ACTIVIDAD COLABORATIVA 2: EJERCICIOS PRÁCTICOS DE GESTIÓN DE LA MEMORIA

2024

Ricardo Vicente

German Sierra

Jorge Saldaña

Esteban Velasquez

1. Tabla de Contenidos

2. Explicación Funcional del Código

Función main().

Objetivo: Configura los usuarios iniciales y da acceso al sistema mediante el login.

Instancias y Asignaciones:

- Se crean las instancias de UsuarioAdministrador, Profesor y Estudiante.
- Se crea un **Aula** y se asigna un Profesor y varios Estudiante al aula. Esta organización de objetos refleja la estructura básica del sistema educativo representado.

Liberación de Memoria: Al finalizar el main, se liberan manualmente los objetos creados en memoria dinámica.

Función de Autenticación login().

- Se solicitan username y password.
- Se realiza una búsqueda en las listas de UsuarioAdministrador, Profesor, y Estudiante.

Redirección de Usuarios: Tras validar el acceso, el usuario es redirigido a la interfaz correspondiente (Aplicacion()).

Variables y Tipos:

- Variables principales: username y password.
- Estructuras de datos: Listas de std::list para contener objetos de cada tipo de usuario.

Opciones de Menú: Muestra un menú que permite al estudiante acceder a recursos como la Mochila Virtual, la Calculadora Avanzada, o el Juego de Adivinar.

1. **MochilaVirtual:** Aunque no implementa funcionalidades adicionales, la estructura está preparada para desarrollo futuro.
2. **CalculadoraAvanzada:** Ofrece funciones avanzadas de cálculo y lógica matemática, llamando a la clase "**CalculadoraAvanzada**".
3. **JuegoAdivinar:** Ejecuta el método juegoAdivinar() para jugar a adivinar un número aleatorio.

Función Aplicacion (Profesor).

Opciones de Menú: Permite al profesor ver los estudiantes de su **aula** (mostrarEstudiantes()) y obtener información general del **aula** (mostrarAula()).

Variables y Métodos:

1. **profesor.getAula().mostrarEstudiantes():** imprime una lista los estudiantes del aula.
2. **profesor.getAula().mostrarAula():** Proporciona detalles del aula, como el nombre y los recursos.

Función Aplicación (UsuarioAdministrador).

Opciones de Menú: Proporciona al administrador la posibilidad de ver, crear o eliminar usuarios.

1. **mostrarUsuarios():** Llama a admin.mostrarUsuarios para listar los usuarios del sistema.
2. **crearUsuario()**: Permite crear usuarios de tres tipos. Al elegir el tipo (Administrador, Profesor o Estudiante), se solicita información del usuario y se añade a la lista correspondiente.
3. **eliminarUsuarioPorID():** Permite eliminar un usuario mediante su ID, eliminándolo de la lista tras verificar su existencia.

Juego Adivinar Número.

Permite al usuario jugar a un juego durante la ejecución de la aplicación.

- **juegoAdivinar():** genera un número aleatorio del 1 al 100 y el usuario deberá intentar adivinarlo. Por cada intento, el programa especificará si el número es mayor o menor que el intento actual. Cuando el usuario acierte el número, se le felicitará y se mostrará por pantalla el número de intentos que haya usado para adivinarlo.

Calculadora Avanzada.

Accesible a cualquier usuario, la calculadora estará dividida en 2 partes:

- **Operaciones básicas** (simple suma, resta, multiplicación, división y potencias de enteros o doubles);
- **Operaciones con matrices**;

Las operaciones básicas, manejadas a través de la clase “**Operacion**”, utiliza los métodos de suma, resta, multiplicación y división para devolver el resultado double entre 2 valores solicitados al usuario.

