Tarea 1 EDD

Juan De la Barra

Lucas Vigorena

Sec.2

**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. 4. **print\_vector** y **print\_matrix**: imprimen vectores y matrices en la salida estándar. Ambas funciones simplemente iteran a través de los datos y los imprimen en la salida estándar.
2. 6. **destroy\_vector** y **destroy\_matrix**: liberanla memoria asignada a vectores y matrices respectivamente. Estas funciones utilizan la función free para liberar la memoria asignada a los datos almacenados en la estructura de vector o matriz.
3. **transpose\_matrix:** toma una matriz y devuelve su transpuesta. La transpuesta se calcula creando una nueva matriz con el número de filas y columnas invertidas, y luego transponiendo los datos de la matriz original.
4. **sum\_matrix\_matrix:** toma dos matrices y devuelve su suma. La suma se calcula verificando que las matrices tengan las mismas dimensiones, creando una nueva matriz con la misma dimensión y sumando los datos de las dos matrices originales.
5. **mult\_matrix\_matrix:** toma dos matrices y devuelve su 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.
6. **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, creando un nuevo vector con la dimensión adecuada y realizando la operación de multiplicación de matriz y vector estándar.