Universidad Mariano Gálvez de Guatemala

Proyecto final

ALGORTIMOS

Facultad de Ingeniería en Sistemas de Información

UNIVERSIDAD MARIANO GÁLVEZ DE GUATEMALA

FACULTAD DE INGENIERÍA EN SISTEMAS DE INFORMACIÓN

PROYECTO FINAL

SISTEMA DE GESTION DE ESTUDIANTES Y NOTAS

Autor(es):

Marco Antonio Barrera Gonzales	**1590-21-18772**
Franklin Omar Salazar Barrera	**1590-25-23258**
Diego Efrén López Ramírez	**1590-25-24391**
Ángel Emilio Samayoa Donis	**1590-25-14338**
Kevin Dennis Herrera Herrarte	**1590-25-22632**
José Manuel Arias Arias	**1590-24-18430**

Catedrático
Luis Xiloj

Curso
Algoritmos

Guatemala, 02/11/2025

ÍNDICE

1. INTRODUCCIÓN

El código fuente analizado implementa un Sistema de Gestión Académica integral, diseñado para operar completamente a través de una interfaz de consola basada en texto. Su propósito central es facilitar la administración de estudiantes, cursos y calificaciones para una carrera o sección específica. El sistema se destaca por su persistencia de datos a través de un formato de archivo .csv personalizado, garantizando que toda la información se conserve entre sesiones.

Las funcionalidades clave incluyen la gestión completa (crear, editar, eliminar) tanto de estudiantes como de cursos. Una característica notable es la consistencia de datos forzada por el sistema: al crear un nuevo curso, este se añade automáticamente a todos los estudiantes existentes, y viceversa. Los módulos de Calificaciones y Promedios, que constituyen el núcleo operativo, solo se habilitan una vez que existen datos mínimos (al menos un estudiante y un curso), previniendo errores de ejecución. El sistema permite la edición detallada de las notas (zona, parciales, examen) y culmina en un robusto módulo de reportes que genera tablas de promedios tanto por estudiante como por materia, con la capacidad de exportar estos informes a archivos CSV individuales. Arquitectónicamente, el sistema sigue un paradigma de programación procedural, utilizando variables globales para la gestión del estado y empleando características modernas de C++ como lambdas para construir una interfaz de usuario interactiva y con capacidad de respuesta en tiempo real.

2. SISTEMA DE GESTION DE ESTUDIANTES Y NOTAS

3. Arquitectura y Estructura de Datos

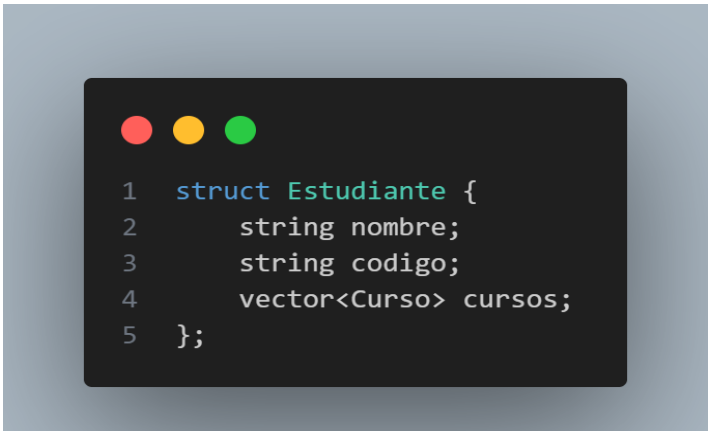
El diseño del sistema se fundamenta en un modelo de datos simple pero efectivo, gestionado a través de variables globales y un paradigma de programación procedural.

3.1. Modelo de Datos Central

Representa una materia y sus componentes de calificación. Contiene el nombre del curso (string) y las notas desglosadas en variables de tipo flotante: zona (30 Puntos), parcial1 (15 puntos), parcial2 (15 puntos) y examen final (40 puntos)

