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

Trabajo Práctico N°0

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

1. Comandos para compilación

Los comandos que ejecutamos para compilar el programa en Linux y NETBSD respectivamente fueron exactamente los provistos por la catedra sin ninguna modificación, si con la el recado de ejecutarlos sobre el directorio contenedor del archivo C.

Estos son para LINUX:
gcc -Wall -o Tp0 Tp0.c

Para NETBSD:
gcc -Wall -O0 Tp0.cmediumskip
gcc -Wall -O0 -S -mrnames Tp0.c (en el caso de querer generar solo el archivo assembly para MIPS32.

2. Diseño e implementación

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
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
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. Pruebas

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

En ella se encuentra una bateria de 27 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 27)
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 13)
2.32 8 4 6 5 8 7 7 8

Ejemplo de caso con dimensión no entera.

(Prueba 25)
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.

4. Código fuente C

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

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

double* input = NULL;           //GLOBAL ACCESS VARIABLE
matrix_t* matrix_a = NULL;      //GLOBAL ACCESS VARIABLE
matrix_t* matrix_b = NULL;      //GLOBAL ACCESS VARIABLE
matrix_t* matrix_c = NULL;      //GLOBAL ACCESS VARIABLE

void freeInputArray(){
    if (input != NULL){
        free(input);
    }
    input = NULL;
}

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){
        free(m->array);
        free(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");

    destroy_matrix(matrix_a);
    destroy_matrix(matrix_b);
    destroy_matrix(matrix_c);
    freeInputArray();
    exit (EXIT_FAILURE);
}

char *readLine(FILE* fp){
//The size is extended by the input with the value of the provisional
```

```
    int size = 10; //HARDCODED
    char *str;
    int ch;
    size_t len = 0;

    str = realloc(NULL, sizeof(char)*size);//size is start size
    if(!str)return str;
    while(EOF!=(ch=fgetc(fp)) && ch != '\n'){
        str[len++]=ch;
        if(len==size){
            str = realloc(str, sizeof(char)*(size+=16)); //HARDCODED
            if(!str)return str;
        }
    }

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

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

    str = realloc(str, sizeof(char)*len);

    if (str == NULL){
        raiseError("REALLOC ERROR: null pointer returned");
    }

    return str;
}

void readElementsInLine(int dimention, double* array){

    char* line = readLine(stdin);
    char* head_line_pointer = line;

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

    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");
        }

        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
                    }
            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
            }
            break;
        }

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

    }
}

double* readInput(int* dimention){

    float firstInputElement;//initialized as double to check if corrupted input
    double* array;
    int returnValue;

    //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");}}
        else{exit (0);} // en este caso se identifica que el EOF no se debe a un error y s
    }
    //CHECK IF INPUT IS NUMERIC
    if (returnValue != 1){
        raiseError("Dimension no numerica");
    }

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

    //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");
    }
    //READ WHOLE LINE
    readElementsInLine((*dimention), array);

    return array;
}

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

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

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

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

matrix_t* create_matrix(size_t rows, size_t cols){
    matrix_t *matriz = malloc(sizeof(matrix_t));
    if (matriz == NULL){ //si no puede reservar la memoria, deja el puntero en NULL
        raiseError("no se pudo allocar memoria para matriz");
    }
    matriz->array = malloc(sizeof(double) * cols * rows); //representara los elementos de
    if (matriz->array == NULL){ //si no puede reservar la memoria, deja el puntero en NULL
        free(matriz);
        raiseError("no se pudo allocar memoria para elementos de matriz");
    }
    matriz->rows = rows;
    matriz->cols = cols;
    return matriz;
}

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 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;
}

void print_matrix(FILE* out, matrix_t* matrix_m){

    int dimention = matrix_m->rows;
    double x;
    int i;
    int return_value;

    if ((return_value = fprintf(out,"%d",dimention)) < 0){raiseError("FPRINTF ERROR: I/O e
    if ((return_value = fprintf(out,"%c",' ')) < 0){raiseError("FPRINTF ERROR: I/O error")

        for (i = 0; i < dimention*dimention; i++){
        x = matrix_m->array[i];
        if ((return_value = fprintf(out,"%g",x)) < 0){raiseError("FPRINTF ERROR: I/O error
        if ((return_value = fprintf(out,"%c",' ')) < 0){raiseError("FPRINTF ERROR: I/O err
        }
    if ((return_value = fprintf(out,"\n")) < 0){raiseError("FPRINTF ERROR: I/O error");}
}

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";
            outputFile(OUT, fileName);
            endProgram = true;
        }

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

    //MAIN PROGRAM
    while (!endProgram){

        int dimention;
        input = readInput(&dimention);
        matrix_a = create_matrix(dimention, dimention);
        matrix_b = create_matrix(dimention, dimention);
        fillUpMatrices(matrix_a, matrix_b, dimention, input);

        matrix_c = matrix_multiply(matrix_a, matrix_b);
        print_matrix(OUT, matrix_c);

        destroy_matrix(matrix_a);
        destroy_matrix(matrix_b);
        destroy_matrix(matrix_c);
        freeInputArray();
    }
    return 0;
}
```

5. Codigo Assembly MIPS32

