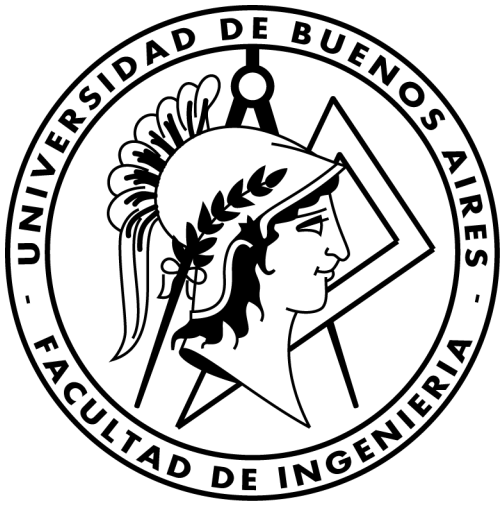

6620 - Organización del computador
Facultad de Ingeniería de la universidad de Buenos Aires

Trabajo Práctico 0
Multiplicador de matrices cuadradas

Luciana Piazzi 90638
Franco M. Di Maria 100498
Gonzalo Marino 97794

1. Documentacion

1.1. Archivos

- matrix.c
- matrix.h
- main.c
- testing.c
- testing.h
- mainTest.c
- testFiles/

1.1.1. matrix.c y matrix.h

Aqui se encuentran la estructura y primitivas de la matriz pedida en el enunciado:

```
typedef struct matrix {
    size_t rows;
    size_t cols;
    double* array;
} matrix_t;

// Constructor de matrix_t
/*
PRE: Recibe un la cantidad (size_t) de filas y de
columnas de la matriz.
POST: Devuelve un puntero a una matriz (matrix_t *)
segun los parametros recibidos, reservada en memoria
dinamica.
Queda a responsabilidad del usuario destruir la matriz
por medio de su destructor.
Si la puntero devuelto es NULL, implica que ocurrio
un error.
*/
matrix_t* create_matrix(size_t rows, size_t cols);

// Destructor de matrix_t
/*
PRE: Recibe un puntero a una matriz (matrix_t)
creada por su constructor.
POST: Destruye la matriz recibida
*/
```

```
void destroy_matrix(matrix_t* m);

// Imprime matrix_t sobre el file pointer fp
/*
PRE: Recibe un flujo de salida (FILE *), y un
matriz (matrix_t *).
POST: Imprime la matriz recibida de la forma:
N C_11 C_12 ... C_nn \n
Donde N es la cantidad de filas
Devuelve un numero (int), menor a cero en caso de
error, o cero en caso de exito.
*/
int print_matrix(FILE* fp, matrix_t* m);

// Multiplica las matrices en m1 y m2
/*
PRE: Recibe dos punteros a matrices (matrix_t *)
POST: Multiplica las matrices recibidas, y devuelve
una nueva matriz, almacenada en memoria dinamica.
Queda a responsabilidad del usuario liberar esta
memoria por medio de sus destructor.
Devuelve NULL en caso de error.
*/
matrix_t* matrix_multiply(matrix_t* m1, matrix_t* m2);
```

Ademas, se definirion dos primitivas adicionales para facilitar la multiplicacion :

```
// Devuelve un arreglo de con los elementos en
// orden de la fila n
/*
PRE: Recibe un puntero a una matriz (matrix_t *),
y el indice de una fila en la misma.
POST: Devuelve un arreglo dinamico con los elementos
de la fila n (double *), de la matriz recibida.
*/
double * matrix_get_row(matrix_t* m, size_t row_n);

// Devuelve un arreglo de con los elementos en
// orden de la columna n
/*
PRE: Recibe un puntero a una matriz (matrix_t *),
y el indice de una columna en la misma.
POST: Devuelve un arreglo dinamico con los elementos
de la columnas n (double *), de la matriz recibida.
*/
double * matrix_get_col(matrix_t* m, size_t col_n);
```

