



6620 - ORGANIZACIÓN DEL COMPUTADOR
Facultad de Ingeniería de la universidad de
Buenos Aires

Trabajo Práctico 1 Multiplicador de matrices cuadradas

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

1. Documentacion

1.1. Archivos

1.1.1. C

- c_matrix.c
- matrix.h
- main.c
- print_matrix.c
- mymalloc_2.c
- mymalloc.h

1.1.2. MIPS Assembly

- matrix_multiply.S
- mips_create_matrix.S
- mips_destroy_matrix.S
- mips_matrix_get_col.S
- mips_matrix_get_row.S
- mymalloc.S

1.1.3. mymalloc.S

Subrutina realizada por la cátedra para realizar un malloc desde assembly de MIPS.

1.1.4. mips_create_matrix.S

En este archivo se encuentra la creación de una matriz nueva, tomando como parámetros la cantidad de filas (esperado en el registro a0) y la cantidad de columnas (esperado en el registro a1).

TAMAÑO STACK FRAME
1 WORD (32 BITS)

44		ABA	CALLER
40			
36	a1		
32	a0		
28	-	SRA	RUTINA
24	RA		
20	\$FP		
16	GP		
12		ABA	
8			
4			
0			

```
#include <mips/regdef.h>
#include <sys/syscall.h>
#define TAMSTACKFRAME 32
#define OFFSET_GP 16
#define OFFSET_FP 20
#define OFFSET_RA 24

.text    # texto a continuacion
.abicalls    # Vamos a respetar las convenciones
.align 2     # Alinea (solamente) la siguiente
              # instruccion a 2 bytes

# PRE: Recibe la cantidad de filas y columnas de
# la matriz a traves de a0 y a1.
# POST: Devuelve un puntero a un nueva matriz de
# dichas dimensiones, o 0 en caso de error.

# Declaro y defino mi funcion
.globl create_matrix
.ent create_matrix
create_matrix:
# Inicio Receta

.frame $fp, TAMSTACKFRAME, ra
.set noreorder
.cpload t9
.set reorder
subu sp, sp, TAMSTACKFRAME
.cprestore OFFSET_GP
sw $fp, OFFSET_FP(sp)
sw ra, OFFSET_RA(sp)
move $fp, sp
sw a0, TAMSTACKFRAME($fp)    # rows
sw a1, (TAMSTACKFRAME + 4)($fp) # cols

# Fin receta
```

```

# Inicio codigo de la funcion

# Creamos matriz nueva
# Reservamos memoria para la estructura matrix_t
li a0, 12          # 4 (rows, size_t)
                   # + 4 (cols, size_t)
                   # + 4 (array, ptr)

la t9, mymalloc
jal ra, t9
move a2, v0        # a2 = v0 = new_matrix (ptr)
ble a2, 0, create_matrix_error
sw a2, (TAMSTACKFRAME + 8)($fp) # ptr new_matrix

# Recuperamos registros que pudieran perderse
lw a0, (TAMSTACKFRAME)($fp)
lw a1, (TAMSTACKFRAME + 4)($fp)

# Inicializamos new_matrix
sw a0, 0(a2)      # new_matrix->rows = a0 = rows
sw a1, 4(a2)      # new_matrix->cols = a1 = cols
li t0, 0          # t0 = 0
sw t0, 8(a2)      # new_matrix->array = t0 = 0
                   # (por defecto)

# Retornamos matrix creada
move v0, a2        # v0 = a2 = new_matrix (ptr)
                   # (guardo valor en registro
                   # retorno)

b end_create_matrix

create_matrix_error:
    li v0, 0        # Codigo error
    b end_create_matrix

# Fin codigo de la funcion

# Finalizo la rutina
end_create_matrix:
    # Restauro registros ABI
    lw gp, OFFSET_GP(sp)
    lw $fp, OFFSET_FP(sp)
    lw ra, OFFSET_RA(sp)

    # Destruyo Stack Frame
    addu sp, sp, TAMSTACKFRAME

    # Devuelvo control al SO,
    # o a la Caller
    jr ra

```

```
.end create_matrix
```

1.1.5. mips_destroy_matrix.S

En este archivo se encuentra la destruccion de una matriz creada por `create_matrix` y que ademas se le asigno un arreglo de valores almacenado en memoria dinamica. Su unico parametro es el puntero a la estructura `matrix_t` que recibe por el registro `a0`.