Para las operaciones con matrices, se creará una nueva clase “**Matriz**”, que contendrá , y que actuará como una ;
El constructor de la matriz creará un array con un tamaño igual a las filas, y por cada

espacio de las filas se creará un nuevo array de tipo double con un tamaño igual a las columnas. La razón por la que se trabajará con datos double es para ampliar el rango de números con los que se podrá operar.
Del otro lado, el destructor se encargará de destruir cada columna array de la matriz y luego, la fila array

El método "**añadirDato**" agrega, en la posición dada, el valor aportado, aunque este método es más usado dentro de otros métodos para crear nuevas matrices
Del otro lado el método "**crearMatriz**" pedirá al usuario que ingrese cada valor de cada posición de la matriz, usando el método anterior para añadirlos, e imprimiendo por consola tanto el dato ingresado como su posición.

Aunque uno de los extras establecidos para la calculadora fue el cálculo del determinante de una matriz y la creación de la matriz traspuesta a una dada, para la mejor ejecución del cálculo de la división entre matrices, se priorizó la creación de métodos que puedan hacerlo, ya que la división de dos matrices implica la inversión de la segunda y, por lo tanto, la creación de su adjunta y traspuesta dividido por su determinante.

En el método "**trasponer**", se inicializará una nueva matriz traspuesta, cuyo número de filas y columnas será invertido respecto a la matriz de la que heredará, finalmente devolviendola.

El método "**determinante**", el cual devolverá un double, se basará en la reducción de la matriz en sus partes más pequeñas, siempre que las filas y columnas de dicha matriz sean iguales, y por lo tanto, que sea cuadrática, sino será imposible; si la matriz estará formada por un solo elemento, se devolverá dicha posición como resultado; en el caso de que sea una matriz 2 * 2, se restará el producto del primero y el cuarto elemento con el del segundo y tercero, siendo esta la matriz regular más pequeña por la cual el método se basará.
En caso de que sea una matriz N * N, siendo N > 2, se entrará un bucle de tamaño igual al número de filas (el cual afectará e invertirá cada vez el signo de la multiplicación), y se declarará una nueva matriz interna, con una fila y columna menos a la anterior, cuyos valores corresponderán a cada valor de la matriz anterior que no estén en su primera fila o en la columna en la que se está trabajando; una vez calculada toda la matriz interna, se llamará al método determinante usando dicha matriz, y se multiplicará el valor que devolverá por el valor de la primera columna y el signo; este bucle se repetirá hasta que la matriz interna creada sea de 2 * 2, de tal forma que su determinante se pueda conseguir con facilidad, y por tantas veces como el número de filas, hasta que devuelva el determinante.

En cuanto al método "**adjunta**", su proceso será bastante similar al del determinante: las filas de la matriz deben ser iguales a las columnas, se inicializará una nueva matriz igual, el signo se verá afectado por los bucles principales (aunque

esta vez será por la suma de la i y la j, ya que el signo que se obtendrá dependerá de la suma del número de fila y columna del elemento en que se aplica, y como en el anterior, se creará una matriz más pequeña por cada posición de la anterior, creada con los elementos en las posiciones que los bucles superiores no ocupan al momento; a esta matriz se le deberá sacar su cofactor, siendo el producto de su determinante multiplicado por el signo del momento, valor que se le añadirá a la matriz adjunta a devolver.

Obtenidas todos estos métodos, se implementarán en el nuevo método "**inversa**", en el que se declarará una nueva variable determinante, igual al método determinante de la matriz, y se aplicará a una nueva matriz instanciada tanto el método trasponer como el adjunta; una vez hecho esto, se dividirá cada elemento de la matriz por su determinante, y se devolverá la matriz inversa.

Los métodos "**sumarMatrices**" y "**restarMatrices**" tendrán un trabajo similar: los dos se llamarán con la matriz 'A', y tendrán como atributo la matriz 'B', y si estas son del mismo tamaño, se devolverá una matriz resultado cuyos valores serán la suma o resta de los correspondientes de las otras 2.

En la función de "**multiplicarMatrices**", siempre que el número de columnas de la primera matriz sea igual al número de filas de la segunda matriz atributo, se inicializará una matriz resultado (con todos los elementos puestos a 0 para evitar errores de suma); por cada fila de la primera matriz, y por cada columna de la segunda, al elemento de la matriz resultado cuyas filas y columnas corresponden con las anteriores, se le aplicará la fórmula de suma del producto entre un elemento de la fila de la primera matriz y de la columna de la segunda. Tras estos bucles, se devolverá la nueva matriz.

