UNIVERSIDAD ADOLFO IBAÑEZ
FACULTAD DE INGENIERÍA Y CIENCIAS

Tarea 1
TICS311: Estructura de Datos y Algoritmos

Fecha de Entrega: 05 de Abril 2023, 23:59 hrs.

El objetivo de esta tarea es que se familiarice con el lenguaje C. Específicamente, profundizaremos el manejo de punteros y el manejo de archivos.

Instrucciones

1. La tarea se puede hacer en grupos de a lo más 3 personas. Debe respetar los grupos inscritos en la planilla de Webcursos.
2. Se adjunta un archivo `main.c` con la definición de los structs y funciones a implementar. Puede agregar a este archivo cualquier función que necesite. Debe subir este archivo a Webcursos con sus respuestas. No suba ningún otro archivo `.c`.
3. Junto con el archivo `main.c`, **debe** subir un informe indicando el compilador que utilizó y explicando brevemente la lógica detrás de la implementación de cada una de sus funciones.
4. Se aceptan atrasos. Se descontará 1.0 pts por cada día de atraso.
5. Notar que hay un punto extra, luego la nota máxima es un 8.0.
6. **Importante:** La implementación la deben hacer ustedes mismos sin usar librerías externas. Por supuesto, pueden buscar información en internet como ayuda, pero asegúrense de entender el código que están escribiendo. Recuerde que debe explicar su implementación en el informe. Las únicas librerías permitidas son las básicas: `stdio.h, stdlib.h, string.h`.

NumC: librería para vectores y matrices

Su misión es implementar en C una pequeña librería que ofrezca funcionalidades básicas de vectores y matrices. Para esto consideraremos dos structs `vector` y `matrix`:

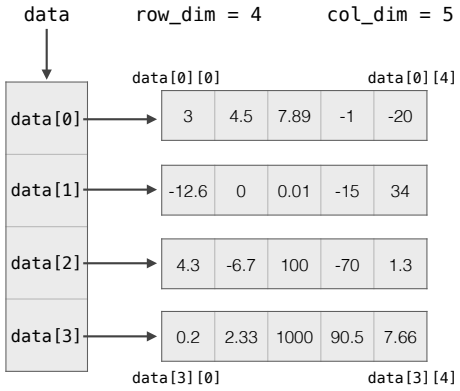
```c
typedef struct v {
    float *data;
    int dim;
} vector;

typedef struct m {
    float **data;
    int row_dim;
    int col_dim;
} matrix;
```

El struct `vector` tiene un campo `float *data` el cual contiene las entradas del vector. Notar que `data` corresponde a un arreglo de floats. El campo `int dim` indica la dimensión del vector. Similarmente, el struct `matrix` tiene un campo `float **data` el cual contiene las entradas de la matriz. En este caso, `data` corresponde a un arreglo de arreglos de floats cada uno de los

cuales corresponde a una fila de la matriz. Los campos `row_dim` y `col_dim` indican la cantidad de filas y columnas de la matriz, respectivamente. La siguiente figura ilustra cómo se representa una matriz con el struct `matrix`.

$$\begin{bmatrix} 3 & 4.5 & 7.89 & -1 & -20 \\ -12.6 & 0 & 0.01 & -15 & 34 \\ 4.3 & -6.7 & 100 & -70 & 1.3 \\ 0.2 & 2.33 & 1000 & 90.5 & 7.66 \end{bmatrix}$$



Debe implementar las siguientes funciones:

1. (1.0 pts) `vector create_vector_from_file(char *filename)`: Crea un vector a partir del archivo `filename`. Retorna el nuevo vector. La convención para escribir un vector en un archivo es la siguiente. El archivo tiene una línea con los floats separados por el caracter *espacio*. Al final de esta línea hay un salto de línea (caracter `\n`). No hay nada más en el archivo. Notar que su función debe deducir automáticamente la dimensión del vector.

2. (1.0 pts) `matrix create_matrix_from_file(char *filename)`: Crea una matriz a partir del archivo `filename`. Retorna la nueva matriz. La convención para escribir una matriz en un archivo es la siguiente. El archivo tiene una línea por *cada* fila de la matriz. Cada línea consta de floats separados por el caracter *espacio*. Al final de cada línea hay un salto de línea (caracter `\n`). No hay nada más en el archivo. Notar que su función debe deducir automáticamente las dimensiones de la matriz.