TAMAÑO STACK FRAME

1 WORD (32 BITS)			
44		ABA	CALLER
40			
36			
32	a0		
28	-	SRA	RUTINA
24	RA		
20	\$FP		
16	GP		
12		ABA	
8			
4			
0			

```
#include <mips/regdef.h>
#include <sys/syscall.h>
#define TAMSTACKFRAME 32
#define OFFSET_GP 16
#define OFFSET_FP 20
#define OFFSET_RA 24

.text    # texto a continuacion
.abicalls    # Vamos a respetar las convenciones
.align 2     # Alinea (solamente) la siguiente
              # instruccion a 2 bytes

# PRE: Recibe un ptr a matrix_t creado por
# create_matrix, cuyo arreglo de elementos
# puede estar almacenado en el heap a traves
# de la funcion mymalloc, o ser nulo (0)
# POST: Destruye la matrix y libera el su
# arreglo si es que existe

# Declaro y defino mi funcion
.globl destroy_matrix
.ent destroy_matrix
destroy_matrix:
# Inicio Receta

.frame $fp, TAMSTACKFRAME, ra
```

```

        .set noreorder
        .cpload t9
        .set reorder
        subu sp, sp, TAMSTACKFRAME
        .cpstore OFFSET_GP
        sw $fp, OFFSET_FP(sp)
        sw ra, OFFSET_RA(sp)
        move $fp, sp
        sw a0, TAMSTACKFRAME($fp)      # ptr matrix_t

# Fin receta

# Inicio codigo de la funcion
# Libero matrix->array
lw a0, 8(a0)      # a0 = matrix->array
ble a0, 0, continuar_destruir    # a0 <= 0 ?
la t9, myfree
jal ra, t9
continuar_destruir:
# Recupero registros perdidos
lw a0, TAMSTACKFRAME($fp)      # a0 = ptr matrix

# Libero matrix
la t9, myfree
jal ra, t9

# Fin codigo de la funcion

# Finalizo la rutina

# Restauro registros ABI
lw gp, OFFSET_GP(sp)
lw $fp, OFFSET_FP(sp)
lw ra, OFFSET_RA(sp)

# Destruyo Stack Frame
addu sp, sp, TAMSTACKFRAME

# Devuelvo control al SO,
# o a la Caller
jr ra

.end destroy_matrix

```

Cabe decir, que no son necesarias las rutinas `create_matrix` y `destroy_matrix`, implementadas en MIPS, pues se puede utilizar las mismas, ya creadas en C, enmascarando los `malloc` y `free` en `mymalloc` y `myfree`, respectivamente, que pueden tener uno u otra implementacion segun si se desea utilizar el codigo provisto por la catedra, las funciones de C. Esto ultimo no se tuvo en cuenta en un principio, por lo que tras codearlas se decidio dejarlas en el proyecto.

1.1.6. `matrix_multiply.S`

Este fuente cuenta con el código en MIPS de la mutliplicación de matrices. Utiliza las subrutinas *mips_create_matrix.S* para crear la matriz resultado y el par *mips_matrix_get_row.S* y *mips_matrix_get_col.S* para obtener una cierta fila o columna de la matriz de tipo `matrix_t`. Realiza la multiplicación de las dos matrices pasadas por parámetro mientras aloja los resultados en la estructura creada con *mips_create_matrix.S* y finalmente devuelve un puntero a ésta.

Stack frame:

Tamaño Frame	1 word (32 bits)	
60	a3	CALLER
56	a2	
52	a1	
48	a0	
44	s1	SRA
40	ra	
36	\$fp	
32	gp	
28	t4	LTA
24	t3	
20	t1	
16	t0	
12		ABA
8		
4		
0		