Gracias a los métodos anteriormente creados, el método "**dividirMatrices**" es bastante elemental: si las dos matrices son cuadráticas e iguales, se declarará una matriz inversa a la segunda llamando a dicho método y se llamará al de multiplicarMatrices para su operación, devolviendo así el resultado

Nota: durante la solicitud de los valores o números de filas y columnas de las matrices u operaciones básicas, si el input aportado no es del tipo de dato similar al que requiere la función, notificará del error al usuario y le pedirá que ingrese otra vez nuevos valores, hasta que sean correctos

Aula.

El código de la clase **Aula**, tiene el objetivo de gestionar un **aula** asociada a un profesor y sus estudiantes. Esta clase incluye un identificador (ID), un puntero a un objeto Profesor, y una lista de punteros a objetos Estudiante. La funcionalidad principal del **Aula** es

añadir estudiantes y mostrar la información del **aula**, los estudiantes y sus notas. Es importante resaltar que un **aula** solo tiene un profesor pero muchos estudiantes.

GestorArchivos.

La clase **GestorArchivos** permite gestionar archivos dentro de un directorio especificado por los desarrolladores. Los métodos implementados proporcionan funcionalidades para inicializar y listar archivos, mostrar el contenido de un archivo, eliminar archivos y cargar archivos desde otra ubicación al directorio especificado. Esta clase va mano a mano con la **MochilaVirtual** y funciona como un englobador de las funciones que deben cumplirse directamente con el almacenamiento del computador.

Profesor.

La clase **Profesor** gestiona la administración de un aula y permite al profesor acceder y visualizar la información de sus estudiantes. Esta clase organiza a los estudiantes en una estructura de clase específica, facilitando la gestión eficiente de los datos académicos y personales de cada uno. La clase **Profesor** contiene un conjunto de estudiantes que forman la clase de este profesor. Permite que el profesor visualice tanto el número total de estudiantes inscritos como el listado completo de estudiantes asignados, promoviendo la gestión organizada del aula. Acceso a Información de Estudiantes: A través de la clase Profesor, el profesor puede acceder a los datos de cada estudiante, incluyendo su nombre, apellido, y calificaciones en cada asignatura. Esto permite al profesor llevar un registro de las notas de cada estudiante y facilita una visión completa del rendimiento general de su clase.

Estudiante.

La clase **Estudiante** permite gestionar la información individual de cada estudiante en un **aula**. Sus funciones principales incluyen la carga de datos personales y académicos, así como su asociación con un profesor específico. La clase almacena el nombre, apellido, identificador (ID), nombre de usuario y contraseña de cada estudiante. Al inicializarse, la clase **Estudiante** carga automáticamente las calificaciones de cada asignatura (matemáticas, inglés e historia) desde un archivo de almacenamiento externo. También incluye métodos para mostrar estas calificaciones en un formato claro y organizado, que permite al profesor revisar el desempeño académico de cada estudiante.

La clase **Estudiante** está diseñada para trabajar en conjunto con la clase Profesor, permitiendo que cada estudiante esté vinculado exclusivamente a un profesor. Esto facilita que los datos de los estudiantes sean accesibles sólo para el profesor a cargo de su aula, garantizando una relación uno a uno entre profesor y estudiante.

MochilaVirtual.

La clase **MochilaVirtual** se diseña para servir como una interfaz interactiva que facilita la administración de archivos a través de la clase **GestorArchivos** . Al inicializar una instancia de **MochilaVirtual**, se abre un bucle que permite al usuario acceder a distintas

funcionalidades relacionadas con la gestión de archivos, tales como listar, cargar, abrir o eliminar archivos. Dentro del contexto de esta aplicación, esta clase es una referencia a un almacenamiento estructurado de archivos que el usuario puede manipular mediante inputs en la consola.

