Facultad de Ingeniería
Universidad de Buenos Aires
Organizacion de Computadoras (66.20)

Trabajo Práctico N°1

2^{do} Cuatrimestre, 2019

Boada, Ignacio Daniel	95212	ignacio.boada@outlook.com
Goñi, Mauro Ariel	87646	maurogoni@gmail.com
Perez Machado, Axel Mauro	101127	axelmpm@gmail.com

Índice

1. Comandos para compilación **2**

2. Diseño e implementación codigo C **3**

3. Diseño e implementación codigo MIPS **4**
3.1. Implementacion previa 4
3.2. Implementacion modificada 5
3.3. Stackframe (matrix multiply) 6
3.4. Stackframe original (matrix multiply) 7
3.5. Struct matrix 8
3.6. Implementacion de create matrix 9
3.7. Stackframe (create matrix) 9

4. Pruebas **10**

5. Conclusiones **11**

6. Codigo MIPS32 de matrix multply **12**

7. Codigo MIPS32 de create matrix **15**

8. Código fuente C **17**

9. Codigo MIPS32 del codigo C **24**

10.Enunciado **25**

1. Comandos para compilación

PARA DEBUGGEAR CON GDB:

```
gcc -Wall -o tp1 tp1.c matrix_multiply.S create_matrix.S mymalloc.S -ggdb
```

PARA COMPILAR EL PROGRAMA PARA CORRERLO:

```
gcc -Wall -o tp1 tp1.c matrix_multiply.S create_matrix.S mymalloc.S
```

PARA OBTENER CODIGO MIPS:

```
gcc -Wall -O0 -S -mrnames tp1.c
```

2. Diseño e implementación codigo C

Nuestra implementación no es nada fuera de lo común y busca ser lo mas simple y directa posible.

1. Primero se leen todos lo inputs en la primer linea de stdin y se almacenan en un array
2. Se crean las dos matrices (esto se hace llamando a la funcion implementada en codigo MIPS)
3. Se llenan sus elementos con los valores leidos
4. Se realiza el producto que luego se almacena en una tercera matriz que se crea (esto se hace llamando a la funcion implementada en codigo MIPS)
5. Se envia el resultado por stdout
6. Se libera memoria
7. Se vuelve a empezar todo el proceso con la siguiente linea de stdin

Consideraciones:

1. Si algun error de formato aparece, este se valida en el primer paso, se informa del problema por stderr y se corta el programa
2. Para facilitar la implementación no consideramos un error de formato una dimension positiva no nula de tipo float o double pero de mantiza nula .
3. Para facilitar la implementación no consideramos como un error de formato un archivo vacio. Ante tal situacion simplemente no se hace nada y se cierra el programa
4. Ante el ingreso de comandos -h,-V,–help,–version, estos se manejan abriendo un archivo de texto que se encuentra en el mismo directorio que el archivo compilado y el fuente. Esto permite cambiar el mensaje a mostrar sin tocar el codigo ni recompilar.

3. Diseño e implementación codigo MIPS

En este tp se tuvo como objetivo implementar la primitiva matrix multiply ya presente en el tp anterior.

3.1. Implementacion previa

```
matrix_t* matrix_multiply(matrix_t* matrix_a,matrix_t* matrix_b){

    int dimention = matrix_a->rows;

    matrix_t* matrix_c = create_matrix(dimention,dimention);

    int row;
    int column;
    int i;
    int j;
    double element;

    for (i = 0; i < dimention*dimention; i++){

        row = (int)(i / dimention);
        column = (int)(i % dimention);

        element = 0;
        for (j = 0; j < dimention; j++){

            element += matrix_a->array[row*dimention + j] * matrix_b->array[j*dimention +
        }
        matrix_c->array[i] = element;
    }
    return matrix_c;
}
```

Para que la implementacion fuese mas directa y mas facil de corregir para nosotros y para los docentes, optamos por modificar ligeramente la implementacion antes presentada por otra equivalente y mas similar a lo que pasa en codigo assembly

3.2. Implementacion modificada

```
matrix_t* matrix_multiply(matrix_t* matrix_a,matrix_t* matrix_b){
    int aux1;
    int aux2;
    int row;
    int column;
    int i;
    int j;
    int dimention;
    double element;
    dimention = matrix_a->rows;
    matrix_t* matrix_c;
    matrix_c = create_matrix(dimention,dimention);
    for (i = 0; i < dimention*dimention; i++){
        row = (int)(i / dimention);
        column = (int)(i %dimention);
        element = 0;
        for (j = 0; j < dimention; j++){
            aux1 = row*dimention;
            aux1 = aux1 + j;
            aux1 = matrix_a->array[aux1];
            aux2 = j*dimention;
            aux2 = aux2 + column;
            aux2 = matrix_b->array[aux2];
            aux1 = aux1 * aux2;
            element = element + aux1;}
        matrix_c->array[i] = element;}
    return matrix_c;}
```

Dicha implementacion fue debidamente testeada para asegurarse de que efectivamente fuese equivalente

Se agrega tambien un archivo a dicha implementacion en la entrega digital

Ademas tambien en el mismo espiritu de facilitar la tarea de correccion se entregan en forma digital dos versiones de codigo MIPS de matrix multiply. Una que tiene una abundante cantidad de comentarios, esa es la que esta en este informe, y otra sin ningun comentario. Hacemos esto porque notamos que lucia mas limpio y legible la version sin comentarios, pero no queriamos entregar una sin comentario alguno por si dificultaba el entendimiento de la implementacion.