1.1.2. main.c

En este archivo se hace uso de las estructura matriz ya mencionada, para implementar el programa pedido. Ademas, se utilizaron otras funciones auxiliares, para procesar (parsear) las lineas de matrices recibidas por la entrada estandard. A continuacion se describen dichas funciones :

```
// Lee lineas de la entrada estandard
/*
PRE: Recibe un puntero a un numero (size_t *).
POST: Lee de la entrada estandar lineas (hasta
encontrar un \n; y termina al encontrar un EOF),
y la devuelve (char *), almacenando su longitud
la variable size, recibida por parametro.
Queda a responsabilidad del usuario liberar la
memoria reservada, por medio de la funcion free.
*/
char* read_from_stdin(size_t* size);

// Parsea una cadena de matrices, a un arreglo de
// matrices
/*
PRE : Recibe una cadena con matrices cuadradas
en el formato :
N a11 .... ann b11 .... bnn,
donde N es la dimension esta matrices, tal que,
NxN.
POST: Devuelve un arreglo de numero (double *),
con los elementos de la primer matriz, seguidos
de los elementos de la segunda.
Ademas almacena en matrix_size el la dimension
de la matriz.
*/
double* parse(
    char* string_matrixes,
    size_t* matrix_size
);

// Extrae un arreglo de elementos de una la matriz
// desde un arreglo de elementos para dos matrices
/*
PRE : Recibe:
    un arreglo con los elementos de dos matrices
    cuadradas, unos seguidos de los otros.
    la dimension (de lado) de estas matrices
    y un numero:
        0 : para referirse a la primer matriz
```

```
        1 : para referirse a la segunda matriz
POST: Devuelve un arreglo con los elementos de la
matriz seleccionada (0 o 1).
Queda a responsabilidad del usuario liberar la memoria
reservada por medio de la funcion free.
*/
double* extract_matrix(
    double* array_matrixes,
    size_t matrix_size,
    size_t matrix_num
);

// Loguea un mensaje por stderr y sale de la aplicacion.
/*
PRE: Recibe un codigo de error (int)
POST: Imprime por salida de error estandar, un mensaje
descriptivo para el error en cuestion, y termina la
ejecucion del programa.
*/
void logAndExit(int error_code);
```

Haciendo uso de las funciones descriptas en este archivo y en matrix.c/h, se implementa el programa, mediante las funciones :

```
//Procesa lineas de mutliplicacion de matrices
/*
PRE: Recibe una linea correspondiente a dos matrices
a multiplicar.
POST: Procesa la linea, realizando la multiplicacion,
e imprimiendo el resultado por entrada estandard.
*/
void process_line(char* line);

// Ejecuta el programa de desde el parseo de la
// entrada estandar, la multiplicacion de  matrices
// cuadradas y la impresion del resultado.
// (Para todas las lineas de multiplicacion de
// matrices recibidas por entrada estandard)
void exec_program();
```

Por ultimo se listan las funciones utlizadas para para ejecutar el programa desde sus argumentos, y brindar informacion al usuario :

```
// Imprime por salida estandard, la distintas
// formas de ejecutar el programa
void show_help();
```

```
// Imprime la version del programa
void version()

// Procesa los argumentos del programa para
// mostrar ayuda, version o ejecutar el programa
// dependiendo del caso.
int main( int argc, const char* argv[] );
```

1.1.3. testing.c y testing.h

Estos archivos contienen funciones utilizadas para realizar pruebas de la interfaz de la matriz. Estas pruebas ayudan a entender el funcionamiento esta interfaz.

A continuacion se listan algunas funciones auxiliares utilizadas para testear la interfaz :

```
// Imprime una prueba junto a su resultado
/*
PRE: Recibe las descripcion de un test, y un valor
booleano que indique el resultado del test.
POST: Imprime la descripcion del test, seguido del
resultado tal que si:
assertion == True entonces se imprime 'OK'
assertion == False entonces se imprime 'ERROR'
*/
void print_test(
    const char* test_description,
    bool assertion
);

// Imprime una prueba con una advertencia de verificar
// la memoria utilizada con valgrind
/*
PRE : Recibe una descripcion de la prueba.
POST: Imprime la prueba por stderr, junto a indicacion
de cheaquear valgrind.
*/
void print_test_valgrind(const char * test_description);
```