El constructor de **MochilaVirtual** invoca el método init, que contiene este bucle de control y gestiona la interacción del usuario con los archivos almacenados en un directorio específico.

3. Explicación de Teoría y su Implementación en el Código.

Abstracción y Tipos Abstractos de Datos (TAD).

Las clases UsuarioAdministrador, Profesor, y Estudiante encapsulan tanto la interfaz pública como la implementación de sus funciones y datos. Esto permite a cada tipo de usuario operar sin conocer la estructura interna de los demás, lo que mejora la organización del código y facilita la extensibilidad del sistema.

Encapsulamiento.

Cada clase tiene sus atributos y métodos privados y controlados. Por ejemplo, los métodos getUsername y getPassword protegen el acceso directo a las credenciales de los usuarios.

Herencia y Especialización.

UsuarioAdministrador y Profesor derivan de la clase base Usuario, manteniendo una estructura común pero con comportamientos específicos para cada uno.

Polimorfismo.

La función Aplicacion() muestra una sobrecarga de funciones, proporcionando una interfaz flexible para cada tipo de usuario.

Genericidad.

Se usa el contenedor std::list para gestionar listas de usuarios de tipo UsuarioAdministrador, Profesor, y Estudiante. Esto permite una implementación flexible.

Almacenamiento Libre y la Pila.

La memoria asignada dinámicamente con new en el almacenamiento libre mientras que variables locales ocupan espacio en la pila.

En la clase **GestorArchivos** podemos ver que las variables miembro direccion y pathArchivos se almacenan en la pila cuando **GestorArchivos** se instancia (tal y como vimos en clase). Según su definición en la documentación de C + +, el uso de std::filesystem::path y std::ifstream permite que los archivos residan en el almacenamiento libre.

Uso de la Memoria Dinámica en la Calculadora.

Cualquier variable creada, tanto para los elementos operandos como los intermedios como los finales a devolverse serán guardados en punteros a sus direcciones de almacenamiento libre; al mismo tiempo, tendrán un destructor que se encargue de eliminar cada instancia creada una vez utilizada.

En la clase **MochilaVirtual** init utiliza un puntero dinámico a **GestorArchivos** para acceder a las funcionalidades de archivos y su almacenamiento es dinámico.

Manejo de excepciones en las Matrices.

En el caso de que los requerimientos para el cálculo de matrices o de valores particulares no se consiga, se notificará al usuario; estas excepciones están presentes en los siguientes métodos:

- **determinante**() y **adjunta**(): la matriz aportada debe ser cuadrática, por lo tanto el número de filas y columnas deben ser iguales;
- **suma**(), **resta**() y **división()**: las dos matrices deben tener las mismas dimensiones;
- **multiplicación**(): la cantidad de filas de la primera matriz debe ser igual a la de las columnas de la segunda;

Punteros.

En las clases **Aula**, **Profesor** y **Estudiante** se usan punteros variables referentes a objetos en vez de almacenar los objetos directamente. Esto permite que el **Aula** trabaje con las referencias en memoria de estos objetos, evitando la duplicación de datos y permitiendo modificaciones en tiempo de ejecución.

4. Manual de Uso del Programa.

Instrucciones de Uso:

El programa cuenta con tres tipos de Usuarios (Administrador, Profesor y Estudiante) que pueden acceder a través de unos logins y passwords determinados. Con el fin de demostrar su funcionalidad es inminente que use los siguientes datos para la ejecución del programa. Cuando se le soliciten las credenciales, ingrese los siguientes datos para iniciar sesión según el tipo de usuario:

- **Administrador**: Username = admin1… Password = 1234
- **Profesor**: Username = juan… Password = 1234
- **Estudiante**: Username = carlos… Password = 1234

Navegación de funcionalidades para cada Usuario:

1. Usuario Administrador

Funcionalidad 1: Ver todos los Usuarios.