Tambien, porque nos fue de enorme ayuda a nosotros, agregamos en este informe y en la entrega digital la implementacion del stackframe

3.3. Stackframe (matrix multiply)

```
    ================================================================ STACK FRAME de MAIN
60  ptr_matrix_b
56  ptr_matrix_a
    ================================================================ STACK FRAME de Multiply
    ============================== SRA
52  <padding>
48  ra
44  $fp
40  gp
    ============================== LTA
36  element(2da parte dado que es un double)
32  element
28  ptr_matrix_c
24  row (es el row calculado en cada ciclo de for, no el row de los structs)
20  column (es el column calculado en cada ciclo de for, no el column de los structs)
16  dimention
    ============================== ABA
12  <nada, se pone por convencion>
8   <nada, se pone por convencion>
4   dimention
0   dimention
```

Un detalle a indicar sobre este item es que el stackframe que nos hubiese gustado hacer en un principio era el siguiente:

3.4. Stackframe original (matrix multiply)

```
    ================================================================ STACK FRAME de MAIN
92  ptr_matrix_b
88  ptr_matrix_a
    ================================================================ STACK FRAME de Multiply
    ============================== SRA
84  <padding>
80  ra
76  fp
72  gp
    ============================== LTA
68  aux2(2da parte)
64  aux2
60  aux1(2da parte)
56  aux1
52  element(2da parte)
48  element
44  ptr_matrix_c
40  ptr_matrix_b
36  ptr_matrix_a
32  j
28  i
24  row
20  column
16  dimention
    ============================== ABA
12  <nada, se pone por convencion>
8   <nada, se pone por convencion>
4   dimention
0   dimention
```

Pero lo que veiamos mala idea era que en cada ciclo de loop necesitabamos constantemente cargar y descargar de memoria : aux1, aux2, i y j y nos parecio que eso afectaba mucho a la performance, por lo que decidimos desechar esa idea.

Otra cosa a destacar, y que nos dio bastantes dolores de cabeza, esta en como estan estructurados los datos en el struct matrix. Por mal entender esto en un principio tuvimos que estar varias horas debugeando.

3.5. Struct matrix

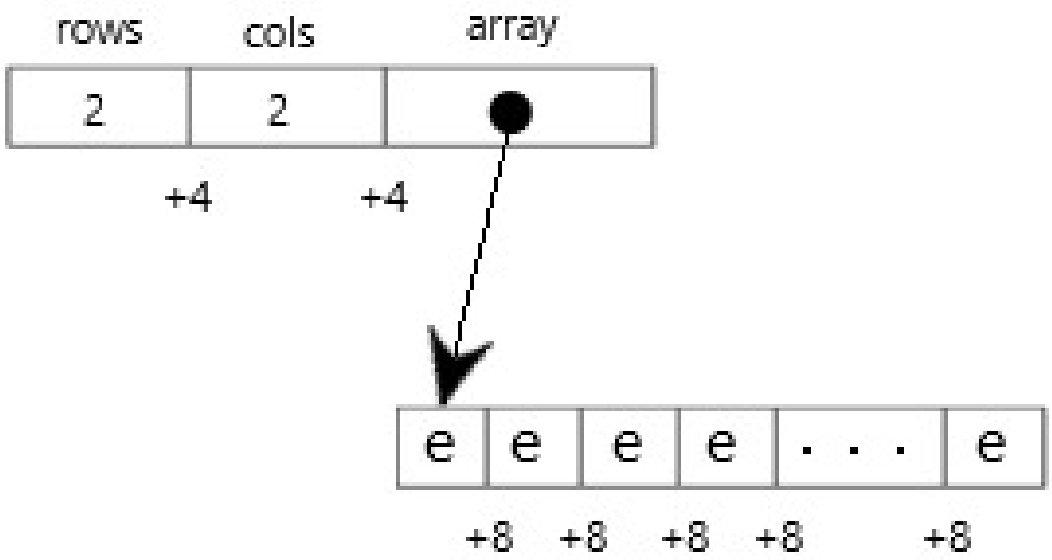

Figura 1: organizacion de los datos en el struct matrix

Este es un ejemplo de una matriz de 2x2.

Los +4 corresponden a saltos de 4 bytes por tratarse cols y rows de size_t.

Los +8 corresponden a saltos de 8 bytes por tratarse de doubles.

3.6. Implementacion de create matrix

La implementacion es una traduccion bastante directa de la version en C antes entregada.

En la implementacion de create matix en MIPS32 se hizo uso de las funciones provistas por la catedra mymalloc y myfree, siendo estas las analogas a malloc y free en C. Dado la implementacion de mymalloc fue necesario utilizar myfree para la liberacion de memoria allocateada por mymalloc de modo que tambien figura myfree en la implementacion de destroy matrix.