```
                .file   1 "tp0.c"
        .section .mdebug.abi32
        .previous
        .abicalls
        .globl  input
        .globl  input
        .section        .bss
        .align  2
        .type   input, @object
        .size   input, 4
input:
        .space  4
        .globl  matrix_a
        .globl  matrix_a
        .align  2
        .type   matrix_a, @object
        .size   matrix_a, 4
matrix_a:
        .space  4
        .globl  matrix_b
        .globl  matrix_b
        .align  2
        .type   matrix_b, @object
        .size   matrix_b, 4
matrix_b:
        .space  4
        .globl  matrix_c
        .globl  matrix_c
        .align  2
        .type   matrix_c, @object
        .size   matrix_c, 4
matrix_c:
        .space  4
        .text
        .align  2
        .globl  freeInputArray
        .ent    freeInputArray
freeInputArray:
        .frame  $fp,40,$ra              # vars= 0, regs= 3/0, args= 16, extra= 8
        .mask   0xd0000000,-8
        .fmask  0x00000000,0
        .set    noreorder
        .cpload $t9
        .set    reorder
        subu    $sp,$sp,40
        .cprestore 16
        sw      $ra,32($sp)
        sw      $fp,28($sp)
        sw      $gp,24($sp)
        move    $fp,$sp
        lw      $v0,input
        beq     $v0,$zero,$L18
        lw      $a0,input
        la      $t9,free
```

```
        jal     $ra,$t9
$L18:
        sw      $zero,input
        move    $sp,$fp
        lw      $ra,32($sp)
        lw      $fp,28($sp)
        addu    $sp,$sp,40
        j       $ra
        .end    freeInputArray
        .size   freeInputArray, .-freeInputArray
        .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)
$L20:
        lw      $v0,24($fp)
        lw      $v1,48($fp)
        slt     $v0,$v0,$v1
        bne     $v0,$zero,$L23
        b       $L19
$L23:
        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       $L20
$L19:
```

```
        move    $sp,$fp
        lw      $ra,40($sp)
        lw      $fp,36($sp)
        addu    $sp,$sp,48
        j       $ra
        .end    printArray
        .size   printArray, .-printArray
        .align  2
        .globl  destroy_matrix
        .ent    destroy_matrix
destroy_matrix:
        .frame  $fp,40,$ra              # vars= 0, regs= 3/0, args= 16, extra= 8
        .mask   0xd0000000,-8
        .fmask  0x00000000,0
        .set    noreorder
        .cpload $t9
        .set    reorder
        subu    $sp,$sp,40
        .cprestore 16
        sw      $ra,32($sp)
        sw      $fp,28($sp)
        sw      $gp,24($sp)
        move    $fp,$sp
        sw      $a0,40($fp)
        lw      $v0,40($fp)
        beq     $v0,$zero,$L24
        lw      $v0,40($fp)
        lw      $a0,8($v0)
        la      $t9,free
        jal     $ra,$t9
        lw      $a0,40($fp)
        la      $t9,free
        jal     $ra,$t9
$L24:
        move    $sp,$fp
        lw      $ra,32($sp)
        lw      $fp,28($sp)
        addu    $sp,$sp,40
        j       $ra
        .end    destroy_matrix
        .size   destroy_matrix, .-destroy_matrix
        .rdata
        .align  2
$LC1:
        .ascii  "\n\000"
        .align  2
$LC2:
        .ascii  "==========\n\000"
        .align  2
$LC3:
        .ascii  "ERROR MESSAGE: %s\n\000"
        .text
        .align  2
        .globl  raiseError
        .ent    raiseError
raiseError:
```

```
        .frame  $fp,40,$ra              # vars= 0, regs= 3/0, args= 16, extra= 8
        .mask   0xd0000000,-8
        .fmask  0x00000000,0
        .set    noreorder
        .cpload $t9
        .set    reorder
        subu    $sp,$sp,40
        .cprestore 16
        sw      $ra,32($sp)
        sw      $fp,28($sp)
        sw      $gp,24($sp)
        move    $fp,$sp
        sw      $a0,40($fp)
        la      $a0,__sF+176
        la      $a1,$LC1
        la      $t9,fprintf
        jal     $ra,$t9
        la      $a0,__sF+176
        la      $a1,$LC2
        la      $t9,fprintf
        jal     $ra,$t9
        la      $a0,__sF+176
        la      $a1,$LC3
        lw      $a2,40($fp)
        la      $t9,fprintf
        jal     $ra,$t9
        la      $a0,__sF+176
        la      $a1,$LC2
        la      $t9,fprintf
        jal     $ra,$t9
        la      $a0,__sF+176
        la      $a1,$LC1
        la      $t9,fprintf
        jal     $ra,$t9
        lw      $a0,matrix_a
        la      $t9,destroy_matrix
        jal     $ra,$t9
        lw      $a0,matrix_b
        la      $t9,destroy_matrix
        jal     $ra,$t9
        lw      $a0,matrix_c
        la      $t9,destroy_matrix
        jal     $ra,$t9
        la      $t9,freeInputArray
        jal     $ra,$t9
        li      $a0,1                   # 0x1
        la      $t9,exit
        jal     $ra,$t9
        .end    raiseError
        .size   raiseError, .-raiseError
        .rdata
        .align  2
$LC4:
        .ascii  "FGETC ERROR: I/O error\000"
        .align  2
$LC5:
```