1. Tras autenticarse con la información correcta,

```
Estudiante aniadido.
Estudiante aniadido.
Estudiante aniadido.
Username:admin1
 Password:1234
```

2. Verás el siguiente menú:

```
 ----- Bienvenido Andrea Pinzon -----
Recursos:
0) salir.
1) Ver todos los Usuarios.
2) Crear Usuario.
3) Eliminar Usuario.
Escoge un recurso:
1
```

3. Donde al ingresar el valor **1**, verás el listado de todos los usuarios:

```
----- Administradores  -----
#1 ID: 13576 Andrea Pinzon
----- Profesores  -----
#2 ID: 44897 Profe Juan
----- Estudiantes  -----
#3 ID: 89213 Carlos
#4 ID: 95639 Mario
#5 ID: 86396 Juana
```

<u>Funcionalidad 2: Crear un Usuario.</u>

1. Tras autenticarse nuevamente,

```
Estudiante aniadido.
Estudiante aniadido.
Estudiante aniadido.
Username:admin1
 Password:1234
```

2. Verás el siguiente menú:

```
  ----- Bienvenido Andrea Pinzon -----
 Recursos:
 0) salir.
 1) Ver todos los Usuarios.
 2) Crear Usuario.
 3) Eliminar Usuario.
 Escoge un recurso:
```

3. Donde al ingresar el valor **2**, verás las opciones de creación:

```
2
Que tipo de Usuario deseas crear?:
1) Administrador.
2) Profesor.
3) Estudiante.
```

4. Ingresando un número del **1 al 3**, podrás confirmar el tipo de usuario que quieres crear. Le seguirá un menú con información general para cualquier

tipo:

```
Ingresa el nombre del Estudiante: Francisco
 El ID del Estudiante:12345
 El Username del Estudiante:fran
 El Password del Estudiante:1234
 No se en
contraron notas para el estudiante Francisco.
```

5. Se creará el usuario sin notas. Estas se deben inicializar manualmente en el fichero “notas.txt” ubicada en “../data”

Funcionalidad 3: Eliminar un Usuario.

1. Tras autenticarse nuevamente,

```
Estudiante aniadido.
Estudiante aniadido.
Estudiante aniadido.
Username:admin1
 Password:1234
```

2. Verás el siguiente menú:

```
  ----- Bienvenido Andrea Pinzon -----
 Recursos:
 0) salir.
 1) Ver todos los Usuarios.
 2) Crear Usuario.
 3) Eliminar Usuario.
 Escoge un recurso:
```

3. Donde al ingresar el valor **3**, verás el área de eliminación:

```
3
Ingresa el ID exacto del Usuario que deseas eliminar:
```

4. Debes ingresar el ID exacto del usuario a eliminar:

```
Ingresa el ID exacto del Usuario que deseas eliminar:86396
 Estudiante con ID 86396 ha sido eliminado.
```

Funcionalidad 4: Salir.

1. Tras autenticarse nuevamente,

```
Estudiante aniadido.
Estudiante aniadido.
Estudiante aniadido.
Username:admin1
 Password:1234
```

2. Verás el siguiente menú:

```
 ----- Bienvenido Andrea Pinzon -----
Recursos:
0) salir.
1) Ver todos los Usuarios.
2) Crear Usuario.
3) Eliminar Usuario.
Escoge un recurso:
```

3. Donde al ingresar el valor **0**, saldrá del programa:

```
0
Saliendo de aplicacion
```

2. Usuario Profesor

Funcionalidad 1: Ver a todos los estudiantes y sus notas en el aula del profesor.