Tambien se tuvo que agregar una minima validacion del retorno de mymalloc en create matrix y otra en matrix multiply en caso de que hubiese habido un fallo.

Aca tambien se agrega el stackframe de la funcion create matrix.

3.7. Stackframe (create matrix)

```
    ================================================== STACK FRAME de matrix_multiply o
44  cols
40  rows
    ================================================== STACK FRAME de create_matrix
    ============================== SRA
36  <padding>
32  ra
28  $fp
24  gp
    ============================== LTA
20  ptr_array
16  ptr_matrix_c
    ============================== ABA
12  <nada, se pone por convencion>
8   <nada, se pone por convencion>
4   <nada, se pone por convencion>
0   (sizeof(double) * cols * rows) o sizeof(matrix_t)
```

4. Pruebas

Para una completa descripción de las pruebas corridas se tiene la carpeta entregada de "Pruebas"

Ademas tambien se entrega un archivo llamado çorridas de pruebas"que muestra los resultados de las corridas de pruebas.

En ella se encuentra una bateria de 26 pruebas que evaluan a grandes rasgos:

1. Errores de dimension
2. Errores de formato en elementos
3. Errores en alguna linea del archivo con otras en formato correcto
4. Producto entre identidad y una matriz cualquiera
5. Producto entre matriz nula y una matriz cualquiera
6. Producto entre una matriz inversible y su inversa
7. Varios productos consecutivos sin errores
8. Productos usando matrices grandes y chicas

Entre estas tenemos como ejemplos:

(Prueba 26)
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

Ejemplo provisto por la cátedra.

(Prueba 4)
2 4 8 9 7 1 0 0 1

Ejemplo de producto por identidad.

(Prueba 12)
2.32 8 4 6 5 8 7 7 8

Ejemplo de caso con dimensión no entera.

(Prueba 24)
2 4 5 7 2 -0.074074074 0.185185185 0.259259259 -0.148148148

Ejemplo bastante interesante donde se prueban elementos con mantiza y que ademas son negativos y donde ademas se testea el aspecto numérico del programa, es decir, la precisión.

5. Conclusiones

El trabajo práctico realizado fue introductorio para que comenzemos a estar relacionarnos con las herramientas que se utilizan en la cátedra.

Considerando la finalidad del trabajo práctico podemos concluir que fue de gran utilidad, ya que, en primer lugar tuvimos que aprender la sintaxis de C y comprender cómo este lenguaje maneja archivos y memoria. Otro punto importante fue que tuvimos que utilizar por primera vez el entorno NetBSD para correr el programa y obtener el código en MIPS, lo que será útil para el siguiente trabajo práctico ya que se utilizará el lenguaje que implementa MIPS.

A lo largo del tp nos encontramos con muchos errores que se debieron solucionar, propios de una mala utilización del lenguaje C, por lo que tuvimos que mejorar el uso de funciones y la modularización de las mismas. Con este trabajo tuvimos que considerar cada uno de los posibles errores que arrojan las librerías de C y pensar cómo debería reaccionar el programa ante cada uno de ellos. El hecho de tener que liberar memoria nos planteó problemas ya que teníamos que considerar mantener referencias siempre a las posiciones de memoria dinámica, y tenerlo en cuenta de liberarlas ante cada posible error. Una vez finalizada la programación y comenzadas las pruebas, nos dimos cuenta que habían muchas posibilidades de ingreso que no habíamos tenído en cuenta, por lo que debimos agregar validaciones y cortes del programa ante ingresos de datos inválidos.

A la bolsa de problemas y complicaciones en este ultimo tp realizado (TP1) debemos ahora sumar la extra capa de complejidad que agrega el programar en lenguaje assembly (tendiendo en cuenta que cambia fundamentalmente la forma en que se programa), la complejidad que agrega tener una idea de donde se esta parado en el stackframe, la diferenciacion entre registros en CPU y posiciones de memoria dentro del stackframe, el recordar respetar las convenciones ABI y el aprender a manejar la herramienta de debuggin GDB.

Una consideración que no es menor, es el trabajo fue en equipo. La comunicación en el equipo tampoco es sencilla ya que todos manejamos horarios diferentes y esto agrega una dificultad extra al problema de fondo, El aprendizaje que nos dejó este proceso es que es mejor organizarse bien desde un principio, y dividir tareas de manera bien modularizadas.

Finalmente podemos concluir que el trabajo nos dejó aprendizajes muy importantes para seguir con la materia, ya que aprender C es de suma utilidad, correr programas en NetBSD parece que se utilizará a lo largo de todo el curso, y sobretodo cómo trabajar en varias plataformas como Linux y NetBSD sin que hayan conflictos en la codificación permitiendo la portabilidad.

