensaje de que ha habido un error en la creación o lectura del archivo.
Finalmente cierra el archivo
Utiliza las funciones de estructuras de almacenamiento físico(fopen, fseek, fwrite, fclose), patrón de búsqueda lineal y una estructuras de control (if)

- **estudiante.h:**

  Definición de struct Estudiante

```
struct Estudiante
{
    char name[25];
    char lastname[25];
    char email[60];
    char password[60];
    int creditos;
    int id;
};

Estudiante validarEstudiante(const char* mail, const char* contraseña);

void modificarEstudiante(Estudiante estudiante);
```

  El fragmento de código define una estructura llamada "Estudiante"

Tenemos el struct Estudiante contiene: la variable password, email, name, lastname tiene el tipo de dato char, y el id y creditos son de tipo integer

  El código declara dos funciones: validarEstudiante y modificarEstudiante.

- **main.cpp**

```
#include <iostream>

#include <cstring>

#include "estudiante.h"

#include "beneficio.h"

using namespace std;
```

```cpp
int main()
{
    Estudiante estudiante;

    char mail[60];
    char contraseña[60];
    int id;
    int coste;

    cout << "\t\t\t\tBENEFICIOS UTN" << endl;
    cout << "\t\t\t\t--------------" << endl;

    bool login = false;//Esta variable se usará para controlar si el estudiante ha iniciado sesión correctamente.

    while(!login){ //El estudiante no ha iniciado sesión correctamente.
        cout << "\n\tMail: ";
        cin.getline(mail, sizeof(mail));

        cout << "\n\tContraseña: ";
        cin.getline(contraseña, sizeof(contraseña));

        estudiante = validarEstudiante(mail, contraseña);

        if (estudiante.creditos == -1){
            cout << "\n\tEl mail o la contraseña son incorrectos. Por favor, ingreselos de nuevo" << endl;
        }
        else{
            login = true;
            cout << "\n\tBienvenido, "; cout << estudiante.name; cout << " "; cout << estudiante.lastname << endl;
        }
    }

    cout << "\nA continuacion se le mostrara los beneficios a los que puede acceder: " << endl;

    consultarBeneficios(estudiante.creditos);

    cout << "\nSeleccione el Id del beneficio al que desea acceder: ";
    cin >> id;

    coste = costeBeneficio(id, estudiante.creditos);

    if(coste != -1)
    {
        estudiante.creditos -= coste;
        modificarEstudiante(estudiante);
        cout << "\nEl beneficio ha sido adquirido con exito" << endl;
    }
    else
    {
        cout << "\nEl beneficio al que quiso acceder no se encuentra disponible o no existe" << endl;
    }

    return 0;
}
```

Se crea una variable llamada estudiante de tipo Estudiante. Esta variable se utilizará para almacenar la información del estudiante que inicia sesión.

Se codea bool login = false; para controlar el bucle de inicio de sesión.
Nos encontramos con un bucle que continúa hasta que el usuario inicie sesión correctamente.

Se pide al usuario que ingrese su mail y contraseña.
Se llama a la función validarEstudiante(mail, contraseña) para verificar si los datos son correctos. Si son incorrectos, se muestra un mensaje de error y se vuelve a pedir al usuario.
Si los datos son correctos, se muestra un mensaje de bienvenida y se actualiza login a true para salir del bucle.

Se muestra un mensaje indicando que se mostrarán los beneficios disponibles.
Se llama a la función consultarBeneficios para mostrar los beneficios que el estudiante puede acceder basados en sus créditos.

Se pide al usuario que ingrese el id del beneficio que desea adquirir. Se lee el mismo
Se llama a la función costeBeneficio para obtener el costo del beneficio seleccionado.
Se verifica si el beneficio seleccionado está disponible y si el estudiante tiene suficientes créditos para adquirirlo.
Si es posible adquirir el beneficio, se actualizan los créditos del estudiante y se llama a la función modificarEstudiante para actualizar la información del estudiante en el archivo.
Se muestra un mensaje de confirmación.
Si el beneficio no está disponible o el id es inválido, se muestra un mensaje de error.
El programa finaliza y devuelve 0 para indicar que se ejecutó sin errores.

Este programa utiliza estructuras de control como bucles y condicionales. Podemos mencionar el uso del patrón de recorrido secuencial