1. Tras autenticarse como profesor,

```
Estudiante aniadido.
Estudiante aniadido.
Estudiante aniadido.
Username:juan
 Password:1234
```

2. Verás el siguiente menú:

```
 ----- Bienvenido Profe Juan -----
Recursos:
0) salir.
1) Ver Estudiante en mi Aula.
2) Ver Informacion de mi Aula.
Escoge un recurso:
```

3. Al digitar **1**, se mostrará el menú con la información:

```
1
----- Estudiantes en Aula A -----
#1 Carlos
        Matematicas: 5
        Ingles: 6.5
        Historia: 7
#2 Mario
        Matematicas: 8
        Ingles: 9
        Historia: 10
#3 Juana
        Matematicas: 4.5
        Ingles: 6
        Historia: 7.5
```

<u>Funcionalidad 2: Ver la cantidad de estudiantes en el aula y una lista de sus nombres.</u>

1. Tras autenticarse como profesor,

```
Estudiante aniadido.
Estudiante aniadido.
Estudiante aniadido.
Username:juan
 Password:1234
```

2. Verás el siguiente menú:

```
 ----- Bienvenido Profe Juan -----
Recursos:
0) salir.
1) Ver Estudiante en mi Aula.
2) Ver Informacion de mi Aula.
Escoge un recurso:
```

3. Al digitar **2**, se mostrará el menú con la información:

```
----- Aula A -----
Numero de estudiantes: 3
#1 Carlos
#2 Mario
#3 Juana
```

Funcionalidad 3: Salir.

4. Tras autenticarse como profesor,

```
Estudiante aniadido.
Estudiante aniadido.
Estudiante aniadido.
Username:juan
 Password:1234
```

5. Verás el siguiente menú:

```
 ----- Bienvenido Profe Juan -----
Recursos:
0) salir.
1) Ver Estudiante en mi Aula.
2) Ver Informacion de mi Aula.
Escoge un recurso:
```

6. Donde al ingresar el valor **0**, saldrá del programa:

```
0
Saliendo de aplicacion
```

3. Usuario Estudiante

Funcionalidad 1: Mochila Digital.

1. Tras autenticarse como estudiante,

```
Estudiante aniadido.
Estudiante aniadido.
Estudiante aniadido.
Username:carlos
 Password:1234
```

2. Verás el siguiente menú:

```
 ----- Bienvenido Carlos -----
Recursos:
0) Salir.
1) Mochila Digital.
2) Calculadora Avanzada.
3) Juego.
Escoge un recurso:
```

3. Al ingresar el dígito **1**, veremos el siguiente menú:

```
1
----- Mochila Digital -----
┌┘Que deseas hacer?:
0) Salir.
1) Ver listado de Archivos.
2) Cargar Archivo.
```

   a. Si ingresamos el valor **1** nuevamente, podremos ver el listado de archivos dentro de "../data/Ficheros"

```
1
----- Lista de Archivos -----
        1): Fichero Prueba:
        2): Nota Examen1:


┌┘Que deseas hacer?:
0) Salir:
1) Abrir un Archivo:
2) Eliminar un Archivo:
```

      i. Si ingresamos **1** nuevamente, podremos abrir un archivo y ver su contenido a través de su índice:

```
Ingresa el indice del archivo:1
 ----- Archivo Abierto -----
        Fichero Prueba: Lorem Ipsum.
```

      ii. Si ingresamos **2**, eliminamos un archivo a través de su índice:

```
Ingresa el indice del archivo: 2
 Archivo "NotaExamen.txt" eliminado.
```

      iii. Si ingresamos **0**, saldrá del programa:

```
0
Saliendo de aplicacion
```

4. Si ingresamos **2**, podremos cargar un archivo pasando la dirección en el computador del archivo objetivo:

```
2
Ingresa la ubicacion de tu archivo:C:\Users\esvel\OneDrive\Desktop\Code\C++\Unidad2\Grupo1_AC2\data\notas.txt
 File copied successfully to "../data/Ficheros\\notas.txt"
```

5. Si ingresamos **0**, saldrá del programa:

```
0
Saliendo de aplicacion
```

Funcionalidad 2: Calculadora Digital.

