Mini-Mat

(X Aniversario)

PRÁCTICA DE FUNDAMENTOS DE PROGRAMACIÓN

Se pide realizar un programa en C que, mediante una interfaz de línea de comandos, también llamada **CLI** (*Command Line Interface*), admita comandos por teclado que luego deberá analizar y, en su caso, ejecutar, mostrando cuando sea necesario el resultado (o el mensaje de error) que sea pertinente, dicho resultado se mostrará también en la misma consola. En todos los casos, cuando el usuario introduzca un comando, el prompt de la interfaz y el propio comando deberán imprimirse por pantalla en **color blanco**. Cuando el programa imprima por pantalla un resultado después de ejecutar correctamente un comando del usuario, dicho resultado se imprimirá en **color verde** y, cuando el programa deba responder con un mensaje de error, este aparecerá en pantalla en **color rojo**.

De manera general el programa deberá ejecutarse siguiendo el flujo de control que se indica en el siguiente pseudocódigo:

```
1   Imprimir en pantalla datos del alumno:
    - apellidos, nombre
    - correo electrónico

2   Mostrar el prompt en color blanco

3   Leer el comando introducido por el usuario por teclado

4   Analizar el comando del usuario:

5       ⇒ Si el comando es incorrecto se indica
          con un mensaje de error en rojo y se vuelve al paso 2

6       ⇒ Si el comando no se puede ejecutar se indica
          con un mensaje de error en rojo y se vuelve al paso 2

7       ⇒ Si el comando es nulo o vacío no se hace nada
          y se vuelve al paso 2

8       ⇒ Si el comando es "quit" se libera toda la memoria
          dinámica que el programa esté ocupando y se finaliza
          mostrando por pantalla el mensaje "exit ok" en verde

9       ⇒ Se ejecuta el comando según las especificaciones
          y se vuelve al paso 2. Si es pertinente mostrar un
          resultado por pantalla se imprime en color verde
```

Específicamente, el programa que se pide realizar ha de hacer operaciones con matrices de números, pudiendo asignar matrices a variables para luego poder operar, bien con las variables, bien con las matrices indicadas explícitamente. Una matriz explícita se escribirá entre paréntesis '**(...)**', indicando cada fila de números de la matriz separadas por el carácter '**|**' (barra vertical o "*pipe*"). Véanse los siguientes ejemplos:

Comando		Matriz	Filas x Columnas
`(1) ⇔ ( 1 ) ⇔ (  1  )`	⇔	$\begin{pmatrix}1\end{pmatrix}$	1 x 1
`(  1  2  3  | 4 5 6)`	⇔	$\begin{pmatrix}1 & 2 & 3\\4 & 5 & 6\end{pmatrix}$	2 x 3
`(5.5 -2.4  |  3.33  0  )`	⇔	$\begin{pmatrix}5.5 & -2.4\\3.33 & 0\end{pmatrix}$	2 x 2
`(0  2.37  -3.764  9.9  -5.22)`	⇔	$\begin{pmatrix}0 & 2.37 & -3.764 & 9.9 & -5.22\end{pmatrix}$	1 x 5
`(  1 | -2 |  3  | -4)`	⇔	$\begin{pmatrix}1\\-2\\3\\-4\end{pmatrix}$	4 x 1
`(1.1 1.2 1.3 | 2 2 2 | -3 3 0.5)`	⇔	$\begin{pmatrix}1.1 & 1.2 & 1.3\\2 & 2 & 2\\-3 & 3 & 0.5\end{pmatrix}$	3 x 3

Nótese que en la columna "Comando", para separar los elementos de cada matriz, se han usado cantidades arbitrarias de espacios y/o tabuladores, en todos los casos esos comandos son correctos. Por otra parte, todas las filas de una matriz deben tener siempre el mismo número de elementos, sería incorrecta una matriz declarada como la siguiente:

```
( 1  2  3  |  4  5  |  6  7  8 )
```

Por simplicidad, en adelante se usarán los términos **M1**, **M2**, **M3**, etc… para hacer referencia a una matriz explícitamente declarada de forma correcta, es decir, siguiendo la sintaxis indicada anteriormente.