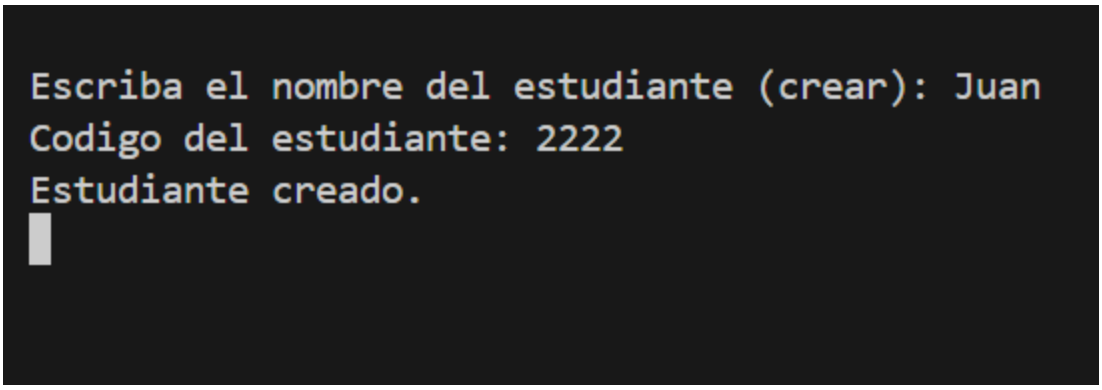
1. La nota total de un curso se calcula como la suma de estas cuatro componentes.
2. Estructura Estudiante: Contiene la información personal del estudiante y su progreso académico. Incluye el nombre (string), el codigo (string), y un vector (vector<Curso>) que almacena todas las calificaciones (cursos) del estudiante en las distintas materias.

Este modelo es considerado "simple pero efectivo" ya que encapsula toda la información necesaria (cursos y calificaciones) dentro de la estructura principal de Estudiante, y maneja la lista de estudiantes y cursos globalmente.



```
1  struct Estudiante {
2      string nombre;
3      string codigo;
4      vector<Curso> cursos;
5  };
```

Código: Estructura de estudiante Fuente: elaboración propia en C++.



Resultado: Muestra a Estudiante y a el Código de estudiante Fuente: elaboración propia en Windows Terminal

3.2. Gestión del Estado Global

La aplicación mantiene su estado a través de un conjunto de variables globales, lo que permite un acceso unificado a los datos desde cualquier función del programa:

La persistencia y el acceso a los datos se manejan centralizadamente mediante el uso intensivo de variables globales, accesibles directamente desde cualquier función del programa:

• **estudiantes:** Un vector<Estudiante> global que actúa como el repositorio principal para todos los datos de los estudiantes y sus calificaciones.

• **cursos:** Un vector<string> global que lista los nombres de todos los cursos disponibles en la sección.

• **carrera/Sección:** Una variable global (string) que almacena la carrera o sección a la que pertenece el archivo de datos actual.

• **Gestión de Archivos:** Las variables globales archivoSeleccionado (string) y extension (const string, ".csv") definen el archivo en uso para la carga y guardado de datos.

Casi todas las funciones de negocio, como crearCurso, eliminarEstudiante o guardarDatos, interactúan y modifican directamente estas variables globales.

```cpp
struct Curso {
    string nombre;
    float zona;
    float parcial1;
    float parcial2;
    float examen;    // ...existing code...
    string carreraSeccion = "";
    // ...existing code...
};
```

Código: Estructura de curso y variable de carrera sección, Fuente: elaboración propia en C++.

```cpp
void guardarDatos() {
    string nombreArchivo = archivoSeleccionado + extension;
    ofstream archivo(nombreArchivo);

    if (!archivo.is_open()) {
        cout << "Error al guardar los datos" << endl;
        return;
    }
```

Código: Archivo seleccionado + Extensión con una condición, Fuente: elaboración propia en C++.

3.3. Paradigma de Programación

El código está implementado siguiendo un enfoque puramente procedural. Las estructuras (structs) se utilizan como contenedores pasivos de datos (PODs), sin métodos asociados. La lógica operativa está encapsulada en funciones globales que manipulan directamente las variables de estado. A pesar de su naturaleza procedural, el sistema hace uso de características de C++ moderno, como std::function y expresiones lambda, para implementar callbacks que permiten la actualización dinámica de la interfaz de usuario durante la entrada de texto.

```cpp
struct Curso {
    string nombre;
    float zona;
    float parcial1;
    float parcial2;
    float examen;    // ...existing code...
    string carreraSeccion = "";
    // ...existing code...
};

struct Estudiante {
    string nombre;
    string codigo;
    vector<Curso> cursos;
};

vector<Estudiante> estudiantes;
string carreraSeccion = "";
vector<string> cursos;
```

Código: contenedores pasivos de datos y también tenemos un Mecanismo de Almacenamiento (PODs), **Fuente: elaboración propia en C++.**

3.4. Gestión de Persistencia de Datos

El sistema asegura la durabilidad de la información mediante un mecanismo de almacenamiento en archivos de texto plano con un formato estructurado.