```
        .ascii  "REALLOC ERROR: null pointer returned\000"
        .text
        .align  2
        .globl  readLine
        .ent    readLine
readLine:
        .frame  $fp,64,$ra              # vars= 24, regs= 3/0, args= 16, extra= 8
        .mask   0xd0000000,-8
        .fmask  0x00000000,0
        .set    noreorder
        .cpload $t9
        .set    reorder
        subu    $sp,$sp,64
        .cprestore 16
        sw      $ra,56($sp)
        sw      $fp,52($sp)
        sw      $gp,48($sp)
        move    $fp,$sp
        sw      $a0,64($fp)
        li      $v0,10                  # 0xa
        sw      $v0,24($fp)
        sw      $zero,36($fp)
        move    $a0,$zero
        lw      $a1,24($fp)
        la      $t9,realloc
        jal     $ra,$t9
        sw      $v0,28($fp)
        lw      $v0,28($fp)
        bne     $v0,$zero,$L28
        lw      $v0,28($fp)
        sw      $v0,40($fp)
        b       $L27
$L28:
        .set    noreorder
        nop
        .set    reorder
$L29:
        lw      $a0,64($fp)
        la      $t9,fgetc
        jal     $ra,$t9
        sw      $v0,32($fp)
        lw      $v1,32($fp)
        li      $v0,-1                  # 0xffffffffffffffff
        beq     $v1,$v0,$L30
        lw      $v1,32($fp)
        li      $v0,10                  # 0xa
        bne     $v1,$v0,$L31
        b       $L30
$L31:
        addu    $a1,$fp,36
        lw      $v1,0($a1)
        move    $a0,$v1
        lw      $v0,28($fp)
        addu    $a0,$a0,$v0
        lbu     $v0,32($fp)
        sb      $v0,0($a0)
```

```
        addu    $v1,$v1,1
        sw      $v1,0($a1)
        lw      $v1,36($fp)
        lw      $v0,24($fp)
        bne     $v1,$v0,$L29
        lw      $v0,24($fp)
        addu    $v0,$v0,16
        sw      $v0,24($fp)
        lw      $a0,28($fp)
        move    $a1,$v0
        la      $t9,realloc
        jal     $ra,$t9
        sw      $v0,28($fp)
        lw      $v0,28($fp)
        bne     $v0,$zero,$L29
        lw      $v0,28($fp)
        sw      $v0,40($fp)
        b       $L27
$L30:
        lhu     $v0,__sF+12
        srl     $v0,$v0,6
        andi    $v0,$v0,0x1
        beq     $v0,$zero,$L35
        lw      $a0,28($fp)
        la      $t9,free
        jal     $ra,$t9
        la      $a0,$LC4
        la      $t9,raiseError
        jal     $ra,$t9
$L35:
        addu    $a1,$fp,36
        lw      $v1,0($a1)
        move    $a0,$v1
        lw      $v0,28($fp)
        addu    $v0,$a0,$v0
        sb      $zero,0($v0)
        addu    $v1,$v1,1
        sw      $v1,0($a1)
        lw      $a0,28($fp)
        lw      $a1,36($fp)
        la      $t9,realloc
        jal     $ra,$t9
        sw      $v0,28($fp)
        lw      $v0,28($fp)
        bne     $v0,$zero,$L36
        la      $a0,$LC5
        la      $t9,raiseError
        jal     $ra,$t9
$L36:
        lw      $v0,28($fp)
        sw      $v0,40($fp)
$L27:
        lw      $v0,40($fp)
        move    $sp,$fp
        lw      $ra,56($sp)
        lw      $fp,52($sp)
```

```
        addu    $sp,$sp,64
        j       $ra
        .end    readLine
        .size   readLine, .-readLine
        .rdata
        .align  2
$LC6:
        .ascii  "%g%n\000"
        .align  2
$LC7:
        .ascii  "SSCANF ERROR: I/O error\000"
        .align  2
$LC8:
        .ascii  "La cantidad de numeros es mayor a lo especificado segun "
        .ascii  "la dimension\000"
        .align  2
$LC9:
        .ascii  "La cantidad de numeros es menor a lo especificado segun "
        .ascii  "la dimension\000"
        .align  2
$LC10:
        .ascii  "Input no numerico\000"
        .text
        .align  2
        .globl  readElementsInLine
        .ent    readElementsInLine
readElementsInLine:
        .frame  $fp,72,$ra              # vars= 32, regs= 3/0, args= 16, extra= 8
        .mask   0xd0000000,-8
        .fmask  0x00000000,0
        .set    noreorder
        .cpload $t9
        .set    reorder
        subu    $sp,$sp,72
        .cprestore 16
        sw      $ra,64($sp)
        sw      $fp,60($sp)
        sw      $gp,56($sp)
        move    $fp,$sp
        sw      $a0,72($fp)
        sw      $a1,76($fp)
        la      $a0,__sF
        la      $t9,readLine
        jal     $ra,$t9
        sw      $v0,24($fp)
        lw      $v0,24($fp)
        sw      $v0,28($fp)
        sw      $zero,40($fp)
        sw      $zero,48($fp)
$L38:
        addu    $v0,$fp,32
        addu    $v1,$fp,36
        lw      $a0,28($fp)
        la      $a1,$LC6
        move    $a2,$v0
        move    $a3,$v1
```

```
        la      $t9,sscanf
        jal     $ra,$t9
        sw      $v0,44($fp)
        lhu     $v0,__sF+12
        srl     $v0,$v0,6
        andi    $v0,$v0,0x1
        beq     $v0,$zero,$L41
        lw      $a0,76($fp)
        la      $t9,free
        jal     $ra,$t9
        lw      $a0,24($fp)
        la      $t9,free
        jal     $ra,$t9
        la      $a0,$LC7
        la      $t9,raiseError
        jal     $ra,$t9
$L41:
        lw      $v1,44($fp)
        li      $v0,1                   # 0x1
        bne     $v1,$v0,$L42
        lw      $v1,28($fp)
        lw      $v0,36($fp)
        addu    $v0,$v1,$v0
        sw      $v0,28($fp)
        lw      $v0,40($fp)
        sll     $v1,$v0,3
        lw      $v0,76($fp)
        addu    $v0,$v1,$v0
        l.s     $f0,32($fp)
        cvt.d.s $f0,$f0
        s.d     $f0,0($v0)
        lw      $v0,40($fp)
        addu    $v0,$v0,1
        sw      $v0,40($fp)
        lw      $v1,72($fp)
        lw      $v0,72($fp)
        mult    $v1,$v0
        mflo    $v0
        sll     $v1,$v0,1
        lw      $v0,40($fp)
        slt     $v0,$v1,$v0
        beq     $v0,$zero,$L38
        lw      $a0,76($fp)
        la      $t9,free
        jal     $ra,$t9
        lw      $a0,24($fp)
        la      $t9,free
        jal     $ra,$t9
        la      $a0,$LC8
        la      $t9,raiseError
        jal     $ra,$t9
        b       $L38
$L42:
        lw      $v1,44($fp)
        li      $v0,-1                  # 0xffffffffffffffff
        bne     $v1,$v0,$L44
```