A continuacion se listan las pruebas de interfaz realizadas :

```
/*
Al construir una matriz, el array interno inicializa nulo
*/
void test_create_matrix_internal_array_is_null();

/*
```

```
Imprimir la matriz, imprime la cantidad de filas seguido
de un listado de los valores de la matriz (por fila)
*/
void test_print_matrix_print_rows_count_and_list_of
    _values();

/*
Al asignar un array reservado en memoria dinamica al
atributo array de la matriz, luego al destruir la matriz,
tambien se libera la memoria reservada
*/
void test_asign_dynamic_array_to_matrix_and_destroy
    _also_free_array();

/*
Crea 2 matrices con sus respectivos arreglos dinamicos,
realiza la multiplcacion entre ambas matrices y compara
el resultado. Finalmente destruye las matrices y libera
la memoria utilizada
*/
void test_multiply_two_matrixs();

/*
Crea 2 matrices con sus respectivos arreglos dinamicos,
de dimnesiones distintos (como requisito deben
tener la misma dimension) realiza la multiplcacion entre
ambas matrices y retorna NULL por la pre condicion.
Finalmente destruye las matrices y libera la memoria
utilizada.
*/
void test_multiply_two_matrix_different_sizes();

/*
Crea una matriz con el arreglo lleno de cero y se crea
otra matriz con su arreglo con numeros realiza la
multiplcacion entre ambas matrices y compara si el
resultado el array esta completo con cero. Finalmente
destruye las matrices y libera la memoria utilizada.
*/
void test_multiply_with_zero_matrix();

// Ejecuta todas las pruebas
void run_all_tests();
```

1.1.4. mainTest.c

Este archivo sirve para generar un ejecutable donde se incluyan las pruebas de interfaz realizadas en testing.c y testing.h

1.1.5. testFiles/

En este directorio se encuentran los archivos de texto utilizado para realizar pruebas del programa completo.

1.1.6. Codigo_mips/

En este directorio se encuentran los fuentes main.c y matrix.c ensamblados en mips, resultando en dos archivos assembler :

- main.s
- matrix.s

2. Comandos para compilar

Se provee un archivo Makefile para facilitar la compilacion.

2.1. Caracteristicas del Makefile

A continuacion se dictan las distintas formas de uso :

2.1.1. Compilar programa

Compilar programa principal de multiplicacion de matrices cuadradas.

```
make tp0
```

2.1.2. Compilar pruebas de la interfaz

Compilar pruebas de la interfaz.

```
make test-sources
```

2.1.3. Compilar todo

Compilar tanto el programa como las pruebas de la interfaz generando dos ejectuables distintos.

```
make all
```

2.1.4. Limpiar ejecutables

Remueve los archivos ejecutables y .o

```
make clean
```

2.1.5. Ejecutar pruebas de interfaz

Ejecuta las pruebas de interfaz.

```
make tests
```

2.2. Ejecución del programa

El programa se puede ejecutar como :

```
./tp0
```

Tambien se puede proveer de algun archivo de texto con varias lineas de matrices al multilplicar, e introducirlo en el programa por la entrada estandar de la forma:

```
cat <algun test>.txt | ./tp0
```

3. Pruebas realizadas

A continuacion se muestran las corridas de varias pruebas realizadas :

3.1. Pruebas del enunciado

3.1.1. Pruebas argumentos

```
$ ./tp0 -h
        Usage:
                ./tp0 -h
                ./tp0 -V
                ./tp0 < in_file > out_file
        tp0 [options]
        Options:
                -V, --version   Print version and quit.
                -h, --help      Print this information.
        Examples:
                ./tp0 < in.txt > out.txt
                cat in.txt | ./tp0 > out.txt
```

3.1.2. Prueba de ejemplo

```
$ cat example.txt
2 1 2 3 4 1 2 3 4
3 1 2 3 4 5 6.1 3 2 1 1 0 0 0 1 0 0 0 1

$ cat example.txt | ./tp0
2 7 10 15 22
3 1 2 3 4 5 6.1 3 2 1
```

