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;

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

void raiseError(const char* s){

    fprintf(st\,"\n");
    fprintf(stderr,"==========\n");
    fprintf(stderr,"ERROR MESSAGE: %s\n",s);
    fprintf(stderr,"==========\n");
    fprintf(stderr,"\n");
    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;
        }
    }
    str[len++]='\0';

    return realloc(str, sizeof(char)*len);
}

void readElementsInLine(int dimention, double* array){

    char* line = readLine(stdin);

    float x;
```

```
    int offset;
    int i = 0;
    int returnValue;
    int cantidadDeElementosLeidos;

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

        if (returnValue == 1){
            line += offset;
            array[i] = (double)x;
            i++;
            continue;
        }

        if (returnValue == -1){
            cantidadDeElementosLeidos = i;
            if(cantidadDeElementosLeidos != dimention*dimention*2){
                free(array);
                raiseError("No coincide dimension con cantidad de elementos ingresados
            }
            break;
        }

        if (returnValue != 1){
            free(array);
            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){
        exit (0);
    }
    //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);

    //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);
        fprintf(out,"%c",c);
    }
}

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

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

```
    }
}

void fillUpMatrices(matrix_t* matrix_a, matrix_t* matrix_b, int dimention,double* inpu

    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*dimentio
        }
        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;

    fprintf(out,"%d",dimention);
    fprintf(out,"%c",' ');

    for (i = 0; i < dimention*dimention; i++){
        x = matrix_m->array[i];
            fprintf(out,"%g",x);
        fprintf(out,"%c",' ');
    }
```

```
    fprintf(out,"\n");
}

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;
        }

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

    //MAIN PROGRAM
    bool thereAreMoreProductsToDo = true;
    while (thereAreMoreProductsToDo && !endProgram){

        matrix_t* matrix_a;
        matrix_t* matrix_b;
        matrix_t* matrix_c;

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

        //printArray(dimention*dimention,matrix_a->array);
        //printArray(dimention*dimention,matrix_b->array);

        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);

        if (input != NULL){
            free(input);
        }
    }
    return 0;
}
```

5. Codigo Assembly MIPS32

```
        .file   1 "Tp0.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
```

6. Enunciado

Se adjunta el enunciado del trábajo práctico 0.

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.