```
        lw      $v0,40($fp)
        sw      $v0,48($fp)
        lw      $v1,72($fp)
        lw      $v0,72($fp)
        mult    $v1,$v0
        mflo    $v0
        sll     $v1,$v0,1
        lw      $v0,48($fp)
        beq     $v0,$v1,$L37
        lw      $a0,76($fp)
        la      $t9,free
        jal     $ra,$t9
        lw      $a0,24($fp)
        la      $t9,free
        jal     $ra,$t9
        la      $a0,$LC9
        la      $t9,raiseError
        jal     $ra,$t9
        b       $L37
$L44:
        lw      $v1,44($fp)
        li      $v0,1                   # 0x1
        beq     $v1,$v0,$L38
        lw      $a0,76($fp)
        la      $t9,free
        jal     $ra,$t9
        lw      $a0,24($fp)
        la      $t9,free
        jal     $ra,$t9
        la      $a0,$LC10
        la      $t9,raiseError
        jal     $ra,$t9
$L37:
        move    $sp,$fp
        lw      $ra,64($sp)
        lw      $fp,60($sp)
        addu    $sp,$sp,72
        j       $ra
        .end    readElementsInLine
        .size   readElementsInLine, .-readElementsInLine
        .rdata
        .align  2
$LC11:
        .ascii  "%g\000"
        .align  2
$LC12:
        .ascii  "FSCANF ERROR: I/O error\000"
        .align  2
$LC13:
        .ascii  "Dimension no numerica\000"
        .align  2
$LC14:
        .ascii  "La dimension no es entera positiva\000"
        .align  2
$LC15:
        .ascii  "No se pudo allocar memoria para inputs\000"
```

```
        .text
        .align  2
        .globl  readInput
        .ent    readInput
readInput:
        .frame  $fp,56,$ra              # vars= 16, regs= 3/0, args= 16, extra= 8
        .mask   0xd0000000,-8
        .fmask  0x00000000,0
        .set    noreorder
        .cpload $t9
        .set    reorder
        subu    $sp,$sp,56
        .cprestore 16
        sw      $ra,48($sp)
        sw      $fp,44($sp)
        sw      $gp,40($sp)
        move    $fp,$sp
        sw      $a0,56($fp)
        la      $a0,__sF
        la      $a1,$LC11
        addu    $a2,$fp,24
        la      $t9,fscanf
        jal     $ra,$t9
        sw      $v0,32($fp)
        lw      $v1,32($fp)
        li      $v0,-1                  # 0xffffffffffffffff
        bne     $v1,$v0,$L48
        lhu     $v0,__sF+12
        srl     $v0,$v0,6
        andi    $v0,$v0,0x1
        beq     $v0,$zero,$L49
        la      $a0,$LC12
        la      $t9,raiseError
        jal     $ra,$t9
        b       $L48
$L49:
        move    $a0,$zero
        la      $t9,exit
        jal     $ra,$t9
$L48:
        lw      $v1,32($fp)
        li      $v0,1                   # 0x1
        beq     $v1,$v0,$L51
        la      $a0,$LC13
        la      $t9,raiseError
        jal     $ra,$t9
$L51:
        l.s     $f0,24($fp)
        trunc.w.s $f0,$f0,$v0
        cvt.s.w $f2,$f0
        l.s     $f0,24($fp)
        sub.s   $f0,$f0,$f2
        s.s     $f0,36($fp)
        l.s     $f2,36($fp)
        mtc1    $zero,$f0
        c.lt.s  $f0,$f2
```

```
        bc1t    $L53
        l.s     $f2,24($fp)
        mtc1    $zero,$f0
        c.le.s  $f2,$f0
        bc1t    $L53
        b       $L52
$L53:
        la      $a0,$LC14
        la      $t9,raiseError
        jal     $ra,$t9
$L52:
        lw      $v0,56($fp)
        l.s     $f0,24($fp)
        trunc.w.s $f0,$f0,$v1
        s.s     $f0,0($v0)
        lw      $v0,56($fp)
        lw      $v1,56($fp)
        lw      $a0,0($v0)
        lw      $v0,0($v1)
        mult    $a0,$v0
        mflo    $v0
        sll     $v0,$v0,4
        move    $a0,$v0
        la      $t9,malloc
        jal     $ra,$t9
        sw      $v0,28($fp)
        lw      $v0,28($fp)
        bne     $v0,$zero,$L54
        la      $a0,$LC15
        la      $t9,raiseError
        jal     $ra,$t9
$L54:
        lw      $v0,56($fp)
        lw      $a0,0($v0)
        lw      $a1,28($fp)
        la      $t9,readElementsInLine
        jal     $ra,$t9
        lw      $v0,28($fp)
        move    $sp,$fp
        lw      $ra,48($sp)
        lw      $fp,44($sp)
        addu    $sp,$sp,56
        j       $ra
        .end    readInput
        .size   readInput, .-readInput
        .rdata
        .align  2
$LC16:
        .ascii  "\000"
        .space  99
        .align  2
$LC17:
        .ascii  "./\000"
        .align  2
$LC18:
        .ascii  "r\000"
```