1. Tras autenticarse como estudiante,

```
Estudiante aniadido.
Estudiante aniadido.
Estudiante aniadido.
Username:carlos
 Password:1234
```

2. Verás el siguiente menú:

```
 ----- Bienvenido Carlos -----
Recursos:
0) Salir.
1) Mochila Digital.
2) Calculadora Avanzada.
3) Juego.
Escoge un recurso:
```

3. Al ingresar el dígito **2**, veremos el siguiente menú:

```
2
Ingresa que tipo de operacion que se quiere realizar:
1) Operaciones Basicas
2) Operaciones con Matrices
3) Salir
```

   a. Al ingresar el dígito **1**, entramos en el menú de operaciones entre números básicos:

```
1
Ingresa la operacion que se quiere realizar:
1) Suma
2) Resta
3) Multiplicacion
4) Division
5) Potencia
```

i. Introduciendo cualquiera de estos índices obtendremos el mismo proceso pero con el resultado escogido. Ejemplo:

```
Ingresa la operacion que se quiere realizar:
1) Suma
2) Resta
3) Multiplicacion
4) Division
5) Potencia
3
Ingresa el primer y segundo numero:
11
12
Resultado: Resultado: 132
```

b. Al ingresar el dígito **2**, entramos en el menú de operaciones entre matrices:

```
Ingresa que tipo de operacion que se quiere realizar:
1) Operaciones Basicas
2) Operaciones con Matrices
3) Salir
2
----- Calculadora Avanzada Matrices -----
1. Sumar Matrices
2. Restar Matrices
3. Multiplicar Matrices
4. Dividir Matrices
5. Transponer Matriz
6. Calcular Determinante
7. Calcular Inversa
8. Salir
Elige una opci├│n:
```

i. Introduciendo cualquiera de estos índices obtendremos el mismo proceso pero con el resultado escogido. Ejemplo:

```
Elige una opci├│n:3
 Multiplicar dos matrices:
Ingresa el n├╢mero de filas:2
 Ingresa el n├╢mero de columnas:2
 Ingresa cada valor de la matriz:
2
Valor agnadido en [0][0]: 2
5
Valor agnadido en [0][1]: 5
4
Valor agnadido en [1][0]: 4
3
Valor agnadido en [1][1]: 3
Ingresa el n├╢mero de filas:2
 Ingresa el n├╢mero de columnas:2
 Ingresa cada valor de la matriz:
1
Valor agnadido en [0][0]: 1
2
Valor agnadido en [0][1]: 2
3
Valor agnadido en [1][0]: 3
5
Valor agnadido en [1][1]: 5
Matriz Impresa:
17 | 29 |
13 | 23 |
```

4. Si ingresamos **3**, saldrá del programa:

```
0
Saliendo de aplicacion
```

<u>Funcionalidad 3: Juego.</u>

1. Tras autenticarse como estudiante,

```
Estudiante aniadido.
Estudiante aniadido.
Estudiante aniadido.
Username:carlos
 Password:1234
```

2. Verás el siguiente menú:

```
 ----- Bienvenido Carlos -----
Recursos:
0) Salir.
1) Mochila Digital.
2) Calculadora Avanzada.
3) Juego.
Escoge un recurso:
```

3. Al ingresar el dígito **3**, entraremos directamente al juego:

```
Escoge un recurso:
3
Adivina el numero entre 1 y 100.
Introduce un numero:
```

4. Donde digitando números finalizamos el juego.

```
Adivina el numero entre 1 y 100.
Introduce un numero:50
 El numero es menor que 50.
Introduce un numero:25
 El numero es menor que 25.
Introduce un numero:12
 El numero es mayor que 12.
Introduce un numero:19
 El numero es menor que 19.
Introduce un numero:17
 El numero es menor que 17.
Introduce un numero:13
 Felicidades! Has adivinado el numero en 6 intentos.
Opcion no válida.
```

Funcionalidad 4: Salir.

1. Tras autenticarse como estudiante,

```
Estudiante aniadido.
Estudiante aniadido.
Estudiante aniadido.
Username:carlos
 Password:1234
```

2. Verás el siguiente menú:

```
 ----- Bienvenido Carlos -----
Recursos:
0) Salir.
1) Mochila Digital.
2) Calculadora Avanzada.
3) Juego.
Escoge un recurso:
```

3. Donde al ingresar el valor **0**, saldrá del programa:

```
0
Saliendo de aplicacion
```