El formato estructurado de los archivos facilita tanto la lectura como la escritura de los datos, permitiendo operaciones como agregar, modificar o eliminar registros sin pérdida de información.

Cada registro se organiza siguiendo un patrón predefinido (por ejemplo, identificador, nombre del estudiante, curso, nota, etc.), lo que posibilita un manejo ordenado y eficiente de la información.

```cpp
bool archivoVacio(string nombreArchivo) {
    ifstream archivo(nombreArchivo, std::ios::binary | std::ios::ate);

    if(!archivo.is_open()){
        cout << "No se pudo abrir el archivo" << endl;
        return true;
    }

    return archivo.tellg() == 0;
}

// Guarda todos los datos en el archivo CSV
void guardarDatos() {
    string nombreArchivo = archivoSeleccionado + extension;
    ofstream archivo(nombreArchivo);

    if (!archivo.is_open()) {
        cout << "Error al guardar los datos" << endl;
        return;
    }
```

Código: lectura/escritura de CSV y guardador de archivos con una condición de que no se puede abrir el archivo, Fuente: elaboración propia en C++.

3.5. Flujo de Carga y Guardado

1. **cargar Datos:** Esta función lee el archivo .csv seleccionado, valida el marcador COMPATIBLE y reconstruye el estado de la aplicación poblando las variables globales estudiantes, cursos y carrera/sección.
2. **guardar Datos:** Se invoca después de casi cualquier operación que modifique el estado de la aplicación (crear, editar o eliminar un estudiante/curso; actualizar calificaciones). Esta función reescribe completamente el archivo .csv con los datos actuales, garantizando una alta integridad y persistencia de los cambios.

```
void cargarDatos() {
    string nombreArchivo = archivoSeleccionado + extension;

    // Verificar si el archivo está vacío o no existe
    if (archivoVacio(nombreArchivo)) {
        return; // No hay datos que cargar
    }

    ifstream archivo(nombreArchivo);

    if (!archivo.is_open()) {
        cout << "Error al cargar los datos" << endl;
        return;
    }

    string linea;
```

Código: Cargar Datos, Fuente: elaboración propia en C++.

```
void guardarDatos() {
    string nombreArchivo = archivoSeleccionado + extension;
    ofstream archivo(nombreArchivo);

    if (!archivo.is_open()) {
        cout << "Error al guardar los datos" << endl;
        return;
    }
```

Código: Guardar datos, Fuente: elaboración propia en C++.

3.6. Selección y Creación de Archivos

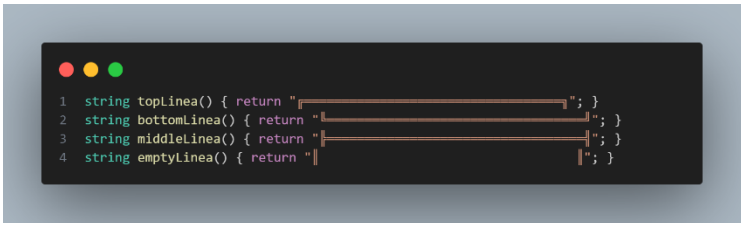
4. Interfaz de Usuario y Navegación

- La interacción con el usuario se gestiona a través de una interfaz de consola robusta y dinámica.

