**Tarea 1 EDD Sec. 2**

Juan De la Barra

Lucas Vigorena

**Introducción:**

Parte del código fue hecho en parte en Visual Studio Code en Windows con el compilador gcc usando la extensión code runner, pero luego fue preferido usar replit para facilitar el avance cooperativo.

**Sobre los contenidos:**

Antes que nada, se incluyen las tres librerías permitidas para poder ser usadas.

Por enunciado, el código define dos estructuras: una para vectores y otra para matrices. La estructura de vector contiene un campo para el tamaño del vector y otro para los datos en sí, mientras que la estructura de matriz contiene campos para el número de filas y columnas, así como un puntero a los datos en sí, que son un arreglo de arreglos de números de punto flotante.

Sobre la estructura del código, la función inicial *main* llama a otra función orquestadora sobre la cual se decide que hacer. Antes de usar las funciones que requiere la tarea se han creado tres funciones extra cuyo propósito es probar que las funciones requeridas funcionen correctamente, activándolas todas en tres grupos:

* probarFuncionesVectores: las funciones de vectores (1, 3, 5)
* probarFuncionesMatrix: las funciones de matrices (2, 4, 6)
* probarFuncionesOperaciones: y al final las funciones de operaciones (7 a 10)

De esta forma logramos probar el código más organizado y mejor enfocado. Cabe mencionar que sabemos que funcionan bien ya que se corren todas las funciones y retornan todas 0, por lo que no hay error en ningún proceso.

**Sobre las funciones requeridas:**

1. **create\_vector\_from\_file**: recibe como parámetro un puntero a *char* que indica el nombre del archivo a abrir. Luego se declara un puntero a un objeto FILE y se utiliza la función *fopen* para abrir el archivo en modo lectura.

Si fopen devuelve NULL, retorna 1 ya que hubo un error y no se pudo leer el archivo. Luego lee los datos línea por línea utilizando la función fgets y divide los datos en números de punto flotante utilizando la función strtok.

Si se pudo leer correctamente, se aumenta la dimensión del vector en 1, se asigna más memoria utilizando "realloc" para que el vector tenga capacidad para almacenar el nuevo elemento y finalmente se guarda el número en la última posición del vector. Así se almacenan los números leídos en un arreglo dinámico en la estructura de vector, que se devuelve como resultado.

1. **create\_matrix\_from\_file:** es similar a la anterior, pero en este caso lee una matriz de un archivo. Se declara una variable MAX\_LINE\_SIZE que contendrá la línea máxima del archivo, con un tamaño máximo definido en la macro 1000. En el siguiente bloque de código, se utiliza la función fgets para leer el archivo línea por línea y almacenar cada fila de la matriz en el campo data de M.

Dentro de un bucle while, fgets se llama con los siguientes argumentos: la variable MAX\_LINE\_SIZE, el tamaño máximo permitido para cada línea, y el puntero al archivo fp. fgets devuelve un puntero al inicio de la línea leída, por lo que se utiliza esta variable para iterar a través de los tokens de la línea.

Dentro del while, se incrementa la variable M.row\_dim, lo que indica que se ha leído una nueva fila de la matriz. Luego, se utiliza *realloc* para redimensionar la memoria asignada al campo data de M, para que pueda almacenar una matriz con la nueva dimensión de filas. Después, se utiliza la función *strtok* para dividir la línea en tokens (números separados por espacios), y se asigna cada token a la matriz M.data y se retorna la matriz M.

1. **print\_vector:** toma como entrada una estructura de tipo vector recorre todos los elementos del vector utiliza un ciclo for y los imprime uno por uno.
2. **print\_matrix**: Tiene un ciclo for que recorre todas las filas de la matriz y dentro de este ciclo hay otro que recorre todas las columnas imprimiendo el valor de cada entrada de la matriz utilizando la función printf. Para esto, se accede al valor de la entrada de la matriz M.data[i][j], donde i es el índice de la fila y j es el índice de la columna.

Después de imprimir todos los valores de la fila actual, la función inserta un salto de línea con printf("\n") para pasar a la siguiente fila. Al finalizar los ciclos for, la función no realiza ninguna acción adicional.

1. **destroy\_vector:** Utiliza la función free para liberar la memoria reservada por malloc y realloc en la creación del vector. Primero, la función llama a free para liberar la memoria reservada por malloc y realloc para cada entrada del vector, utilizando un ciclo for para recorrer todas las entradas del vector. Luego, llama a free para liberar la memoria reservada por malloc y realloc para el propio campo data del vector.
2. **destroy\_matrix**: Liberar la memoria utilizada por una matriz, la función destruye primero cada una de las filas de la matriz, utilizando la función free(). Para esto, utiliza un bucle for que itera sobre las filas de la matriz, y dentro de este bucle, utiliza la función free() para liberar la memoria utilizada por la fila correspondiente.

Finalmente, la función libera la memoria utilizada por el puntero a las filas de la matriz, utilizando la función free().

1. **transpose\_matrix:** toma como entrada una matriz M y devuelve una nueva matriz que corresponde a la transpuesta de la matriz M. Para ello, la función primero crea una nueva matriz N con el número de filas de M igual al número de columnas de M y el número de columnas de N igual al número de filas de M. A continuación, la función recorre la matriz M y asigna los valores de cada posición M[i][j] a la posición N[j][i].
2. **sum\_matrix\_matrix:** toma dos matrices M y N y devuelve su suma. La suma se calcula verificando que las matrices tengan las mismas dimensiones, en caso contrario, se imprime un mensaje de error y se devuelve la matriz M. Para crear la matriz de la suma se recorren ambas matrices y se suman las entradas correspondientes para guardarlas en la matriz O.
3. **mult\_matrix\_matrix:** recibe dos matrices M y N y retorna una nueva matriz O que corresponde al producto matricial. El producto se calcula verificando que las dimensiones de las matrices sean compatibles para multiplicarse, creando una nueva matriz con la dimensión adecuada y realizando la operación matricial estándar.
4. **mult\_matrix\_vector:** toma una matriz y un vector y devuelve su producto. El producto se calcula verificando que las dimensiones de la matriz y el vector sean compatibles para multiplicarse. Si las dimensiones son compatibles, crea un nuevo vector w que será el resultado de la multiplicación de la matriz M y el vector v. Se multiplican los elementos de cada fila de la matriz por los elementos correspondientes del vector y se suman los productos. El tamaño del nuevo vector w será el número de filas de la matriz M. Finalmente, la función devuelve un puntero al nuevo vector w.