Codigo en assembler de MIPS32:

```
#include <mips/regdef.h>
#include <sys/syscall.h>
#define TAMSTACKFRAME 48
#define OFFSET_GP 32
#define OFFSET_FP 36
#define OFFSET_RA 40
#define OFFSET_S1 44
#define OFFSET_T0 16
#define OFFSET_T1 20
#define OFFSET_T3 24
#define OFFSET_T4 28

.text          # texto a continuacion
.abicalls      # Vamos a respetar las convenciones
.align 2       # Alinea (solamente) la siguiente
               # instruccion a 2 bytes

# PRE: Recibe dos matrices a traves de los
# registros a0 y a1.
# POST: Devuelve un nueva matriz, que es el
# producto de la multiplicacion de las dos
# anteriores
# Queda a responsabilidad del usuario destruir
# la matriz mediante la funcion destroy_matrix

# Declaro y defino mi funcion
.globl matrix_multiply
```



```

        .ent matrix_multiply
matrix_multiply:
    # Inicio Receta

    .frame $fp, TAMSTACKFRAME, ra
    .set noreorder
    .cpload t9
    .set reorder
    subu sp, sp, TAMSTACKFRAME
    .cpstore OFFSET_GP
    sw $fp, OFFSET_FP(sp)
    sw ra, OFFSET_RA(sp)
    move $fp, sp
    sw a0, TAMSTACKFRAME($fp)      # ptr matrix_1
    sw a1, (TAMSTACKFRAME + 4)($fp) # ptr matrix_2
    sw s1, OFFSET_S1($fp) # Guardo este registro
                                # por convenci n

# Fin receta

# Inicio codigo de la funcion

# Creamos matriz nueva
move t0, a0      # t0 = a0 = matrix_1
lw a0, (t0)      # a0 = matrix_1->rows
lw a1, 4(t0)     # a1 = matrix_1->cols
la t9, create_matrix
jal ra, t9
move a2, v0      # a2 = v0 = new_matrix (ptr)
beq a2, 0, matrix_multiply_error      # new_matrix == NULL
sw a2, (TAMSTACKFRAME + 8)($fp)      # ptr new_matrix
# (Guardo pues utilizare mas adelante)

# Recuperamos registros que pudieran perderse
lw a0, (TAMSTACKFRAME)($fp)      # a0 = matrix_1
lw a1, (TAMSTACKFRAME + 4)($fp)  # a1 = matrix_2

# Reservamos memoria para new_matrix->array
lw t0, (a2)      # t0 = new_matrix->rows
mul t2, t0, t0  # t2 = t0 * t0
                                # = new_matrix->rows
                                # * new_matrix->rows
                                # = cantidad posiciones
                                # array
sll t2, t2, 3    # Multiplico por 8 = 2^3,
                                # pues son doubles
                                # => cantidad bytes en array

move a0, t2
la t9, mymalloc
jal ra, t9

```

```

move a3, v0          # a3 = v0 = ptr array
beq a3, -1, array_malloc_error
sw a3, (TAMSTACKFRAME + 12)($fp)          # ptr array

# Recuperamos registros que pudieran perderse
lw a0, (TAMSTACKFRAME)($fp)
lw a1, (TAMSTACKFRAME + 4)($fp)
lw a2, (TAMSTACKFRAME + 8)($fp)
move v0, a2          # v0 = a2 = ptr new_matrix
                        # (guardo valor retorno)

# Asignamos array a new_matrix
sw a3, 8(a2)          # new_matrix->array = a3 = ptr array

lw t0, (a0)           # t0 = matrix_1->rows

#Construyo iteradores
and t3, t3, zero
and s1, s1, zero      # iterador de posici n
                        # dentro de la matriz.

i_loop:
    beq t0, t3, end_matrix_multiply # t3 = i

    #Guardo el estado de los ts en el stack
    sw t0, OFFSET_T0($fp)
    sw t3, OFFSET_T3($fp)

    move a0, a0
    move a1, t3
    la t9, matrix_get_row    # get_row(m1, i)
    jal t9

    # Recuperamos registros que pudieran perderse
    lw a0, (TAMSTACKFRAME)($fp)    # a0 = ptr matrix_1
    lw a1, (TAMSTACKFRAME + 4)($fp) # a1 = ptr matrix_2
    lw a2, (TAMSTACKFRAME + 8)($fp) # a2 = ptr new_matrix
    lw a3, (TAMSTACKFRAME + 12)($fp) # a3 = new_matrix->array
    lw t0, OFFSET_T0($fp)    # new_matrix->rows
    lw t3, OFFSET_T3($fp)    # nro fila actual

    move t1, v0              # Salvo el resultado de get_row
                            # t1 = fila actual (array de doubles)

    and t4, t4, zero          # Iterador t4

j_loop:
    beq t0, t4, end_j_loop    # t4 = j