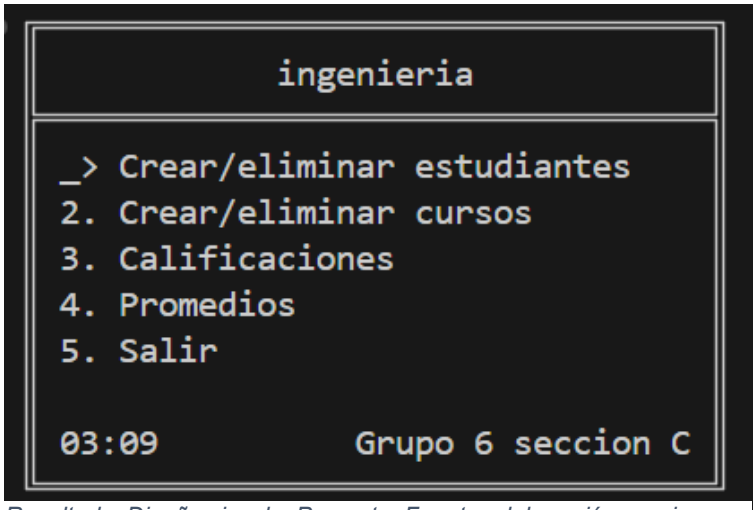
1. **Diseño Visual:** La interfaz está construida con menús basados en texto, utilizando caracteres especiales para dibujar bordes y tablas (ej. topLinea, padding, tabla De Tres), lo que proporciona una experiencia de usuario limpia y organizada.
2. **Navegación:** La navegación por los menús se realiza con las teclas Flecha arriba y Flecha abajo, y la selección con Enter. Esta lógica está centralizada en la función navegarMenu.

3. **Entrada de Datos Interactiva:** Una de las características más destacadas es la captura de entrada de teclado carácter por carácter (mediante _getch). Esto, combinado con funciones de callback, permite que la interfaz se actualice en tiempo real a medida que el usuario escribe. Por ejemplo, al buscar un estudiante, la lista visible se filtra dinámicamente con cada tecla pulsada.

1)



Código: Interfaz, Fuente: elaboración propia en C++.



Resultado: Diseño visual o Proyecto, Fuente: elaboración propia en Windows Terminal

2)

```
void navegarMenu(int limite, int exitOption, function<void(int)> dibujarMenu, function<void(int)> seleccionarOpcion)
{
    char caracter;
    int opcion = 0;

    while ((caracter = _getch())) {
        if (caracter == 72) { // Flecha arriba
            opcion = moverOpcion(opcion - 1, limite);
        }
        if (caracter == 80) { // Flecha abajo
            opcion = moverOpcion(opcion + 1, limite);
        }

        if (caracter == '\r') {
            seleccionarOpcion(opcion);
            if(opcion == exitOption || exitOption == -1) break;
        }

        dibujarMenu(opcion);
        cout << "Opcion: " << opcion + 1 << endl;
    }
```

Código: Navegación con los mandos Flechas de dirección arriba y abajo, Fuente: elaboración propia en C++.

3)

```
string crearTexto( string texto, function<void(string)> ejecutable) {
    char caracter;

    while ((caracter = _getch()) != '\r' || texto.length() < 0) {
        if(caracter == '\b' && texto.length() > 0){
                texto = texto.substr(0, texto.length() - 1);
        } else if(caracter != '\b' && caracter != '\r') {
            texto += caracter;
        }

        ejecutable(texto);
    }

    return texto;
}
```

Código: Ejemplo al crear un texto con función getch, Fuente: elaboración propia en C++.

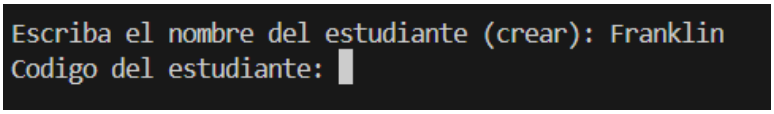
4.1. Funcionalidades Principales del Sistema

El sistema se organiza en cuatro módulos operativos principales, accesibles desde un menú de inicio.

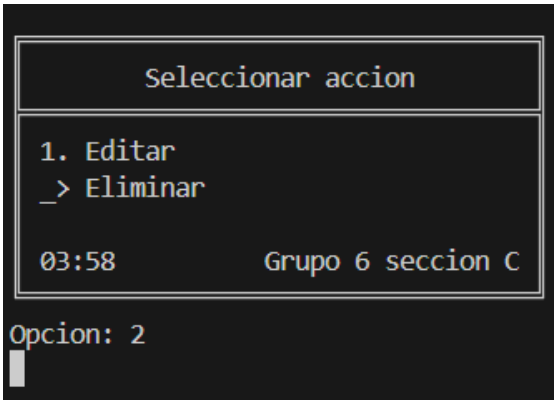
4.2. Módulo de Estudiantes

Este módulo (logicaMenuCrearEstudiante) gestiona el ciclo de vida de los registros de estudiantes.