Como ya se ha dicho, el programa debe poder hacer operaciones con matrices como por ejemplo, asignar una matriz a una variable (los caracteres "**:>**" son el prompt proporcionado por el programa, línea 2 del pseudocódigo anterior):

```
:> var = M1
```

La operación anterior crea una variable de nombre '**var**' a la cual asigna la matriz explícita **M1**. Si ya tenemos una variable con una matriz asignada, es posible asignar esta primera variable a una nueva, por ejemplo así:

```
:> var2 = var
```

Cualquier cadena formada por caracteres alfanuméricos (‘**var**’, ‘**var2**’, ‘**AbC**’, ‘**qhDs**’, etc…) será considerado un nombre válido para una variable, con la restricción de que una variable no puede ocupar más de 15 caracteres de longitud, no puede empezar por un número (‘*1var*’ no es una variable correcta), y tampoco debe coincidir con ninguno de los comandos del programa (dichos comandos se indicarán más adelante).

En general, las operaciones a realizar con matrices serán de dos tipos, de asignación y de muestra de resultados. Una operación de asignación siempre empezará con una variable seguida del signo ‘**=**’ (igual) y a continuación podrá tener una expresión que devuelva una matriz, para simplificar, en los siguientes ejemplos ya no se incluye el prompt al principio de los comandos, ejemplos:

```
var3 = M1              o también  → var3 = var1
var4 = M1 + M2                    → var4 = var1 + M2
var5 = M1 - M2                    → var5 = M1 - var3
var6 = M1 * M2                    → var6 = var1 * var2
```

Cualquiera de las expresiones anteriores creará una nueva variable (‘**var3**’, ‘**var4**’, etc…) a la cual se le asignará el resultado de la expresión que aparece a la derecha del signo igual. Si la variable que se indica al principio del comando (a la izquierda del signo igual) ya existía por alguna operación anterior, el nuevo valor calculado (la nueva matriz) sustituirá al antiguo. La realización de cualquiera de estas operaciones anteriores no producirá ninguna salida o mensaje por pantalla.

Para hacer algunas operaciones con matrices hay que verificar que las filas y columnas de dichas matrices son adecuadas para la operación, cuando no sea así, no se realizará ninguna acción, solo se responderá con un **mensaje de error** adecuado. También se responderá con un **mensaje de error** cuando los comandos no estén bien construidos (tal y como se indica en los ejemplos) o si a la derecha del signo igual se utiliza una variable que no se haya creado anteriormente (variable no declarada).

El segundo tipo de operaciones, las de muestra de resultados, son análogas a las que ya se han visto pero sin la parte de la asignación. En este caso, tras realizar la operación que corresponda en cada caso, siempre que no haya ningún error que impida realizar la operación, dicha **matriz resultante se mostrará por pantalla** en forma de filas y columnas debidamente tabulada. Las operaciones de muestra de resultados válidas son:

```
M1                 o también  → var1
M1 + M2                       → var1 + M2
M1 - M2                       → M1 - var3
M1 * M2                       → var1 * var2
```

Como en el caso anterior, hay que verificar que las dimensiones de las matrices son adecuadas para realizar cada operación y, cuando se usen variables, comprobar que

dichas variables han sido declaradas con anterioridad. Si todo es correcto, tras realizar la operación que corresponda, se **mostrará en pantalla la matriz resultante**, usando 2 decimales para cada número y tabulando correctamente los valores.

Adicionalmente a las operaciones anteriores, el programa debe poder ejecutar una serie de comandos que se van a indicar a continuación, dichos comandos no podrán ser usados como nombres de variables. Nótese que estos comandos que se van a indicar a continuación a veces irán acompañados de uno o varios parámetros, los comandos serán incorrectos si, en cada caso no van seguidos de los parámetros que se indica, cuando esto ocurra el programa deberá responder con un **mensaje de error**.