```

```

sw t0, OFFSET_T0($fp)
sw t1, OFFSET_T1($fp)
sw t3, OFFSET_T3($fp)
sw t4, OFFSET_T4($fp)

#preparo argumentos y llamo a funcion
move a0, a1          # a0 = a1 = ptr matrix_2
move a1, t4          # a1 = nro columna actual
la t9, matrix_get_col
jal t9

# Recuperamos registros que pudieran perderse
lw a0, (TAMSTACKFRAME)($fp)      # a0 = ptr matrix_1
lw a1, (TAMSTACKFRAME + 4)($fp)   # a1 = ptr matrix_2
lw a2, (TAMSTACKFRAME + 8)($fp)   # a2 = ptr new_matrix
lw a3, (TAMSTACKFRAME + 12)($fp)  # a3 = new_matrix->array
lw t0, OFFSET_T0($fp)  # new_matrix->rows
lw t1, OFFSET_T1($fp)  # fila actual (array de doubles)
lw t3, OFFSET_T3($fp)  # t3 = i (indice i)
lw t4, OFFSET_T4($fp)  # t4 = j (indice j)

                move t2, v0          # Salvo el resultado de get_col
                                   # t2 = columna actual
                                   # (array de doubles)

and t5, t5, zero          # iterador t5

k_loop:
    beq t0, t5, end_k_loop # t5 = k

    # Obtengo actual_row[k]
    sll t6, t5, 3
    addu t6, t6, t1
    l.d $f0, (t6)

    # Obtengo actual_col[k]
    sll t6, t5, 3
    addu t6, t6, t2
    l.d $f2, (t6)

    # Sumo y acumulo
    mul.d $f6, $f0, $f2
    add.d $f4, $f4, $f6          # Acumulo en f4

    addiu t5, t5, 1             # Avanzo t5
    b k_loop

end_k_loop:

    #Coloco el numero calculado en su posicion final en la matriz a3

```

```

sll t7, s1, 3      # Escalo el iterador i a tamaño de double
move t8, a3        # t8 = a3 = ptr new matrix
addu t7, t7, t8    # Avanzo t7 bytes de t8 y lo guardo en t7
s.d $f4, (t7)      # Store del resultado en new matrix
li.d $f4, 0        # Reset del acumulador
addiu s1, s1, 1
addiu t4, t4, 1

# Guarda registros que quiero mantener
sw t0, OFFSET_T0($fp)
sw t1, OFFSET_T1($fp)
sw t3, OFFSET_T3($fp)
sw t4, OFFSET_T4($fp)

# Libero arreglo de elementos en columna actual
move a0, t2
la t9, myfree
jal ra, t9

# Recuperamos registros que pudieran perderse
lw a0, (TAMSTACKFRAME)($fp)      # a0 = ptr matrix_1
lw a1, (TAMSTACKFRAME + 4)($fp)  # a1 = ptr matrix_2
lw a2, (TAMSTACKFRAME + 8)($fp)  # a2 = ptr new_matrix
lw a3, (TAMSTACKFRAME + 12)($fp) # a3 = new_matrix->array
lw t0, OFFSET_T0($fp)            # new_matrix->rows
lw t1, OFFSET_T1($fp)            # fila actual (array de doubles)
lw t3, OFFSET_T3($fp)            # t3 = i (índice i)
lw t4, OFFSET_T4($fp)            # t4 = j (índice j)

b j_loop

end_j_loop:
addiu t3, t3, 1                  # Avanzo t3 una posición

# Guardo registros que quiero mantener
sw t0, OFFSET_T0($fp)
sw t3, OFFSET_T3($fp)

# Libero arreglo de elementos en fila actual
move a0, t1
la t9, myfree
jal ra, t9

# Recuperamos registros que pudieran perderse
lw a0, (TAMSTACKFRAME)($fp)      # a0 = ptr matrix_1
lw a1, (TAMSTACKFRAME + 4)($fp)  # a1 = ptr matrix_2
lw a2, (TAMSTACKFRAME + 8)($fp)  # a2 = ptr new_matrix
lw a3, (TAMSTACKFRAME + 12)($fp) # a3 = new_matrix->array
lw t0, OFFSET_T0($fp)            # new_matrix->rows
lw t3, OFFSET_T3($fp)            # nro fila actual