3.2. Nuestras Pruebas

3.2.1. Test 1

En este test multiplicamos dos matrices de dimension 2 identicas, repletas de numeros 1.

```
cat test_1.txt
2 1 1 1 1 1 1 1 1

$ cat test_1.txt | ./tp0
$ 2 2 2 2 2
```

3.2.2. Test 1 bis

En este test repetimos el Test 1, pero agregando distintos espacios entre elemento y elemento y colocando valores decimales en cero.

```
cat test_1_bis.txt
2    1.00    1.0    1.000 1.0   1.000    1.0000    1.0      1.000

$ cat test_1_bis.txt | ./tp0
$ 2 2 2 2 2
```

3.2.3. Test 2

En este test multiplicamos una matrices de dimensión 2. La primera tiene el primero y cuarto elemento muy grandes, y los restantes muy chicos, de forma que se asemejen al cero en la notacion usada. La otra es una matriz donde su primer columnas son numeros 2, y la segunda numeros 1.

```
cat testFiles/test_2.txt
2 1.11e+200 0.1e-200 1.11e+200 0.1e-200 2 1 2 1

$ cat testFiles/test_2.txt | ./tp0
$ 2 2.22e+200 1.11e+200 2.22e+200 1.11e+200
```

3.2.4. Test 3

En el test 3 intentamos multiplicar dos matrices que se asemejen a la identidad, usando valor muy pequeños para reemplazar al cero.

```
cat test_3.txt
3 1 0.1e-300 0.1e-300 0.1e-300 1 0.1e-300 0.1e-300 0.1e-300 1
1 0.1e-300 0.1e-300 0.1e-300 1 0.1e-300 0.1e-300 0.1e-300 1
(Es una sola linea)

$ cat testFiles/test_3.txt | ../tp0
$ 3 1 2e-301 2e-301 2e-301 1 2e-301 2e-301 2e-301 1
```

3.2.5. Test 4

En este test multiplicamos a matrices de dimensión 2, donde la primera esta llena de numeros 1, mientras que la segunda, toma valores muy cercanos a cero.

```
cat testFiles/test_4.txt
2 1 1 1 1 0.1e-300 0.1e-300 0.1e-300 0.1e-300

$ cat test_4.txt | ../tp0
$ 2 2e-301 2e-301 2e-301 2e-301
```

3.2.6. Test 5

En este test creamos y multiplicamos 2 matrices de dimensión 3, de forma tal que la multiplicación no de por resultado el mismo valor en todos los elementos de la matriz.

```
cat testFiles/test3x3Ok.txt
3 1 2 3 1 2 3 1 2 3 2 3 4 5 6 7 6 5 4

$ cat testFiles/test3x3Ok.txt | ../tp0
$ 3 30 30 30 30 30 30 30 30 30
```

3.2.7. Test 6

En este test intentamos la multiplicamos 2 matrices de dimensión 3, pero esta nos debe dar error, debido que en una de estas no se encuentra la cantidad de elementos necesarios.

```
cat testFiles/test3x3Wrong.txt
3 1 2 3 1 2 3 1 2 3 2 3 4 5 6 7 6 5

$ cat testFiles/test3x3Wrong.txt | ../tp0
$ Tamano invalido para la matriz. Solo se permiten
matrices cuadradas.
```

3.2.8. Test 7

En este test se realiza la multiplicamos 2 matrices de dimensión 3, donde una de estas esta compuenpuesta de numeros negativos.

```
cat testFiles/testBigSciNotationOK
2 -1.02857e+200 -1.02857e+200 -1.02857e+200 1 1 2 3 4

$ cat testFiles/testBigSciNotationOK | ../tp0
$ 2 -4.11428e+200 -6.17142e+200 -1.02857e+200 -2.05714e+200
```

3.2.9. Test 8

En este test es muy similar a la anterior, solo que en esta ocasión no se obtiene el tamaño de las matrices, entonces debe retornar un error.

```
cat testFiles/testSciFiNotationWrong
-1.02857e+2 -1.02857e+2 -1.02857e+2 -1.02857e+2 1 1 2 3 4

$ cat testFiles/testSciFiNotationWrong | ../tp0
$ Tamano invalido para la matriz. Solo se permiten
matrices cuadradas.
```