`quit`

Este comando no va acompañado de ningún parámetro, al ejecutarse deberá liberarse toda la memoria dinámica que se esté usando, y finalizar el programa mostrando el mensaje `exit ok` (volviendo el color a blanco para la consola de windows).

`det M1`

Calcula el determinante de la matriz **M1** (también puede ser una variable) y **muestra el resultado por pantalla**. Si no se pudiera realizar el cálculo se responderá con un **mensaje de error** apropiado.

`product n M1`

Calcula el producto de un escalar **n** (número real) por una matriz **M1** (explícita o variable). El resultado es otra matriz, por tanto este comando se podrá usar tanto en una operación de asignación como para mostrar resultados.

`save <nombre_fichero>`

Este comando va seguido de un parámetro que es una cadena de caracteres. Guarda todas las variables que hay en memoria en un fichero con dicho nombre siguiendo un determinado formato que se indicará más adelante (si el fichero ya existe lo machaca). Si no hay variables se indica con un **mensaje de error**.

`load <nombre_fichero>`

También va seguido de una cadena de caracteres. Lee el fichero con el nombre indicado y lo carga en memoria. Si el fichero no existe o no tiene el formato esperado (ver más adelante) no se carga nada y se indica con un **mensaje de error**. Si el fichero se lee correctamente se borran todas las variables anteriores y se sustituyen por las recién leídas del fichero.

`view`

Va sin parámetros, **muestra un listado por pantalla**, en líneas consecutivas, con los nombres de las variables que se han creado y que hay almacenadas en memoria. Si no hay variables simplemente se indica que **“No hay variables!!”**.

Fichero de datos

Los comandos '**save**' y '**load**' escriben y leen, respectivamente, un fichero que debe contener todas las variables que se hayan declarado durante una ejecución del programa, por cada variable deberá guardarse su nombre y, en una línea consecutiva, todo los valores de la matriz con 8 decimales. por ejemplo, una variable que se hubiera creado con el siguiente comando:

```
v1 = (1 -7 3.1415926535 | 0.5 -3.333 2.7182818284)
```

en el fichero se almacenará del siguiente modo:

```
v1
(1.00000000 -7.00000000 3.14159265 | 0.50000000 -3.33300000 2.71828183)
```

Si en memoria hubiera almacenadas 5 variables de nombres '**a**', '**v1**', '**zz**', '**mat1**' y '**matriz2**', el fichero de datos resultante sería así:

```
a
( <valores de la matriz 'a'> )
v1
( <valores de la matriz 'v1'> )
zz
( <valores de la matriz 'zz'> )
mat1
( <valores de la matriz 'mat1'> )
matriz2
( <valores de la matriz 'matriz2'> )
```

Salida por pantalla

Todas las operaciones de muestra de resultados deben mostrar un valor (un número o una matriz) por pantalla en color verde, salvo cuando la operación, por algún error, no se pueda ejecutar. Siempre que se produzca un error deberá mostrarse un mensaje en color rojo, lo más apropiado posible, es decir, que proporcione información acerca del error que se haya detectado.

El programa no deberá mostrar otros mensajes adicionales no solicitados. Se penalizará aquellos trabajos que muestren por pantalla mensajes que no se hayan solicitado explícitamente o que no se correspondan con información de algún error detectado en algunos de los comandos escritos por el usuario.

En ningún caso el programa deberá detener su ejecución con mensajes del tipo "***Pulse una tecla para continuar***", ni otros similares.

Sobre la implementación en C

Para guardar en memoria una matriz se deberá utilizar la estructura "**TMatriz**" tal cual se define a continuación:

```
typedef struct
{
      int f, c;    // filas y columnas
      double **m;  // doble puntero a la memoria de la matriz
} TMatriz;
```

Es necesario mantener una lista de variables para ir guardando cada nueva variable que se va declarando. La estructura "**TVar**" será el nodo donde guardar cada variable en una lista doblemente enlazada:

```
typedef struct variable
{
      char nomVar[16];
      TMatriz *mat;
      struct variable *sig, *ant;
} TVar;
```