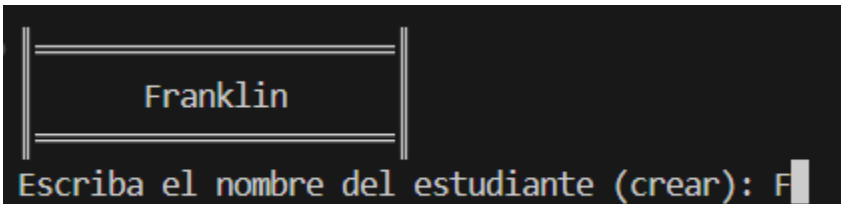
1. **Creación**: Si el nombre introducido por el usuario no corresponde a ningún estudiante existente, el sistema solicita un código y crea un nuevo registro. Al crearse, al estudiante se le asignan automáticamente todos los cursos existentes en el sistema, con todas sus calificaciones inicializadas en 0.0f.
2. **Edición y Eliminación:** Si el nombre introducido coincide con un estudiante existente, se presenta un submenú que permite editar el nombre y código del estudiante o eliminar su registro por completo.
3. **Búsqueda Dinámica:** La función filtrarEstudiantes permite una búsqueda insensible a mayúsculas que actualiza la lista de estudiantes mostrada en tiempo real.



Resultado: Creación del estudiante, Fuente: elaboración propia en Windows Terminal



Resultado: Edición y eliminación de estudiante Fuente: elaboración propia en Windows Terminal



Resultado: búsqueda Dinámica con tal solo la primer letra mayúscula o minúscula; Fuente: elaboración propia en Windows Terminal

4.3. Módulo de Cursos

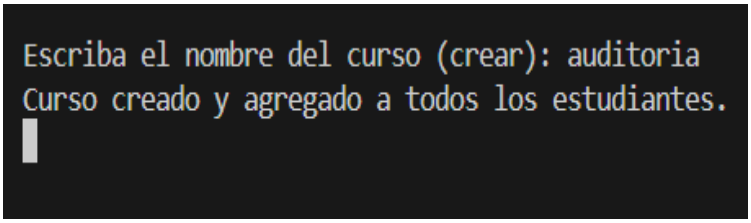
De manera análoga al módulo de estudiantes, esta sección (logicaMenuCrearCurso) gestiona los cursos.

1. **Creación:** Al crear un nuevo curso, este se añade a la lista global cursos y, crucialmente, se agrega al registro de todos los estudiantes existentes, con calificaciones inicializadas en 0.0f.
2. **Edición y Eliminación:** Permite editar el nombre de un curso existente o eliminarlo. La eliminación de un curso implica su remoción de la lista global y de los registros de todos los estudiantes, manteniendo la consistencia de los datos.

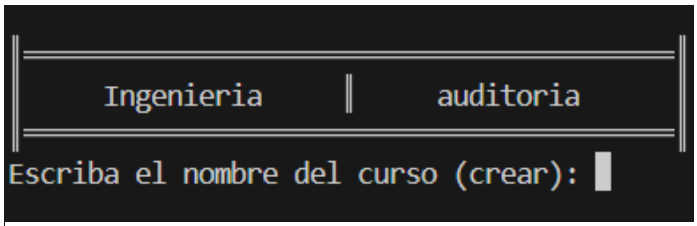
```cpp
void crearCurso(const string& nombre) {
    cursos.push_back(nombre);

    // Agregar el curso a todos los estudiantes con valores inicializados en 0
    for (auto& estudiante : estudiantes) {
        Curso c;
        c.nombre = nombre;
        c.zona = 0.0f;
        c.parcial1 = 0.0f;
        c.parcial2 = 0.0f;
        c.examen = 0.0f;
        estudiante.cursos.push_back(c);
    }
    guardarDatos();
}
```

Código: para crear la función curso, Fuente: elaboración propia en C++.