```

```

        b i_loop

array_malloc_error:
    lw a0, (TAMSTACKFRAME + 8)($fp) # a0 = ptr new_matrix
    la t9, destroy_matrix    # destroy_matrix ya considera
                                # el caso de matrix_t sin array

    jal ra, t9
    b matrix_multiply_error

matrix_multiply_error:
    li v0, 0                  # Codigo error
    b end_matrix_multiply

# Fin codigo de la funcion

# Finalizo la rutina
end_matrix_multiply:
    # Coloco el resultado en v0
    move v0, a2

    # Restauro registros ABI
    lw gp, OFFSET_GP(sp)
    lw $fp, OFFSET_FP(sp)
    lw ra, OFFSET_RA(sp)
    lw s1, OFFSET_S1(sp)

    # Destruyo Stack Frame
    addu sp, sp, TAMSTACKFRAME

    # Devuelvo control al SO,
    # o a la Caller
    jr ra

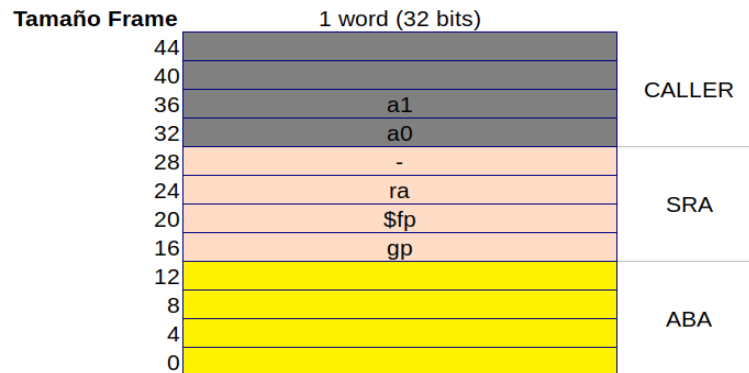
    .end matrix_multiply

```

1.1.7. mips_matrix_get_col.S

En este archivo se encuentra la función creada para obtener la columna del numero especificado para la multiplicación de matrices.

Stack frame:



```

#include <mips/regdef.h>
#include <sys/syscall.h>
#define TAMSTACKFRAME 32
#define OFFSET_GP 16
#define OFFSET_FP 20
#define OFFSET_RA 24
.text
.abicalls
.align 2

# PRE: Recibe un puntero a matrix_t y
# un numero de columna valido en la matriz
# POST: Devuelve un arreglo de doubles con
# elementos de la columna seleccionada
# almacenados en el heap
# Queda a responsabilidad del usuario, liberar
# dicha memoria mediante la rutina myfree

.globl matrix_get_col
.ent matrix_get_col
matrix_get_col:
.frame $fp, TAMSTACKFRAME, ra
.set noreorder
.cpload t9
.set reorder
subu sp,sp,TAMSTACKFRAME
.cprestore OFFSET_GP
sw $fp,OFFSET_FP(sp)
sw ra,OFFSET_RA(sp)
move $fp,sp

sw a0,TAMSTACKFRAME($fp) # a0 = ptr matriz
sw a1,(TAMSTACKFRAME + 4)($fp) # a1 = nro columna

lw a0, 4(a0) # a0 = matriz->cols
sll a0, a0, 3 # a0 = matriz->cols * sizeof(double)

