Trabajo práctico integrador

Seminario de lenguajes opción “C” 2020

Se pide desarrollar una biblioteca (*library*) que permite trabajar con matrices de valores numéricos e implementar un programa que la utilice para cargar matrices desde archivos, realizar operaciones algebraicas con ellas y almacenar los resultados nuevamente en archivos.

Las fechas de entrega, coloquio de defensa y re-entrega se publicarán por la plataforma de Cátedras.

Para la aprobación del trabajo, el mismo debe cumplir con lo solicitado en el enunciado, estar correctamente modularizado y no presentar errores (de lógica, de acceso a memoria, de acceso a archivos, etc). Notar que hay puntos del enunciado que son para mejorar la nota, como el formato **M2** y las funciones “**inplace**” (ver más adelante la definiciones).

Para el desarrollo del programa se deberá implementar la biblioteca (*library*) correctamente modularizada, tratando de respetar reglas para ocultar la implementación y separarla de la interfaz.

- `matrix_t`: matriz con dimensiones NxM de valores double.
- `list_t`: lista de valores double.

El módulo *matriz* debe cumplir con un interfaz similar a la siguiente (los parámetros y/o sus nombres podrían variar acorde a cómo se implemente el módulo):

```
typedef ... error_t;

typedef ... matrix_t;

 /***
   * Carga la matriz desde un archivo. El resultado queda en “mc”.
   * Ver especificación de formato más adelante.
   */
error_t read_matrix(FILE *fp, matix_t **mc);

 /***
   * Almacena la matriz en un archivo. El origen es “mc”.
   * Ver especificación de formato más adelante.
   */
error_t write_matrix(FILE *fp, const matrix_t *mc);
```

```c
 /***
   * Duplica (copia) la matriz en memoria. La copia queda en “m_dst”.
   */
error_t dup_matrix(const matrix_t *m_src, matrix_t **m_dst);

 /***
   * Realiza la suma algebraica de matrices. El resultado queda
   * en “mb”, “mb = ma + mb”.
   * Debe chequear que las dimensiones sean compatibles para
   * la operación. Sino retornar un error que indique el mismo.
   * NOTA: Esta operación no es necesaria para aprobar, pero permite
   * obtener mejor nota.
   */
error_t sum_inplace(const matrix_t *ma,  matrix_t **mb);

 /***
   * Realiza la suma algebraica de matrices. El resultado queda
   * en “mc” que se crea dentro de la función. , “mc = ma + mb”.
   * Debe chequear que las dimensiones sean compatibles para
   * la operación. Sino retornar un error que indique el mismo.
   */
error_t sum(const matrix_t *ma, const matrix_t *mb¸ matrix_t **mc);

 /***
   * Realiza la multiplicación algebraica de una matriz por
   * un escalar. El resultado queda en “mb”, “mb = a * mb”.
   * Debe chequear que las dimensiones sean compatibles para
   * la operación. Sino retornar un error que indique el mismo.
   * NOTA: Esta operación no es necesaria para aprobar, pero permite
   * obtener mejor nota.
   */
error_t mult_scalar_inplace(double a, matrix_t **mb);

 /***
   * Realiza la multiplicación algebraica de una matriz por
   * un escalar. El resultado queda en “mc”, “mc = a * mb”.
   * Debe chequear que las dimensiones sean compatibles para
   * la operación. Sino retornar un error que indique el mismo.
   */
error_t mult_scalar(double a, const matrix_t mb, matrix_t **mc);
```

```
 /***
   * Realiza la multiplicación algebraica entre matrices.
   * El resultado queda en “mc”, “mc = ma * mb”.
   * Debe chequear que las dimensiones sean compatibles para
   * la operación. Sino retornar un error que indique el mismo.
   */
error_t mult(const matrix_t ma, const matrix_t mb¸ matrix_t **mc);

 /***
   * Genera la matriz identidad de “n*n” en “mc”.
   * El resultado queda en “mc”.
   */
error_t idty_matrix(unsigned int n, matrix_t **mc);

 /***
   * Genera la matriz nula de “n*n” en “mc”.
   * El resultado queda en “mc”.
   */
error_t null_matrix(unsigned int n, matrix_t **mc);

 /***
   * Genera una matriz nueva de “rows*cols” cargando en todos las
   * posiciones el valor escalar “a”. El resultado queda en “ma”.
   */
error_t create_and_fill_matrix(unsigned int rows, unsigned int cols,
dobule a, matrix_t **ma);

 /***
   * Obtiene la cantidad de filas de la matriz “ma”
   */
unsigned int get_rows(const matrix_t *ma);

 /***
   * Obtiene la cantidad de columnas de la matriz “ma”
   */
unsigned int get_cols(const matrix_t *ma);

 /***
   * Carga la fila “pos” de la matriz ma en la lista “l”
   */
error_t get_row(unsigned int pos, const matrix_t *ma, list_t *l);
```

```
 /***
   * Carga la columna “pos” de la matriz “ma” en la lista “l”
   */
error_t get_col(unsigned int pos, const matrix_t *ma, list_t *l);

 /***
   * Pasa la matriz “ma” a la lista “l”
   */
error_t matrix2list(const matrix_t *ma, list_t *l);

 /***
   * Redimensiona la matriz “ma” a las dimensiones “newrows*newcols”
   * Si es mas grande la nueva matriz queda con cualquier valor
   * en las nuevas posiciones. Si es más chica se descartan.
   */
error_t resize_matrix(unsigned int newrows, unsigned int newcols,
matrix_t **ma);

 /***
   * Coloca el valor “value” en la “pos=(row,col)” en la matriz “mc”.
   * Debe chequear las dimensiones.
   */
error_t set_elem_matrix(unsigned int row, unsigned int col, double
value, matrix_t **mc);

 /***
   * Obtiene el valor de la “pos=(row,col)” de la matriz “mc” y
   * lo coloca en “value”.
   * Debe chequear las dimensiones.
   */
error_t get_elem_matrix(unsigned int row, unsigned int col, double
*value, const matrix_t *m);

/***
  * Compara las matrices “ma” y “mb”. Si son iguales retorna
  * distinto de cero o true y sino cero o false
  */
int|bool cmp_matrix(const matrix_t *ma, const matrix_t *mb);
```

```c
/***
  * Libera la memoria usada por la matriz “m”
  */
error_t free_matrix(matrix **m);
```