```
        .align  2
$LC19:
        .ascii  "no se pudo abrir archivo de salida\000"
        .align  2
$LC20:
        .ascii  "%c\000"
        .align  2
$LC21:
        .ascii  "FPRINTF ERROR: I/O error\000"
        .text
        .align  2
        .globl  outputFile
        .ent    outputFile
outputFile:
        .frame  $fp,160,$ra             # vars= 120, regs= 3/0, args= 16, extra= 8
        .mask   0xd0000000,-8
        .fmask  0x00000000,0
        .set    noreorder
        .cpload $t9
        .set    reorder
        subu    $sp,$sp,160
        .cprestore 16
        sw      $ra,152($sp)
        sw      $fp,148($sp)
        sw      $gp,144($sp)
        move    $fp,$sp
        sw      $a0,160($fp)
        sw      $a1,164($fp)
        lbu     $v0,$LC16
        sb      $v0,24($fp)
        addu    $v0,$fp,25
        li      $v1,99                  # 0x63
        move    $a0,$v0
        move    $a1,$zero
        move    $a2,$v1
        la      $t9,memset
        jal     $ra,$t9
        addu    $a0,$fp,24
        la      $a1,$LC17
        la      $t9,strcat
        jal     $ra,$t9
        addu    $a0,$fp,24
        lw      $a1,164($fp)
        la      $t9,strcat
        jal     $ra,$t9
        addu    $a0,$fp,24
        la      $a1,$LC18
        la      $t9,fopen
        jal     $ra,$t9
        sw      $v0,132($fp)
        lw      $v0,132($fp)
        bne     $v0,$zero,$L56
        la      $a0,$LC19
        la      $t9,raiseError
        jal     $ra,$t9
$L56:
```

```
        .set    noreorder
        nop
        .set    reorder
$L57:
        lb      $v1,136($fp)
        li      $v0,-1                  # 0xffffffffffffffff
        bne     $v1,$v0,$L59
        b       $L58
$L59:
        lw      $v1,132($fp)
        lw      $v0,132($fp)
        lw      $v0,4($v0)
        addu    $v0,$v0,-1
        sw      $v0,4($v1)
        bgez    $v0,$L60
        lw      $a0,132($fp)
        la      $t9,__srget
        jal     $ra,$t9
        sb      $v0,137($fp)
        b       $L61
$L60:
        lw      $v0,132($fp)
        lw      $v1,0($v0)
        move    $a0,$v1
        lbu     $a0,0($a0)
        sb      $a0,137($fp)
        addu    $v1,$v1,1
        sw      $v1,0($v0)
$L61:
        lbu     $v0,137($fp)
        sb      $v0,136($fp)
        lb      $v0,136($fp)
        lw      $a0,160($fp)
        la      $a1,$LC20
        move    $a2,$v0
        la      $t9,fprintf
        jal     $ra,$t9
        sw      $v0,128($fp)
        lw      $v0,128($fp)
        bgez    $v0,$L57
        la      $a0,$LC21
        la      $t9,raiseError
        jal     $ra,$t9
        b       $L57
$L58:
        lhu     $v0,__sF+12
        srl     $v0,$v0,6
        andi    $v0,$v0,0x1
        beq     $v0,$zero,$L55
        la      $a0,$LC4
        la      $t9,raiseError
        jal     $ra,$t9
$L55:
        move    $sp,$fp
        lw      $ra,152($sp)
        lw      $fp,148($sp)
```

```
        addu    $sp,$sp,160
        j       $ra
        .end    outputFile
        .size   outputFile, .-outputFile
        .rdata
        .align  2
$LC22:
        .ascii  "no se pudo allocar memoria para matriz\000"
        .align  2
$LC23:
        .ascii  "no se pudo allocar memoria para elementos de matriz\000"
        .text
        .align  2
        .globl  create_matrix
        .ent    create_matrix
create_matrix:
        .frame  $fp,48,$ra              # vars= 8, regs= 4/0, args= 16, extra= 8
        .mask   0xd0010000,-4
        .fmask  0x00000000,0
        .set    noreorder
        .cpload $t9
        .set    reorder
        subu    $sp,$sp,48
        .cprestore 16
        sw      $ra,44($sp)
        sw      $fp,40($sp)
        sw      $gp,36($sp)
        sw      $s0,32($sp)
        move    $fp,$sp
        sw      $a0,48($fp)
        sw      $a1,52($fp)
        li      $a0,12                  # 0xc
        la      $t9,malloc
        jal     $ra,$t9
        sw      $v0,24($fp)
        lw      $v0,24($fp)
        bne     $v0,$zero,$L65
        la      $a0,$LC22
        la      $t9,raiseError
        jal     $ra,$t9
$L65:
        lw      $s0,24($fp)
        lw      $v1,52($fp)
        lw      $v0,48($fp)
        mult    $v1,$v0
        mflo    $v0
        sll     $v0,$v0,3
        move    $a0,$v0
        la      $t9,malloc
        jal     $ra,$t9
        sw      $v0,8($s0)
        lw      $v0,24($fp)
        lw      $v0,8($v0)
        bne     $v0,$zero,$L66
        lw      $a0,24($fp)
        la      $t9,free
```