```

```

    la t9, mymalloc
    jal ra, t9
    beq v0, -1, array_malloc_error
    sw v0, (TAMSTACKFRAME + 8)($fp)

    li t0, 0
    lw t1, TAMSTACKFRAME($fp)      # t1 = ptr matriz
    lw t2, 4(t1)                   # t2 = matriz->cols = cant de cols
    lw t3, TAMSTACKFRAME + 4($fp) # t3 = nro columna
    lw t4, 8(t1)                   # t4 = matriz->array
    move a2, v0                    # a2 = array_nro_col

loop:
    bgt t0, t2, exit
    sll t5, t3, 3                  # t3 = nro_columna * sizeof(double)
    addu t6, t4, t5                # * matriz[pos] = ptr matrix[pos]
    l.d $f0, (t6)                 # matriz[pos]
    sll t7, t0, 3                  # Aumento 8 bites al index del array
    addu t8, a2, t7
    s.d $f0, (t8)
    addu t3, t3, t2
    addu t0, t0, 1
    b loop

array_malloc_error:
    li v0, 0
    b exit

exit:
    lw gp, OFFSET_GP(sp)
    lw $fp, OFFSET_FP(sp)
    lw ra, OFFSET_RA(sp)
    addu sp, sp, TAMSTACKFRAME
    jr ra

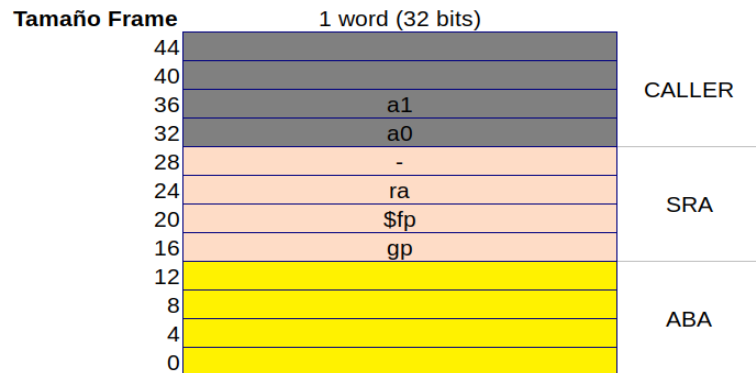
.end matrix_get_col

```

1.1.8. mips_matrix_get_row.S

En este archivo se encuentra la función creada para obtener la fila del número especificado para la multiplicación de matrices.

Stack frame:



```

#include <mips/regdef.h>
#include <sys/syscall.h>
#define TAMSTACKFRAME 32
#define OFFSET_GP 16
#define OFFSET_FP 20
#define OFFSET_RA 24
.text
.abicalls
.align 2

# PRE: Recibe un puntero a matrix_t y
# un numero de fila valido en la matriz
# POST: Devuelve un arreglo de doubles con
# elementos de la fila seleccionada
# almacenados en el heap
# Queda a responsabilidad del usuario, liberar
# dicha memoria mediante la rutina myfree

.global matrix_get_row
.ent matrix_get_row
matrix_get_row:
.frame $fp,TAMSTACKFRAME,ra
.set noreorder
.cpload t9
.set reorder
subu sp,sp,TAMSTACKFRAME
.cprestore OFFSET_GP
sw $fp,OFFSET_FP(sp)
sw ra,OFFSET_RA(sp)
move $fp,sp

sw a0,TAMSTACKFRAME($fp)      # Guardo el primer      # parametro (la matriz)
sw a1,(TAMSTACKFRAME + 4)($fp)  # Guardo el segundo
                                # parametro
                                # de fila

```



```

        lw a0,(a0)
        sll a0,a0,3
        la t9,mymalloc
        jal ra,t9
        beq v0, -1, array_malloc_error
        sw v0,(TAMSTACKFRAME + 8)($fp)

        li t0,0
        lw t1,TAMSTACKFRAME($fp) # t1 = struct matriz
        lw t2,(t1) # t2 = cantidad de filas
        lw t3,TAMSTACKFRAME + 4($fp) # Obtengo numero de fila
        lw t6,8(t1) #t6 = array matrix
        mul t4, t3, t2 # m->rows * row_n
        move a2,v0

loop:
        bgt t0,t2,exit
        addu t5,t4,t0 # (m->rows * row_n) + i
        sll t5,t5,3
        addu t7,t6,t5 # * matriz[(m->rows * row_n) + i]
        l.d $f0,(t7) # matriz[(m->rows * row_n) + i]
        sll t8,t0,3 # Aumento 8 bites al index del array
        addu t5,a2,t8
        s.d $f0,(t5)
        addu t0,t0,1
        b loop

array_malloc_error:
        li v0,0
        b exit

exit:
        lw gp, OFFSET_GP(sp)
        lw $fp, OFFSET_FP(sp)
        lw ra, OFFSET_RA(sp)
        addu sp, sp, TAMSTACKFRAME
        jr ra

.end matrix_get_row

```

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 - MIPS con malloc de la catedra

Compilar programa principal de multiplicacion de matrices cuadradas con código MIPS32, utilizando las funciones mymalloc y myfree provistas por la catedra.

```
make mips_malloc_catedra
```

En esta version, para de programar la funcion de multiplicacion de matrices se utilizó las rutinas de mymalloc y myfree.