3. (0.5 pts) `void print_vector(vector v)`: Imprime el vector v en pantalla. La convención para imprimir vectores es la misma descrita arriba.

4. (0.5 pts) `void print_matrix(matrix A)`: Imprime la matrix A en pantalla. La convención para imprimir matrices es la misma descrita arriba.

5. (0.5 pts) `void destroy_vector(vector v)`: Libera la memoria utilizada por el campo `data` del vector v.

6. (0.5 pts) `void destroy_matrix(matrix A)`: Libera la memoria utilizada por el campo `data` de la matrix A.

7. (1.0 pts) `matrix transpose_matrix(matrix A)`: Retorna una nueva matriz que corresponde a la transpuesta de la matriz A.

8. (0.5 pts) `matrix sum_matrix_matrix(matrix A, matrix B)`: Retorna una nueva matriz que corresponde a la suma de las matrices A y B. Si las dimensiones de las matrices A y B son incompatibles, debe imprimir un mensaje de error y retornar la matriz A.

9. (1.0 pts) `matrix mult_matrix_matrix(matrix A, matrix B)`: Retorna una nueva matriz que corresponde al producto de las matrices A y B (producto clásico de matrices, **no** producto coordenada a coordenada). Si las dimensiones de las matrices A y B son incompatibles, debe imprimir un mensaje de error y retornar la matriz A.

10. (0.5 pts) `vector mult_matrix_vector(matrix A, vector v)`: Entrega un puntero a un nuevo vector que corresponde al producto de la matriz A con el vector v (por la derecha, es decir, Av). Si las dimensiones de la matriz A y el vector v son incompatibles, debe imprimir un mensaje de error y retornar el vector v.

Manejo de archivos

Para abrir un archivo, puede usar la función `fopen` de la librería `stdio.h`. Esta función recibe un `char *` con el nombre del archivo y otro `char *` con el modo (por ejemplo, `"r"` es lectura, y `"w"` escritura). La función retorna un `FILE *`, es decir, un puntero a un objeto de tipo `FILE` (este tipo está definido en `stdio.h` y encapsula el objeto archivo). Para cerrar un archivo después de usarlo, use la función `fclose`.

Una forma útil de leer un archivo es línea por línea usando la función `fgets`. Esta función recibe un arreglo de chars `buffer` en dónde se almacenará la línea (recuerde que una línea es un string, es decir, una secuencia de caracteres terminado en el null terminator), un número `n` que indica la cantidad máxima de caracteres que se copiarán a `buffer` (incluyendo el null terminator), y un `FILE *` apuntando al archivo. Cada llamada de `fgets` rellena `buffer` con la línea actual del archivo (se incluye el caracter de salto de linea `\n` al final). Si ya no quedan más líneas o hay algún error, `fgets` retorna `NULL`. Vea el archivo `ejemplo.c` para un ejemplo de uso.

Finalmente, puede dividir un string `texto` en substrings según un separador (el espacio por ejemplo) usando la función `strtok` de `string.h`. Para obtener el primer substring haga la llamada `strtok(texto, separador)`. Para obtener los substrings siguientes de `texto` debe hacer la llamada `strtok(NULL, separador)` por cada substring. La función `strtok` retorna `NULL` cuando ya no quedan más substrings que entregar.

Para ver un ejemplo de `strtok` junto con `fgets` vea el archivo `ejemplo.c`.

Observaciones y hints

1. Asumiremos que el largo máximo de una línea de archivo es `MAX_LINE_SIZE`. Esta es una macro definida en el archivo `main.c` y su valor por defecto es 1000.

2. Observe que cuando define un nuevo vector o matriz y lo retorna hacia afuera en una función, el espacio de memoria al que apunta el campo `data` debe vivir en memoria dinámica.

3. La memoria utilizada por `data` debe ser la justa y necesaria para que caigan todas las entradas del vector/matriz. En general no asumiremos ninguna cota superior en las dimensiones de los vectores/matrices.

4. Puede servir utilizar la función `atof` de `stdlib.h` que recibe un string (`char *`) y retorna el float descrito por el string.