```
        jal     $ra,$t9
        la      $a0,$LC23
        la      $t9,raiseError
        jal     $ra,$t9
$L66:
        lw      $v1,24($fp)
        lw      $v0,48($fp)
        sw      $v0,0($v1)
        lw      $v1,24($fp)
        lw      $v0,52($fp)
        sw      $v0,4($v1)
        lw      $v0,24($fp)
        move    $sp,$fp
        lw      $ra,44($sp)
        lw      $fp,40($sp)
        lw      $s0,32($sp)
        addu    $sp,$sp,48
        j       $ra
        .end    create_matrix
        .size   create_matrix, .-create_matrix
        .align  2
        .globl  fillUpMatrices
        .ent    fillUpMatrices
fillUpMatrices:
        .frame  $fp,24,$ra              # vars= 8, regs= 2/0, args= 0, extra= 8
        .mask   0x50000000,-4
        .fmask  0x00000000,0
        .set    noreorder
        .cpload $t9
        .set    reorder
        subu    $sp,$sp,24
        .cprestore 0
        sw      $fp,20($sp)
        sw      $gp,16($sp)
        move    $fp,$sp
        sw      $a0,24($fp)
        sw      $a1,28($fp)
        sw      $a2,32($fp)
        sw      $a3,36($fp)
        sw      $zero,8($fp)
$L68:
        lw      $v1,32($fp)
        lw      $v0,32($fp)
        mult    $v1,$v0
        mflo    $v1
        lw      $v0,8($fp)
        slt     $v0,$v0,$v1
        bne     $v0,$zero,$L71
        b       $L69
$L71:
        lw      $a0,24($fp)
        lw      $v0,8($fp)
        sll     $v1,$v0,3
        lw      $v0,8($a0)
        addu    $a0,$v1,$v0
        lw      $v0,8($fp)
```

```
        sll     $v1,$v0,3
        lw      $v0,36($fp)
        addu    $v0,$v1,$v0
        l.d     $f0,0($v0)
        s.d     $f0,0($a0)
        lw      $v0,8($fp)
        addu    $v0,$v0,1
        sw      $v0,8($fp)
        b       $L68
$L69:
        lw      $v0,32($fp)
        lw      $v1,32($fp)
        mult    $v0,$v1
        mflo    $v0
        sw      $v0,8($fp)
$L72:
        lw      $v1,32($fp)
        lw      $v0,32($fp)
        mult    $v1,$v0
        mflo    $v0
        sll     $v1,$v0,1
        lw      $v0,8($fp)
        slt     $v0,$v0,$v1
        bne     $v0,$zero,$L75
        b       $L67
$L75:
        lw      $a0,28($fp)
        lw      $v1,32($fp)
        lw      $v0,32($fp)
        mult    $v1,$v0
        mflo    $v1
        lw      $v0,8($fp)
        subu    $v0,$v0,$v1
        sll     $v1,$v0,3
        lw      $v0,8($a0)
        addu    $a0,$v1,$v0
        lw      $v0,8($fp)
        sll     $v1,$v0,3
        lw      $v0,36($fp)
        addu    $v0,$v1,$v0
        l.d     $f0,0($v0)
        s.d     $f0,0($a0)
        lw      $v0,8($fp)
        addu    $v0,$v0,1
        sw      $v0,8($fp)
        b       $L72
$L67:
        move    $sp,$fp
        lw      $fp,20($sp)
        addu    $sp,$sp,24
        j       $ra
        .end    fillUpMatrices
        .size   fillUpMatrices, .-fillUpMatrices
        .align  2
        .globl  matrix_multiply
        .ent    matrix_multiply
```

```
matrix_multiply:
        .frame  $fp,72,$ra              # vars= 32, regs= 3/0, args= 16, extra= 8
        .mask   0xd0000000,-8
        .fmask  0x00000000,0
        .set    noreorder
        .cpload $t9
        .set    reorder
        subu    $sp,$sp,72
        .cprestore 16
        sw      $ra,64($sp)
        sw      $fp,60($sp)
        sw      $gp,56($sp)
        move    $fp,$sp
        sw      $a0,72($fp)
        sw      $a1,76($fp)
        lw      $v0,72($fp)
        lw      $v0,0($v0)
        sw      $v0,24($fp)
        lw      $a0,24($fp)
        lw      $a1,24($fp)
        la      $t9,create_matrix
        jal     $ra,$t9
        sw      $v0,28($fp)
        sw      $zero,40($fp)
$L77:
        lw      $v1,24($fp)
        lw      $v0,24($fp)
        mult    $v1,$v0
        mflo    $v1
        lw      $v0,40($fp)
        slt     $v0,$v0,$v1
        bne     $v0,$zero,$L80
        b       $L78
$L80:
        lw      $v1,40($fp)
        lw      $v0,24($fp)
        div     $0,$v1,$v0
        mflo    $v1
        .set    noreorder
        bne     $v0,$0,1f
        nop
        break   7
1:
        .set    reorder
        sw      $v1,32($fp)
        lw      $v1,40($fp)
        lw      $v0,24($fp)
        div     $0,$v1,$v0
        mfhi    $v1
        .set    noreorder
        bne     $v0,$0,1f
        nop
        break   7
1:
        .set    reorder
        sw      $v1,36($fp)
```