La lista de variables propiamente dicha estará representada por la estructura "**TVars**";

```
typedef struct
{
      int numVars;
      TVar *primera;
} TVars;
```

Se penalizará una mala gestión de memoria dinámica. Se recomienda implementar diversas funciones para toda la gestión de memoria necesaria para el desarrollo del programa, por ejemplo:

- Crear una matriz
- Eliminar una matriz
- Buscar una variable en la lista de variables
- Guardar una variable nueva en la lista de variables
- Destruir una variable de la lista de variables
- Destruir toda la lista de variables
- Gradar la lista de variables en un fichero
- Leer valores desde un fichero y crear la lista de variables
- etc…

Ampliación 1
Examen Noviembre 2023

Ejercicio 1 (2.5 puntos)

Modificar comando "**view**" para que muestre el listado de variables indicando, junto al nombre de cada variable, también sus dimensiones (fil x col) y los valores de la primera fila de la matriz con dos decimales, p.e.:

```
var1 (4 x 6) : (1.00  -33.26  9999.00  78.00)
```

Ejercicio 2 (2.5 puntos)

Modificar el comando "**product**" para que sea conmutativo, es decir, estas dos operaciones serán válidas y equivalentes:

```
product n M1
product M1 n
```

Ejercicio 3 (2.5 puntos)

Modificar el comando "**load**" para que admita el parámetro opcional "**over**". Cuando esté presente el fichero cargado se solapa con las variables existentes, añadiéndose y no sustituyendo a las ya existentes, salvo que coincidan.

```
load [over] <nombre_fichero>
```

Ejercicio 4 (2.5 puntos)

Nuevo operador para realizar el producto escalar de dos matrices de [1xn] una fila y 'n' números. Usar el carácter '**$**' como operador de la operación (la operación devuelve un número real, no una matriz).

```
M1 $ M2
(m1 y m2 pueden ser cualquier tipo de matriz de una sola fila)
```

 # En todos los ejercicios, cuando corresponda mostrar una salida por pantalla, se hará en color verde, salvo que sea un mensaje de error, que se hará en color rojo.

Avisos:

- La duración del examen es de 3 horas
- Si la práctica contiene algún error, este podría restar a la nota final del examen
- **MUY IMPORTANTE:** MOSTRAR POR PANTALLA AL PRINCIPIO DEL PROGRAMA QUE EJERCICIOS ESTÁN HECHOS Y CUÁLES NO **(no hacerlo restará 1 punto)**
  - **Ejercicio 1: HECHO / SIN HACER**
  - **Ejercicio 2: . . .**

Ampliación 2
Examen Enero 2024

Ejercicio 1 (2.5 puntos)

Modificar el prompt para que muestre el número de variables definidas en el programa, se debe actualizar también cuando se ejecute una instrucción **`load`**.

```
:[n]>
```

Ejercicio 2 (2.5 puntos)

Modificar la aplicación para que el comando **`view`** muestre el listado de variables ordenado alfabéticamente.

Ejercicio 3 (2.5 puntos)

Añadir un nuevo comando **`transp`** para calcular la matriz transpuesta, debe funcionar tanto en una operación de muestra de resultados como de asignación.

```
transp (1 2 | 3 4 | 5 6)          matrizT = transp mat
```

Ejercicio 4 (2.5 puntos)

Añadir un nuevo operador '**`&`**' que debe unir dos matrices del mismo número de filas en una única matriz (muestra de resultados como de asignación).

```
mat = (1 2 | 3 4 | 5 6) & m1          m2 & (7 | 8 | 9)
```

Avisos:

- La duración del examen es de 3 horas
- Si la práctica contiene algún error, este podría restar a la nota final del examen
- **MUY IMPORTANTE:** MOSTRAR POR PANTALLA AL PRINCIPIO DEL PROGRAMA QUE EJERCICIOS ESTÁN HECHOS Y CUÁLES NO **(no hacerlo restará 1 punto)**
  - **`- Ejercicio 1: HECHO / SIN HACER`**
  - **`- Ejercicio 2: . . .`**