6. Codigo MIPS32 de matrix multply

```
#include <mips/regdef.h>

        .text
    .abicalls
        .ent matrix_multiply
        .globl matrix_multiply

matrix_multiply:

#Se crea stack frame de la callee

        .frame  $fp,56,ra
        subu    sp,sp,56

#Se crea SRA de la callee

    sw  ra,48(sp)
        .cprestore 40
        sw      $fp,44(sp)
        move    $fp,sp

#Se crea ABA de la caller

        sw      a0,56($fp)                  #Guarda puntero matrix_a en ABA de la caller (
        sw      a1,60($fp)                  #Guarda puntero matrix_b en ABA de la caller (

#(SE TRADUCE) dimention = matrix_a->rows:

#Se obtiene matriz_a->rows (dimention).

    lw  t0, 56($fp)             #se carga el puntero a la matrix_a (que es el mismo que ma
    lw  t1, 0(t0)               #se carga el valor apuntado por matriz_a->rows en t1 (dime
    sw  t1, 16($fp)                 #se guarda dimention en la LTA de la callee

#Se crea ABA de la callee

    sw  t1,0($fp)                   #dimention = matrix_a->rows; Copio t1 (que es matriz_a-
    sw  t1,4($fp)               #dimention = matrix_a->rows; Copio t1 (que es matriz_a->ro

#(SE TRADUCE) matrix_t* matrix_c;
#(SE TRADUCE) matrix_c = create_matrix(dimention,dimention);

    move a0, t1                 #se guarda dimention en los registros que se pasan por par
    move a1, t1
    jal create_matrix           #Se llama a la funcion create_matrix
    sw  v0, 28($fp)                 #Lo retornado por create_matrix se guarda en v0, y lo

    beqz v0,error_mem           #Si lo retornado por create_matrix es NULL (osea 0) se sal

#(SE TRADUCE) for (i = 0; i < dimention*dimention; i++);

#Para poder usar la misma l gica del programa en C, se aplica una funcionalidad para ir c

    li  t3, 0                   #i = 0
```

```
    lw  t1, 16($fp)             #En 16($fp) esta el valor de dimention. Lo cargo en t1.
    mul t2, t1, t1                  #t1 tiene la dimention, entonces queda dimention*dimen

primer_for:
        bge  t3, t2, return_matrix_multiply          #si "i" es mayor o igual a dimentio

#(SE TRADUCE) row = (int)(i / dimention);

        divu t0, t3, t1                 #t0 = (i / dimention).
        sw   t0, 24($fp)                #row esta en 24($fp), entonces queda row = (int)(i

#(SE TRADUCE) column = (int)(i % dimention);

        remu t4, t3, t1                 #t4 = (i % dimention).
        sw   t4, 20($fp)                    #column esta en 20($fp), entonces queda column


#(SE TRADUCE) element = 0;

    li  t0, 0
    sw  t0, 32($fp)                 #element es 32($fp) por lo que aca se hace element = 0
    sw  t0, 36($fp)                 #como element es un double tambien se debe hacer cero

#(SE TRADUCE) for (j = 0; j < dimention; j++):;

        li t5, 0                                #t5 sera el contrador "j" de la funcion C.

segundo_for:
        bge t5, t1, add_element     #si j>=dimention (t5 >= t1), va a add_elemt y termina

#(SE TRADUCE) aux1 = row*dimention;

        lw t6, 16($fp)              #t6 cargo dimention
        lw t7, 24($fp)              #t7 cargo row
        mul t8, t6, t7              #t8 tengo aux1 —— aux1=row*dimention

#(SE TRADUCE) aux1 = aux1 + j;

        add t8, t8, t5              #aux1= aux1+j

#(SE TRADUCE) aux1 = matrix_a->array[aux1];

        lw  t0, 56($fp)                 #me paro en matriz_a->rows
    addi t0,t0,8                #me paro en matriz_a->array
    lw  t0, 0(t0)               #cargo la direccion de comienzo del array de elementos de
        sll t8, t8, 3               #multiplico por 8 para que equivalga a la cantidad de
        add t0, t0, t8              #se desplaza el puntero al array de matriz_a->array
        l.d $f2, 0(t0)              #cargo el valor apuntado por matrix_a->array[aux1] en


#(SE TRADUCE) aux2 = j*dimention;

    mul t7, t5, t6              #Borro row y obtengo aux2 — aux2=j*dimention

#(SE TRADUCE) aux2 = aux2 + column;
```

```
        lw t4, 20($fp)              #En 20($fp) esta el valor de column. Lo cargo en t4. p
        add t7, t7, t4              #t7 tiene aux2 ---- aux2 = aux2 + column;

#(SE TRADUCE) aux2 = matrix_b->array[aux2];

        lw  t0, 60($fp)             #me paro en matriz_b->rows
    addi t0,t0,8                #me paro en matriz_b->array
    lw  t0, 0(t0)               #cargo la direccion de comienzo del array de elementos de
        sll t7, t7, 3               #multiplico por 8 para que equivalga a la cantidad de
    add t0, t0, t7              #se desplaza el puntero al array de matriz_b->array
        l.d $f4, 0(t0)              #cargo el valor apuntado por matrix_b->array[aux2] en

#(SE TRADUCE) aux1 = aux1 * aux2;

        mul.d $f2, $f2, $f4

#(SE TRADUCE) element = element + aux1;

        l.d  $f0, 32($fp)           #cargo en f0 element, puede estar de mas dado que no ve
        add.d $f0,$f0,$f2           #element = element + aux1
        s.d  $f0, 32($fp)                   #element se guarda en el frame en la posicion 3

        addi t5, t5, 1                  #j++.
    b segundo_for

add_element:

#(SE TRADUCE) matrix_c->array[i] = element;

        lw  t0, 28($fp)             #me paro en matriz_c->rows
    addi t0,t0,8                #me paro en matriz_c->array
    lw  t0, 0(t0)               #cargo la direccion de comienzo del array de elementos de m
        sll t8, t3, 3               #multiplico por 8 para que equivalga a la cantidad de b
        add  t0, t0, t8             #se desplaza el puntero al array de matriz_c->array
    l.d $f0, 32($fp)
        s.d $f0, 0(t0)              #guardo el valor de element f0 en matrix_c->array[i].

        addi t3, t3, 1                  #i++.
    b primer_for

error_mem:
        li v0,0
    b fin

return_matrix_multiply:
    lw v0,28($fp)

fin:
    lw  ra,48(sp)
        lw      $fp,44(sp)
        addu sp,sp,56

        jr      ra
        .end    matrix_multiply
```