```
        sw      $zero,48($fp)
        sw      $zero,52($fp)
        sw      $zero,44($fp)
$L81:
        lw      $v0,44($fp)
        lw      $v1,24($fp)
        slt     $v0,$v0,$v1
        bne     $v0,$zero,$L84
        b       $L82
$L84:
        lw      $a0,72($fp)
        lw      $v1,32($fp)
        lw      $v0,24($fp)
        mult    $v1,$v0
        mflo    $v1
        lw      $v0,44($fp)
        addu    $v0,$v1,$v0
        sll     $v1,$v0,3
        lw      $v0,8($a0)
        addu    $a1,$v1,$v0
        lw      $a0,76($fp)
        lw      $v1,44($fp)
        lw      $v0,24($fp)
        mult    $v1,$v0
        mflo    $v1
        lw      $v0,36($fp)
        addu    $v0,$v1,$v0
        sll     $v1,$v0,3
        lw      $v0,8($a0)
        addu    $v0,$v1,$v0
        l.d     $f2,0($a1)
        l.d     $f0,0($v0)
        mul.d   $f2,$f2,$f0
        l.d     $f0,48($fp)
        add.d   $f0,$f0,$f2
        s.d     $f0,48($fp)
        lw      $v0,44($fp)
        addu    $v0,$v0,1
        sw      $v0,44($fp)
        b       $L81
$L82:
        lw      $a0,28($fp)
        lw      $v0,40($fp)
        sll     $v1,$v0,3
        lw      $v0,8($a0)
        addu    $v0,$v1,$v0
        l.d     $f0,48($fp)
        s.d     $f0,0($v0)
        lw      $v0,40($fp)
        addu    $v0,$v0,1
        sw      $v0,40($fp)
        b       $L77
$L78:
        lw      $v0,28($fp)
        move    $sp,$fp
        lw      $ra,64($sp)
```

```
        lw      $fp,60($sp)
        addu    $sp,$sp,72
        j       $ra
        .end    matrix_multiply
        .size   matrix_multiply, .-matrix_multiply
        .rdata
        .align  2
$LC24:
        .ascii  "%d\000"
        .text
        .align  2
        .globl  print_matrix
        .ent    print_matrix
print_matrix:
        .frame  $fp,64,$ra              # vars= 24, regs= 3/0, args= 16, extra= 8
        .mask   0xd0000000,-8
        .fmask  0x00000000,0
        .set    noreorder
        .cpload $t9
        .set    reorder
        subu    $sp,$sp,64
        .cprestore 16
        sw      $ra,56($sp)
        sw      $fp,52($sp)
        sw      $gp,48($sp)
        move    $fp,$sp
        sw      $a0,64($fp)
        sw      $a1,68($fp)
        lw      $v0,68($fp)
        lw      $v0,0($v0)
        sw      $v0,24($fp)
        lw      $a0,64($fp)
        la      $a1,$LC24
        lw      $a2,24($fp)
        la      $t9,fprintf
        jal     $ra,$t9
        sw      $v0,44($fp)
        lw      $v0,44($fp)
        bgez    $v0,$L86
        la      $a0,$LC21
        la      $t9,raiseError
        jal     $ra,$t9
$L86:
        lw      $a0,64($fp)
        la      $a1,$LC20
        li      $a2,32                  # 0x20
        la      $t9,fprintf
        jal     $ra,$t9
        sw      $v0,44($fp)
        lw      $v0,44($fp)
        bgez    $v0,$L87
        la      $a0,$LC21
        la      $t9,raiseError
        jal     $ra,$t9
$L87:
        sw      $zero,40($fp)
```

```
$L88:
        lw      $v1,24($fp)
        lw      $v0,24($fp)
        mult    $v1,$v0
        mflo    $v1
        lw      $v0,40($fp)
        slt     $v0,$v0,$v1
        bne     $v0,$zero,$L91
        b       $L89
$L91:
        lw      $a0,68($fp)
        lw      $v0,40($fp)
        sll     $v1,$v0,3
        lw      $v0,8($a0)
        addu    $v0,$v1,$v0
        l.d     $f0,0($v0)
        s.d     $f0,32($fp)
        lw      $a0,64($fp)
        la      $a1,$LC11
        lw      $a2,32($fp)
        lw      $a3,36($fp)
        la      $t9,fprintf
        jal     $ra,$t9
        sw      $v0,44($fp)
        lw      $v0,44($fp)
        bgez    $v0,$L92
        la      $a0,$LC21
        la      $t9,raiseError
        jal     $ra,$t9
$L92:
        lw      $a0,64($fp)
        la      $a1,$LC20
        li      $a2,32                  # 0x20
        la      $t9,fprintf
        jal     $ra,$t9
        sw      $v0,44($fp)
        lw      $v0,44($fp)
        bgez    $v0,$L90
        la      $a0,$LC21
        la      $t9,raiseError
        jal     $ra,$t9
$L90:
        lw      $v0,40($fp)
        addu    $v0,$v0,1
        sw      $v0,40($fp)
        b       $L88
$L89:
        lw      $a0,64($fp)
        la      $a1,$LC1
        la      $t9,fprintf
        jal     $ra,$t9
        sw      $v0,44($fp)
        lw      $v0,44($fp)
        bgez    $v0,$L85
        la      $a0,$LC21
        la      $t9,raiseError
```

```
        jal     $ra,$t9
$L85:
        move    $sp,$fp
        lw      $ra,56($sp)
        lw      $fp,52($sp)
        addu    $sp,$sp,64
        j       $ra
        .end    print_matrix
        .size   print_matrix, .-print_matrix
        .rdata
        .align  2
$LC25:
        .ascii  "-h\000"
        .align  2
$LC26:
        .ascii  "--help\000"
        .align  2
$LC27:
        .ascii  "help\000"
        .align  2
$LC28:
        .ascii  "-V\000"
        .align  2
$LC29:
        .ascii  "--version\000"
        .align  2
$LC30:
        .ascii  "version\000"
        .align  2
$LC31:
        .ascii  "command parameter invalid\000"
        .text
        .align  2
        .globl  main
        .ent    main
main:
        .frame  $fp,72,$ra              # vars= 32, regs= 3/0, args= 16, extra= 8
        .mask   0xd0000000,-8
        .fmask  0x00000000,0
        .set    noreorder
        .cpload $t9
        .set    reorder
        subu    $sp,$sp,72
        .cprestore 16
        sw      $ra,64($sp)
        sw      $fp,60($sp)
        sw      $gp,56($sp)
        move    $fp,$sp
        sw      $a0,72($fp)
        sw      $a1,76($fp)
        la      $v0,__sF+88
        sw      $v0,24($fp)
        sb      $zero,28($fp)
        lw      $v0,72($fp)
        slt     $v0,$v0,2
        bne     $v0,$zero,$L96
```