Resultado: Creando un curso y automáticamente se agrega a los estudiantes, Fuente: elaboración propia en Windows Terminal



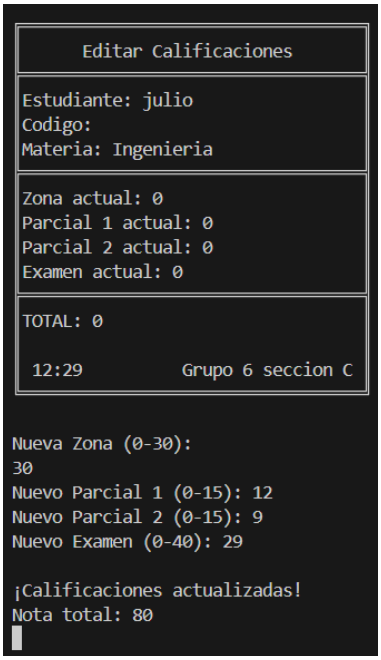
Resultado: ya creados los cursos nos muestra los cursos ya creados Fuente: elaboración propia en Windows Terminal

4.4. Módulo de Calificaciones

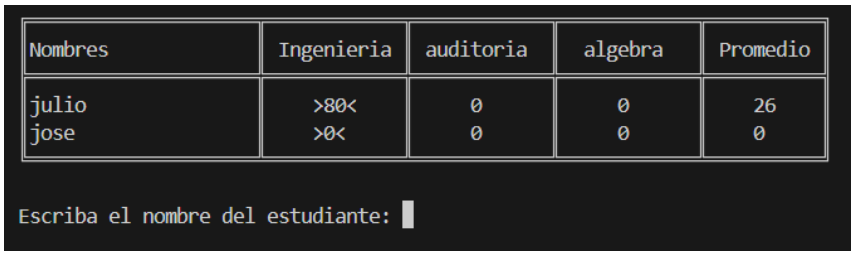
Este módulo (lógica menú Calificaciones) permite la entrada y modificación de notas.

1. **Flujo de Selección:** El proceso requiere que el usuario primero seleccione una materia (validando que exista) y luego un estudiante.
2. **Visualización de Tabla:** Mientras se selecciona al estudiante, el sistema muestra una tabla completa (mostrarTablaCalificaciones) con todos los estudiantes y sus notas finales calculadas para cada curso, destacando la columna del curso previamente seleccionado.
3. **Edición de Notas:** Una vez seleccionado el estudiante, se presenta una pantalla para ingresar las nuevas calificaciones para las cuatro categorías: Zona, Parcial 1, Parcial 2 y Examen. La función calcularNotaTotal suma estos componentes para obtener la nota final del curso.

Resultado: ya creado el curso nos muestra para escoger el que deseamos, Fuente: elaboración propia en Windows Terminal



Resultado: ingresamos las notas del estudiante Fuente: elaboración propia en Windows Terminal