7. Codigo MIPS32 de create matrix

```
#include <mips/regdef.h>

        .text
    .abicalls
        .ent create_matrix
        .globl create_matrix

create_matrix:

#Se crea stack frame de la callee

        .frame  $fp,40,ra
        subu    sp,sp,40

#Se crea SRA de la callee

    sw  ra,32(sp)
        .cprestore 24
        sw      $fp,28(sp)
        move    $fp,sp

#Se guardan parametros en ABA de caller

    sw a0,40($fp)               # guardo rows
    sw a1,44($fp)               # guardo cols

#(SE TRADUCE) matrix_t *matriz = malloc(sizeof(matrix_t));

    li a0,12                    # quiero 12 bytes = 4 bytes (cols) + 4 bytes (rows) + 4 by
    jal mymalloc

#(SE TRADUCE) if (matriz== NULL){ return NULL};

    beqz v0, error1
    sw v0,16($fp)               # guardo matriz en stackframe

#(SE TRADUCE) matriz->array = malloc(sizeof(double) * cols * rows);

    lw t0,40($fp)
    lw t1,44($fp)
    mul t0,t0,t1
    sll t0,t0,3                 # quiero rows bytes * cols bytes * 8 bytes (double)

    move a0,t0
    jal mymalloc

#(SE TRADUCE) if (matriz->array== NULL){free(matriz); return NULL};

    beqz v0, error2
    sw v0,20($fp)               # guardo array en stackframe

#asiganciones

    lw t0,40($fp)               # cargo rows (el valor numerico)
```

```
    lw t1,16($fp)               # cargo matriz (pero tambien estoy apuntando a matriz->row
    sw t0,0(t1)                 # cargo el valor numerico de row en matriz->rows

    lw t0,44($fp)               # cargo cols (el valor numerico)
    lw t1,16($fp)               # cargo matriz
    addiu t1,t1,4               # apunto a matriz->cols
    sw t0,0(t1)                 # cargo el valor numerico de cols en matriz->cols

    lw t0,20($fp)               # cargo array
    lw t1,16($fp)               # cargo matriz
    addiu t1,t1,8               # apunto a matriz->array
    sw t0,0(t1)                 # cargo el valor de array en matriz->array

    lw t1,16($fp)               # cargo matriz
    move v0,t1                  # cargo el puntero a matriz a v0 para retorno
        b return_create_matrix

error1:
    li v0,0                     # se retorna NULL (osea 0)
    b return_create_matrix

error2:
    lw a0,16($fp)               # cargo matriz
    jal myfree
    li v0,0                     # se retorna NULL (osea 0)
    b return_create_matrix

return_create_matrix:
    lw  ra,32(sp)
        lw      $fp,28(sp)
        addu sp,sp,40

        jr      ra
        .end    create_matrix
```