```
        lw      $v0,76($fp)
        addu    $v0,$v0,4
        lw      $a0,0($v0)
        la      $a1,$LC25
        la      $t9,strcmp
        jal     $ra,$t9
        beq     $v0,$zero,$L98
        lw      $v0,76($fp)
        addu    $v0,$v0,4
        lw      $a0,0($v0)
        la      $a1,$LC26
        la      $t9,strcmp
        jal     $ra,$t9
        bne     $v0,$zero,$L97
$L98:
        lw      $v0,$LC27
        sw      $v0,32($fp)
        lbu     $v0,$LC27+4
        sb      $v0,36($fp)
        addu    $v0,$fp,32
        lw      $a0,24($fp)
        move    $a1,$v0
        la      $t9,outputFile
        jal     $ra,$t9
        li      $v0,1                   # 0x1
        sb      $v0,28($fp)
        b       $L96
$L97:
        lw      $v0,76($fp)
        addu    $v0,$v0,4
        lw      $a0,0($v0)
        la      $a1,$LC28
        la      $t9,strcmp
        jal     $ra,$t9
        beq     $v0,$zero,$L101
        lw      $v0,76($fp)
        addu    $v0,$v0,4
        lw      $a0,0($v0)
        la      $a1,$LC29
        la      $t9,strcmp
        jal     $ra,$t9
        bne     $v0,$zero,$L100
$L101:
        lw      $v0,$LC30
        sw      $v0,40($fp)
        lw      $v0,$LC30+4
        sw      $v0,44($fp)
        addu    $v0,$fp,40
        lw      $a0,24($fp)
        move    $a1,$v0
        la      $t9,outputFile
        jal     $ra,$t9
        li      $v0,1                   # 0x1
        sb      $v0,28($fp)
        b       $L96
$L100:
```

```
        la      $a0,$LC31
        la      $t9,raiseError
        jal     $ra,$t9
$L96:
        .set    noreorder
        nop
        .set    reorder
$L103:
        lbu     $v0,28($fp)
        beq     $v0,$zero,$L105
        b       $L104
$L105:
        addu    $v0,$fp,48
        move    $a0,$v0
        la      $t9,readInput
        jal     $ra,$t9
        sw      $v0,input
        lw      $a0,48($fp)
        lw      $a1,48($fp)
        la      $t9,create_matrix
        jal     $ra,$t9
        sw      $v0,matrix_a
        lw      $a0,48($fp)
        lw      $a1,48($fp)
        la      $t9,create_matrix
        jal     $ra,$t9
        sw      $v0,matrix_b
        lw      $a0,matrix_a
        lw      $a1,matrix_b
        lw      $a2,48($fp)
        lw      $a3,input
        la      $t9,fillUpMatrices
        jal     $ra,$t9
        lw      $a0,matrix_a
        lw      $a1,matrix_b
        la      $t9,matrix_multiply
        jal     $ra,$t9
        sw      $v0,matrix_c
        lw      $a0,24($fp)
        lw      $a1,matrix_c
        la      $t9,print_matrix
        jal     $ra,$t9
        lw      $a0,matrix_a
        la      $t9,destroy_matrix
        jal     $ra,$t9
        lw      $a0,matrix_b
        la      $t9,destroy_matrix
        jal     $ra,$t9
        lw      $a0,matrix_c
        la      $t9,destroy_matrix
        jal     $ra,$t9
        la      $t9,freeInputArray
        jal     $ra,$t9
        b       $L103
$L104:
        move    $v0,$zero
```

```
move    $sp,$fp
lw      $ra,64($sp)
lw      $fp,60($sp)
addu    $sp,$sp,72
j       $ra
.end    main
.size   main, .-main
.ident  "GCC: (GNU) 3.3.3 (NetBSD nb3 20040520)"
```

6. Enunciado

Se adjunta el enunciado del trábajo práctico 0.

7. Enunciado

Universidad de Buenos Aires - FIUBA
66.20 Organización de Computadoras
Trabajo práctico 0: Infraestructura básica
2^{do} cuatrimestre de 2019

`$Date: 2019/08/27 23:02:40 $`

1. Objetivos

Familiarizarse con las herramientas de software que usaremos en los siguientes trabajos, implementando un programa y su correspondiente documentación que resuelvan el problema descripto más abajo.

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. Recursos

Usaremos el programa GXemul [1] para simular el entorno de desarrollo que utilizaremos en este y otros trabajos prácticos, una máquina MIPS corriendo una versión reciente del sistema operativo NetBSD [2].

Durante la primera clase del curso hemos presentado brevemente los pasos necesarios para la instalación y configuración del entorno de desarrollo.

5. Implementación

5.1. Programa

El programa, a escribir en lenguaje C, 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.

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.

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

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

5.2. Ejemplos

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

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

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

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

5.3. 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);
```

5.4. Portabilidad

Como es usual, es necesario que la implementación desarrollada provea un grado mínimo de portabilidad. Para satisfacer esto, el programa deberá funcionar al menos en NetBSD/pmax (usando el simulador GXemul [1]) y la versión de Linux (Knoppix, RedHat, Debian, Ubuntu) usada para correr el simulador, Linux/i386.

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;

- El código MIPS32 generado por el compilador[3];
- Este enunciado.

7. Fechas

Fecha de vencimiento: martes 24/9.

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`.

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