El programa a desarrollar debe admitir los siguientes argumentos:

- `--in1|-1` *nombre_archivo*
  Indica la primera matriz a cargar en memoria. (Obligatorio)
- `--in2|-2` *nombre_archivo*
  Indica la segunda matriz a cargar en memoria. (Obligatorio, excepto que se use dup, `idty` o `mult_scalar`)
- `--out|-o` *nombre_archivo*
  Nombre del archivo donde se guardará el resultado.
- `--scalar|-s` *número*
  Número en punto flotante por el que se multiplicarán los elementos de la matriz cargada con `--in1`. (Obligatorio si se usa `mult_scalar`).
- `--op|-p` *(dup|sum|mult_scalar|mult|idty|null|cmp)*
  Indica la operación a realizar, que puede ser una de las siguientes:
  - dup: Genera un duplicado de la matriz indicada con `--in1`, en el archivo indicado con `--out`.
  - `sum`: Suma elemento a elemento las matrices indicadas con `--in1` y `--in2`, almacenando el resultado en el archivo indicado con `--out`. Si las matrices no tienen el mismo tamaño, el programa debe retornar el error: `ERROR_INCOMPATIBLE_MATRICES`.
  - `mult_scalar`: Multiplica cada elemento de la matriz indicada con `--in1` por el número indicado con `--scalar`. El resultado queda en el valor del parámetro `--out`.
  - `mult`: Multiplicar la matriz indicada con `--in1` por la indicada con `--in2`, guardando el resultado en `--out`. Si las matrices no son multiplicables por su tamaño el programa debe retornar el error: `ERROR_INCOMPATIBLE_MATRICES`
  - `idty`: genera la matriz identidad compatible con las dimensiones de la matriz indicada con el parámetro `--in1`. Resultado en el parámetro `--out`. La matriz identidad es una matriz cuadrada de NxN donde solo en su diagonal tiene valores “1” y el resto es “0”. Por ejemplo si la matriz de entrada es de 2x3 la identidad será de 3x3 dando como resultado el producto una matriz de 2x3 idéntica a la de entrada.
  - `null`: genera la matriz nula compatible con las dimensiones de la matriz indicada con el parámetro `--in1`. Resultado en el parámetro de `--out`. La matriz cuadrada será de NxN y todos sus valores serán nulos (“0”).
  - `cmp`: compara la matriz indicada con `--in1` con la indicada en `--in2`. Almacena el resultado de la operación en una matriz de 1*1 indicado en el parámetro `--out`.