8. Código fuente C

```
#include<stdio.h>
#include<string.h>
#include<stdlib.h>
#include<stdbool.h>
#include "mymalloc.h"

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


void printArray(int len,double* array){
    int i;
    for(i=0;i<len;i++){
        printf("elemento %d: %g\n", i, array[i]);
    }
}

void destroy_matrix(matrix_t* m){
    if (m != NULL){
        myfree(m->array);
        myfree(m);
    }
}

void raiseError(const char* s){

    fprintf(stderr,"\n");
    fprintf(stderr,"==========================\n");
    fprintf(stderr,"ERROR MESSAGE: %s\n",s);
    fprintf(stderr,"==========================\n");
    fprintf(stderr,"\n");
}

char *readLine(FILE* fp, double* array){
//The size is extended by the input with the value of the provisional
    int size = 10; //HARDCODED
    char *str = NULL;
    char *errorReturnValue = NULL;
    int ch;
    size_t len = 0;


    str = realloc(NULL, sizeof(char)*size);//size is start size
    if(str == NULL){
        free(array);
        raiseError("REALLOC ERROR: null pointer returned");
        return errorReturnValue;
    }
    else{
        while(EOF!=(ch=fgetc(fp)) && ch != '\n'){
            str[len++]=ch;
```

```
            if(len==size){

                str = realloc(str, sizeof(char)*(size+=16)); //HARDCODED
                if(str == NULL){
                    free(str);
                    free(array);
                    raiseError("REALLOC ERROR: null pointer returned");
                    return errorReturnValue;
                }
            }
        }
    }

    if (ferror(stdin) != 0){
        free(str);
        free(array);
        raiseError("FGETC ERROR: I/O error");
        return errorReturnValue;
    }

    str[len++]='\0';

    str = realloc(str, sizeof(char)*len);
    if (str == NULL){
        free(str);
        free(array);
        raiseError("REALLOC ERROR: null pointer returned");
        return errorReturnValue;
    }

    return str;
}

int readElementsInLine(int dimention, double* array){

    char* line = readLine(stdin, array); //line has all the characters of the current line
    char* head_line_pointer = line;

    float x;
    int offset;
    int i = 0;
    int returnValue;
    int cantidadDeElementosLeidos = 0;
    int errorReturnValue = 1;

    while (true)
    {
        returnValue = sscanf(head_line_pointer, "%g%n", &x, &offset);

        if (ferror(stdin) != 0){
            free(array);
            free(line);
            raiseError("SSCANF ERROR: I/O error");
            return errorReturnValue;
        }
```

```
        if (returnValue == 1){
            head_line_pointer += offset;
            array[i] = (double)x;
            i++;
                        if (i > dimention*dimention*2){
                free(array);
                free(line);
                                raiseError("La cantidad de numeros es mayor a lo especifica
                                return errorReturnValue;
                        }
            continue;
        }

        if (returnValue == -1){
            cantidadDeElementosLeidos = i;
            if(cantidadDeElementosLeidos != dimention*dimention*2){
                free(array);
                free(line);
                raiseError("La cantidad de numeros es menor a lo especificado segun la dim
                return errorReturnValue;
            }
            break;
        }

        if (returnValue != 1){
            free(array);
            free(line);
            raiseError("Input no numerico");
            return errorReturnValue;
        }
    }
    free(line);
    return 0;
}

double* readInput(int* dimention){
    float firstInputElement;//initialized as double to check if corrupted input
    double* array = NULL;
    int returnValue;
    double* errorReturnValue = NULL;

    //READ FIRST
    returnValue = fscanf(stdin,"%g", &firstInputElement);

    //CHECK IF END OF LINE
    if (returnValue == -1){
        if (ferror(stdin) != 0){
            raiseError("FSCANF ERROR: I/O error");
            return errorReturnValue;
        }
        else{
            return errorReturnValue;
        }
    }
    //CHECK IF INPUT IS NUMERIC
    if (returnValue != 1){
```

```
        raiseError("Dimension no numerica");
        return errorReturnValue;
    }

    //CHECK IF INPUT IS TYPE UINT
    float mantiza = firstInputElement - (int)firstInputElement;
    if (mantiza > 0 || (firstInputElement <= 0)){
        raiseError("La dimension no es entera positiva");
        return errorReturnValue;
    }

    //ALLOCATE MEMORY FOR MATRICES INPUT ELEMENTS
    (*dimention) = (int)firstInputElement;
    array = malloc(sizeof(double)*(*dimention)*(*dimention)*2);

    //CHECK IF ALLOCATION IS SUCCESSFULL
    if (array == NULL){
        raiseError("No se pudo allocar memoria para inputs");
        return errorReturnValue;
    }
    //READ WHOLE LINE
    if (readElementsInLine((*dimention), array) != 0){
        return errorReturnValue;
    }
    return array;
}

int outputFile(FILE* out, char fileName[]){
    //ADAPTS FILE NAME
    char s[100] = "";
    strcat(s, "./");
    strcat(s, fileName);
    int return_value;
    int errorReturnValue = 1;

    //TRIES TO OPEN FILE
    FILE* fp;
    fp = fopen(s,"r");
    if(fp == NULL){
        raiseError("no se pudo abrir archivo de salida");
        return errorReturnValue;
    }

    //OUTPUTS
    char c;
        while(c != EOF){
        c = getc(fp);
        if ((return_value = fprintf(out,"%c",c)) < 0){
            raiseError("FPRINTF ERROR: I/O error");
            return errorReturnValue;
        }
        }

    if (ferror(stdin) != 0){
        raiseError("FGETC ERROR: I/O error");
        return errorReturnValue;
```

```
    }
    return 0;
}

matrix_t* create_matrix(size_t rows, size_t cols);// If returns null, there has been an er

void fillUpMatrices(matrix_t* matrix_a, matrix_t* matrix_b, int dimention,double* input){

    int i;
    for (i = 0; i < dimention*dimention; i++){
        matrix_a->array[i] = input[i];
    }

    for (i = dimention*dimention; i < dimention*dimention*2; i++){
        matrix_b->array[i - dimention*dimention] = input[i];
    }
}

matrix_t* matrix_multiply(matrix_t* matrix_a,matrix_t* matrix_b);

int print_matrix(FILE* out, matrix_t* matrix_m){
    int dimention = matrix_m->rows;
    double x;
    int i;

    if (fprintf(out,"%d",dimention) < 0){ //se mira que el valor no sea negativo porque si
        return 1;
    }
    if (fprintf(out,"%c",' ') < 0){
        return 1;
    }

        for (i = 0; i < dimention*dimention; i++){
        x = matrix_m->array[i];
        if (fprintf(out,"%g",x) < 0){
            return 1;
        }
        if (fprintf(out,"%c",' ') < 0){
            return 1;
        }
        }
    if (fprintf(out,"\n") < 0){
        return 1;
    }
    return 0; // si retorna 0 significa que no hubo errores.
}

int main(int argc, const char* argv[]){

    //INITIALIZATION
    FILE* OUT = stdout;
    bool endProgram = false;

    //HANDELING COMANDS
    if (argc > 1){
        if (strcmp(argv[1],"-h") == 0 || strcmp(argv[1],"--help") == 0){
```

```
            char fileName[] = "help";
            if ((outputFile(OUT, fileName)) != 0){
                return 1;
            }
            endProgram = true;
        }

        else if (strcmp(argv[1],"-V") == 0 || strcmp(argv[1],"--version") == 0){
            char fileName[] = "version";
            if ((outputFile(OUT, fileName)) != 0){
                return 1;
            }
            endProgram = true;
        }
        else{
            raiseError("command parameter invalid");
            return 1;
        }
    }

    //MAIN PROGRAM
    double* input = NULL;
    matrix_t* matrix_a = NULL;
    matrix_t* matrix_b = NULL;
    matrix_t* matrix_c = NULL;

    while (!endProgram){
        int dimention;
        input = readInput(&dimention);

        if (input == NULL){
            return 1;
        }
        matrix_a = create_matrix(dimention, dimention);
        if (matrix_a == NULL){
            free(input);
            raiseError("No se pudo allocar memoria para elementos de matriz");
            return 1;
        }

        matrix_b = create_matrix(dimention, dimention);
        if (matrix_b == NULL){
            destroy_matrix(matrix_a);
            free(input);
            raiseError("No se pudo allocar memoria para elementos de matriz");
            return 1;
        }
        fillUpMatrices(matrix_a, matrix_b, dimention, input);

        matrix_c = matrix_multiply(matrix_a, matrix_b);
        if (matrix_c == NULL){
            free(input);
            destroy_matrix(matrix_a);
            destroy_matrix(matrix_b);
            raiseError("No se pudo allocar memoria para elementos de matriz resultante");
            return 1;
```

```
        }

        if (print_matrix(OUT, matrix_c) != 0){
            free(input);
            destroy_matrix(matrix_a);
            destroy_matrix(matrix_b);
            destroy_matrix(matrix_c);
            raiseError("FPRINTF ERROR: I/O error");
            return 1;
        }

        free(input);
        destroy_matrix(matrix_a);
        destroy_matrix(matrix_b);
        destroy_matrix(matrix_c);
    }
    return 0;
}
```