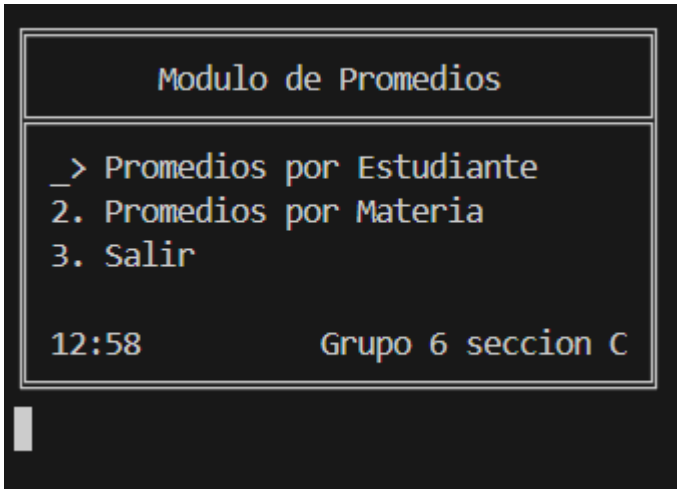


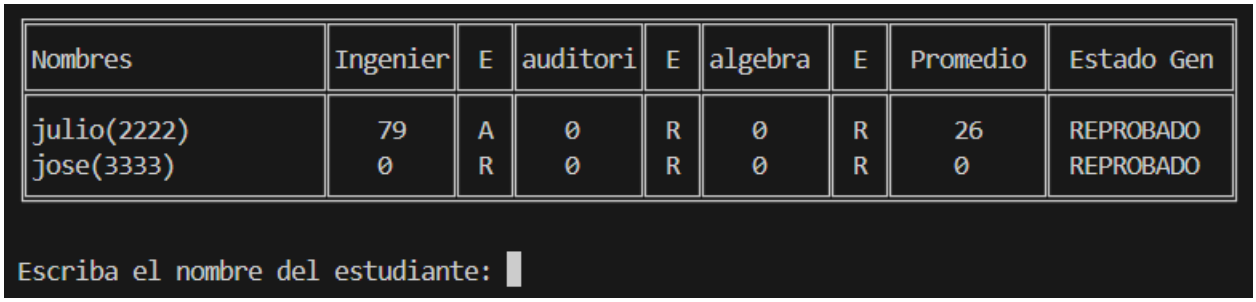
Resultado: nos muestra la tabla con los datos ingresados aun así solo se ha ingresado en el curso (ingeniería) Fuente: elaboración propia en Windows Terminal

4.5. Módulo de Promedios y Reportes

Esta sección (logicaMenuPromedios) se enfoca en la visualización y exportación de resultados académicos. La nota mínima para aprobar es 61.

1. **Promedios por Estudiante:**
2. **Visualización:** Muestra una tabla detallada para un estudiante específico, incluyendo su nota final y su estado de aprobación ("A" para Aprobado, "R" para Reprobado) en cada materia, así como su promedio general y estado general.
3. **Exportación:** Permite generar un archivo CSV (promedios_[nombre]_[codigo].csv) con toda esta información para el estudiante seleccionado.
4. **Promedios por Materia:**
5. **Visualización:** Muestra una tabla para una materia seleccionada, listando a todos los estudiantes, su nota final en esa materia, su estado de aprobación y el promedio general de la clase para esa materia.
6. **Exportación:** Permite generar un archivo CSV (promedios_[materia].csv) que contiene los datos de todos los estudiantes para esa materia, incluyendo al final una fila con el promedio de la clase.

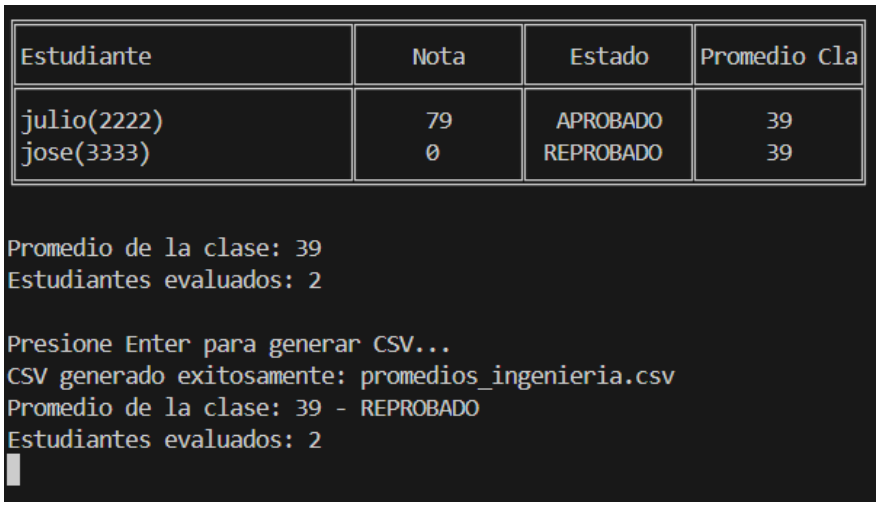




Nombres	Ingenier	E	auditori	E	algebra	E	Promedio	Estado Gen
julio(2222)	79	A	0	R	0	R	26	REPROBADO
jose(3333)	0	R	0	R	0	R	0	REPROBADO

Escriba el nombre del estudiante:

Resultado: Tabla de reprobado ¿Por qué?, aun se a ingresado datos en las demás clases, Fuente: elaboración propia en Windows Terminal



Estudiante	Nota	Estado	Promedio Cla
julio(2222)	79	APROBADO	39
jose(3333)	0	REPROBADO	39