- `--help`
  Imprime la ayuda que indica cómo utilizar el programa en salida estándar y termina.

Mediante el uso del programa anterior o programas nuevos, y utilizando matrices de diferentes tamaños, realizar ejemplos que muestren al menos 2 de las siguientes propiedades de la suma y al menos 3 de las siguientes propiedades de la multiplicación de las matrices:

Propiedades de la suma (deben mostrarse al menos 2):

Conmutativa de la suma:	A + B = B + A
Asociativa de la multiplicación:	(A + B) + C= A + (B + C)
Identidad Aditiva, matriz nula:	A + 0 = A
Inverso aditivo:	A + -A = 0

Propiedades de la multiplicación (deben mostrarse al menos 3):

Distributiva de mult. por escalar:	c*(A+B) = c*A + c*B, donde c es un escalar
Identidad Multiplicativa, matriz identidad:	I*A = A*I = A. donde I es la matriz identidad
Nulo multiplicativo:	0*A = A*0 = 0, donde 0 es la matriz nula
Nulo multiplicativo por escalar:	c*0 = 0, donde 0 es la matriz nula y c un escalar
Asociativa de la multiplicación:	(A*B)*C = A*(B*C)
Distributiva de la mult. con la suma:	A*(B+C) = A*B + A*C

Quienes hayan desaprobado uno o más parciales, deberán implementar (según se les indique particularmente en cada caso) el soporte para los siguientes parámetros:

- `--calc archivo1 op archivo2 = archivo_salida`
  - Donde op puede ser:
    - + equivale a `sum`
    - * equivale a `mult`
- `--calc archivo1 .* scalar = archivo_salida`
  - Donde "`scalar`" es un número en punto flotante, esta operación equivale a `mult_scalar`.
- `--calc archivo1 id = archivo_salida`
  - `id` equivale a "`idty`"

Formato de los archivos

Los archivos que almacenan las matrices pueden estar en 2 formatos diferentes, que denominaremos **M1** y **M2**. El programa implementado deberá detectar automáticamente el formato de los archivos de entrada y operar en consecuencia.

Es importante aclarar en este punto que para aprobar, solo el formato M1 es obligatorio. El formato M2 es para mejorar la nota o para aquellos casos particulares en que se les solicite hacerlo por adeudar a los temas de los parciales.

El formato **M1** codifica la matriz en un archivo de texto ASCII, con la siguiente estructura:
Un encabezado (*header*) con el valor `M1` y un *newline*. Luego, cero o más comentarios, que son líneas que comienzan un caracter numeral (#) y terminan en *newline*. Después de eso, la cantidad de filas y columnas en 2 enteros separados por un espacio y un *newline*. Y finalmente los valores separados por espacios, *newlines*, *tabs* o cualquier otro caracter "blanco".

Los valores no necesariamente tienen que estar alineados en columnas o filas, ya que se irán cargando en el orden que se lean, por lo que pueden estar todos en una sola linea. Por ejemplo, estas dos entradas son equivalentes:

```
$ cat test.m1
M1
## Esto es un comentario
# Matriz de 2 x 6
2 6
1.2 3.4 4.5 6.6 56.5 0.0
2   3   4.5 345 44   4.44
```

```
$ cat test2.m1
M1
# Matriz de 2 x 6
2 6
1.2 3.4 4.5 6.6 56.5 0.0 2 3 4.5 345 44 4.44
```

Para el caso del formato **M2** el encabezado es el mismo, excepto que en lugar de contener un header con el valor `M1`, contiene el valor `M2` (se mantiene el *newline* posterior, los comentarios opcionales y la línea con las dimensiones de la matriz); pero luego de la dimensión vienen los datos en formato **binario**, uno a continuación del otro. Es decir un `double` seguido del otro en formato dependiente de la arquitectura. Por ejemplo:

```
$ cat test.m2
M2
## Esto es un comentario
# Matriz de 2 x 6
2 6
<Binary data>
```

Contenido visto en la materia Matemática I de 1er año

- http://info.unlp.edu.ar/wp-content/uploads/2018/03/Matematica-1.pdf
- https://onedrive.live.com/?authkey=%21AB2GXzOA1V2EjL4&cid=554E57CDDB4F4A8E&id=554E57CDDB4F4A8E%2130295&parId=554E57CDDB4F4A8E%2130290&o=OneUp