9. Codigo MIPS32 del codigo C

```
        .section .mdebug.abi32
        .previous
        .abicalls
        .rdata
        .align  2
$LC0:
        .ascii  "elemento %d: %g\n\000"
        .text
        .align  2
        .globl  printArray
        .ent    printArray
printArray:
        .frame  $fp,48,$ra              # vars= 8, regs= 3/0, args= 16, extra= 8
        .mask   0xd0000000,-8
        .fmask  0x00000000,0
        .set    noreorder
        .cpload $t9
        .set    reorder
        subu    $sp,$sp,48
        .cprestore 16
        sw      $ra,40($sp)
        sw      $fp,36($sp)
        sw      $gp,32($sp)
        move    $fp,$sp
        sw      $a0,48($fp)
        sw      $a1,52($fp)
        sw      $zero,24($fp)
$L18:
        lw      $v0,24($fp)
        lw      $v1,48($fp)
        slt     $v0,$v0,$v1
        bne     $v0,$zero,$L21
        b       $L17
$L21:
        lw      $v0,24($fp)
        sll     $v1,$v0,3
        lw      $v0,52($fp)
        addu    $v0,$v1,$v0
        la      $a0,$LC0
        lw      $a1,24($fp)
        lw      $a2,0($v0)
        lw      $a3,4($v0)
        la      $t9,printf
        jal     $ra,$t9
        lw      $v0,24($fp)
        addu    $v0,$v0,1
        sw      $v0,24($fp)
        b       $L18
$L17:
        move    $sp,$fp
        lw      $ra,40($sp)
        lw      $fp,36($sp)
        addu    $sp,$sp,48
        j       $ra
```

10. Enunciado

Se adjunta el enunciado del trábajo práctico 1.