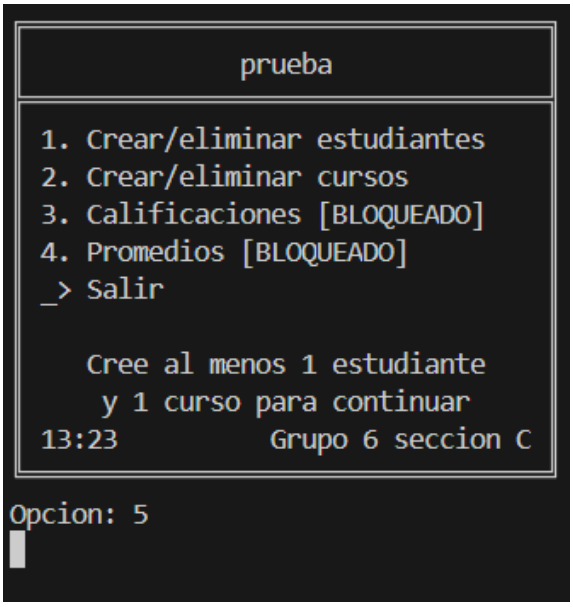
Promedio de la clase: 39
Estudiantes evaluados: 2

Presione Enter para generar CSV...
CSV generado exitosamente: promedios_ingenieria.csv
Promedio de la clase: 39 - REPROBADO
Estudiantes evaluados: 2

Resultado: Tabla con datos ingresados en la materia y nos muestra que si aprobó la materia Fuente: elaboración propia en Windows Terminal

4.6. Lógica de Acceso Condicional

El sistema implementa una salvaguarda para garantizar la estabilidad. Los módulos de "Calificaciones" y "Promedios" permanecen bloqueados ([BLOQUEADO]) en el menú principal hasta que se haya creado al menos un estudiante y un curso. La variable booleana puedeAcceder controla este acceso, previniendo operaciones sobre vectores vacíos que podrían causar errores en tiempo de ejecución. Si el usuario intenta acceder a estas secciones, se muestra un mensaje de advertencia (menuAccesoBloqueado).



Resultado: La tabla nos muestra que al no haber ingresado datos de estudiantes no podemos ingresar a calificaciones ni a promedios, Fuente: elaboración propia en Windows Terminal

5. CONCLUSIONES

- Este sistema presenta una arquitectura procedural sólida y funcional, sustentada en estructuras de datos simples pero efectivas. Usamos struct para representar entidades académicas (estudiantes y cursos) permitiendo una organización clara y directa de la información. A través de variables globales, logramos un manejo centralizado del estado, lo que simplifica las operaciones y garantiza la coherencia de los datos en todas las funciones del programa. Aunque no utiliza clases ni orientación a objetos, el diseño mantiene un equilibrio entre simplicidad y funcionalidad.
- El mecanismo de almacenamiento en archivos .csv constituye un componente esencial que asegura la durabilidad de la información. La estructura personalizada del archivo, junto con las funciones cargarDatos() y guardarDatos(), garantiza la integridad de los registros en cada sesión. Este enfoque, aunque sencillo, demuestra un manejo adecuado de la persistencia de datos, permitiendo al sistema operar sin necesidad de bases de datos externas y facilitando la portabilidad de la información académica.

- Este sistema también ofrece una experiencia interactiva moderna gracias al uso de funciones de navegación con teclas y actualizaciones en tiempo real mediante callbacks y lambdas. El diseño de menús, filtrado dinámico y reportes exportables en formato CSV se convierten en una herramienta eficiente para la gestión académica. Además, la implementación de accesos condicionales refuerza la estabilidad del programa al prevenir errores de ejecución, evidenciando un enfoque responsable en la validación y control de flujo.

6. E-GRAFÍAS

Programación en C++

https://learn.microsoft.com/en-us/cpp/standard-library/fstream?view=msvc-170

Persistencia de Datos (CSV)

https://www.codigazo.com/en-c/guardar-datos-archivo-texto-en-c

https://es.scribd.com/document/755159802/Ficheros-en-c

Interfaces de Consola

https://en.wikipedia.org/wiki/Curses_(programming_library)

https://lib.rs/command-line-interface

Manejo de Errores

https://icarus.cs.weber.edu/~dab/cs1410/textbook/9.Classes_And_Objects/exceptions.html

https://learn.microsoft.com/es-es/dotnet/csharp/fundamentals/exceptions/

https://www.ibm.com/docs/es/i/7.5.0?topic=only-try-blocks-c

C++ Moderno

https://www4.ujaen.es/~fmartin/apuntesC++.pdf

https://learn.microsoft.com/es-es/cpp/cpp/welcome-back-to-cpp-modern-cpp?view=msvc-17