Aqui se nos presento un error, que no logramos resolver, donde al querer liberar memoria que previamente reservamos, la rutina myfree nos levanta un Segmentation Fault. Estos ocurre cuando se desea multiplicar matrices de dimensiones mayor o igual a 2. Creemos que se debe a un problema con las rutina myfree del archivo mymalloc.S provista por la catedra.

2.1.2. Compilar programa - MIPS con malloc de C

Compilar programa principal de multiplicacion de matrices cuadradas con código MIPS32, utilizando las funciones mymalloc y myfree que enmascaran a las funciones de malloc y free de C.

```
make mips_malloc_c
```

En esta version, y para hacer del codigo reutilizable, es enmascaran con las funciones malloc y free de C, bajo las firmas mymalloc y myfree (identicas a las usadas en mymalloc.S). (Estas funciones "wrapper"^{es} encuentran en el archivo mymalloc_2.c).

Utilizando estas funciones de C para reservar memoria, y el mismo codigo utilizado en el ejecutable tp1_malloc_catedra, se obtiene un nuevo archivo ejecutable que no presenta los errores que en el anterior, y su ejecucion es exitosa en todos los casos.

2.1.3. Compilar programa - C

Compilar programa principal de multiplicacion de matrices cuadradas con código C (exclusivamente).

```
make  
o  
make c_common
```

Todas las formas de compilacion generan un archivo ejecutable:

```
tp1
```

2.1.4. Limpiar ejecutables

Remueve los archivos .o

```
make clean
```

2.2. Ejecución del programa

El programa se puede ejecutar como :

```
./tp1
```

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 | ./tp1
```

3. Pruebas realizadas

A continuacion se muestran las corridas de varias pruebas realizadas :

3.1. Pruebas del enunciado

3.1.1. Pruebas argumentos

```
$ ./tp1 -h
Usage:
    ./tp1 -h
    ./tp1 -V
    ./tp1 < in_file > out_file
tp1 [options]
Options:
    -V, --version    Print version and quit.
    -h, --help       Print this information.
Examples:
    ./tp1 < in.txt > out.txt
    cat in.txt | ./tp1 > 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 | ./tp1
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.

El resultado deberia ser un una matriz de dimension 2 llena de numeros 2.

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

```
$ cat testFiles/test_1.txt | ./tp1
$ 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.

El resultado, como en el caso anterior, deberia ser una matriz llena de numeros 2.

```
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 | ./tp1
$ 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.

Se espera la siguiente matriz :

```
2 2.22e+200 1.11e+200 2.22e+200 1.11e+200
```

(calculada mediante la version en C del programa)

```
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 | ./tp1
$ 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 de dimension 3, que se asemejen a la identidad, usando valores muy pequeños para reemplazar al cero. Esperamos que los valores de la matriz generada se encuentran cerca del cero, es decir, numeros en notacion cientifica con exponentes muy negativos.

```
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

$ cat testFiles/test_3.txt | ../tp1
$ 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.

Se espera la siguiente matriz :

```
2 2e-301 2e-301 2e-301 2e-301
```

(calculada mediante la version en C del programa)

```
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 | ../tp1
$ 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.

Se espera la siguiente matriz :

```
3 30 30 30 30 30 30 30 30 30
```

(calculada mediante la version en C del programa)

```
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 | ../tp1
$ 3 30 30 30 30 30 30 30 30 30
```

3.2.7. Test 6

En este test intentamos la multiplicamos de 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 | ../tp1
$ 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 compuesta de numeros negativos.

Se espera la siguiente matriz :

```
2 -4.11428e+200 -6.17142e+200 -1.02857e+200 -2.05714e+200
```

(calculada mediante la version en C del programa)

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

```
$ cat testFiles/testBigSciNotationOK | ../tp1
$ 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 ingresa 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 | ../tp1
$ Tamano invalido para la matriz. Solo se permiten
matrices cuadradas.
```

4. Código

Todo el resto del código no mostrado anteriormente, se presenta adjunto al informe.