Universidad de Buenos Aires - FIUBA
66.20 Organización de Computadoras
Trabajo práctico 1: Programación MIPS
2^{do} cuatrimestre de 2019

`$Date: 2019/10/01 23:05:25 $`

1. Objetivos

Familiarizarse con el conjunto de instrucciones MIPS y el concepto de ABI, extendiendo un programa que resuelva el problema descripto a continuación.

2. Alcance

Este trabajo práctico es de elaboración grupal, evaluación individual, y de carácter obligatorio para todos alumnos del curso.

3. Requisitos

El trabajo deberá ser entregado personalmente, en la fecha estipulada, con una carátula que contenga los datos completos de todos los integrantes, un informe impreso de acuerdo con lo que mencionaremos en la sección 6, y con una copia digital de los archivos fuente necesarios para compilar el trabajo.

4. Descripción

El programa, deberá multiplicar matrices cuadradas de números reales, representados en punto flotante de doble precisión.

Las matrices a multiplicar ingresarán como texto por entrada estándar (`stdin`), donde cáda línea describe completamente cada par de matrices a multiplicar, según el siguiente formato:

$N\ a_{1,1}\ a_{1,2}\ ...\ a_{N,N}\ b_{1,1}\ b_{1,2}\ ...\ b_{N,N}$

La línea anterior representa a las matrices A y B, de NxN. Los elementos de la matriz A son los $a_{x,y}$, siendo x e y los indices de fila y columna respectivamente[1]. Los elementos de la matriz B se representan por los $b_{x,y}$ de la misma forma que los de A.

[1] Notar que es una representación del tipo *row major order*, siguiendo el orden en que C dispone las matrices en memoria.

El fin de línea es el caracter \n (*newline*). Los componentes de la línea están separados entre sí por uno o más espacios. El formato de los números en punto flotante son los que corresponden al especificador de conversión 'g' de `printf`[2].

Por ejemplo, dado el siguiente producto de matrices cuadradas:

$$\begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix} \times \begin{pmatrix} 5 & 6 \\ 7 & 8 \end{pmatrix}$$

Su representación sería:

```
2 1 2 3 4 5 6 7 8
```

Por cada par de matrices que se presenten por cada línea de entrada, el programa deberá multiplicarlas y presentar el resultado por su salida estándar (`stdout`) en el siguiente formato, hasta que llegue al final del archivo de entrada (`EOF`):

$N\ c_{1,1}\ c_{1,2}\ ...\ c_{N,N}$

Ante un error, el progama deberá informar la situación inmediatamente (por `stderr`) y detener su ejecución.

[2] Ver man 3 printf, "Conversion specifiers".

4.1. Ejemplos

Primero, usamos la opción `-h` para ver el mensaje de ayuda:

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

A continuación, ejecutamos algunas pruebas:

```
$ 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
```

En este ejemplo, realizamos las siguientes multiplicaciones, siendo los miembros izquierdos de la ecuación las matrices de entrada (`stdin`), y los miembros derechos las matrices de salida (`stdout`):

$$\begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix} \times \begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix} = \begin{pmatrix} 7 & 10 \\ 15 & 22 \end{pmatrix}$$

$$\begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6.1 \\ 3 & 2 & 1 \end{pmatrix} \times \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6.1 \\ 3 & 2 & 1 \end{pmatrix}$$

4.2. Interfaz

Las matrices deberán ser representadas por el tipo de datos `matrix_t`, definido a continuación:

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

Notar que los atributos `rows` y `cols` representan respectivamente la cantidad filas y columnas de la matriz. El atributo `array` contendrá los elementos de la matriz dispuestos en row-major order [3].

Los métodos a implementar, que aplican sobre el tipo de datos `matrix_t` son:

```
// Constructor de matrix_t
matrix_t* create_matrix(size_t rows, size_t cols);

// Destructor de matrix_t
void destroy_matrix(matrix_t* m);

// Imprime matrix_t sobre el file pointer fp en el formato solicitado
// por el enunciado
int print_matrix(FILE* fp, matrix_t* m);

// Multiplica las matrices en m1 y m2
matrix_t* matrix_multiply(matrix_t* m1, matrix_t* m2);
```

5. Implementación

A diferencia del TP anterior, en este caso deberá suministrarse una implementación de `matrix_multiply()` en código assembly MIPS. Así, el resto del programa deberá estar escrito en C, e interactuar con esta función para realiar la multiplicación de matrices.

6. Informe

El informe deberá incluir:

- Documentación relevante al diseño e implementación del programa;
- Comando(s) para compilar el programa;
- Las corridas de prueba, con los comentarios pertinentes;
- El código fuente, en lenguaje C y MIPS;
- El código MIPS32 generado por el compilador[3];

[3]Por motivos prácticos, en la copia impresa sólo es necesario incluir la primera página del código assembly MIPS32 generado por el compilador.

- Este enunciado.

7. Fechas

Fecha de vencimiento: martes 15/10.

Referencias

[1] GXemul, `http://gavare.se/gxemul/`.

[2] The NetBSD project, `http://www.netbsd.org/`.

[3] Row-major order (Wikipedia), `https://en.wikipedia.org/wiki/Row-major_order`.