Universidad de Buenos Aires - FIUBA
66.20 Organización de Computadoras
Trabajo Práctico 0: Infraestructura Básica

Joaquin Segui, *Padrón Nro. 91.451*
segui.joaquin@gmail.com

Pernin Alejandro, *Padrón Nro. 92.216*
ale.pernin@gmail.com

Menniti Sebastián Ezequiel, *Padrón Nro. 93.445*
mennitise@gmail.com

1. Introducción

Se implemento un programa, en lenguaje C, que se encarga de multiplicar matrices de números reales, representados en punto flotante de doble precisión.

2. Diseño e Implementación

Las matrices a multiplicar se ingresan por entrada estándar (`stdin`), donde cada linea representada una matríz completa en formato de texto, describiendola mediante el siguiente formato:

$$NxM\ a_{1,1}\ a_{1,2}\ ...\ a_{1,M}\ a_{2,1}\ a_{2,2}\ ...\ a_{2,M}\ ...\ a_{N,1}\ a_{N,2}\ ...\ a_{N,M}$$

Esta linea representa a una matríz A, donde N es la cantidad de filas y M la cantidad de columnas de la matríz A. Los elementos de la matríz A son los $a_{x,y}$, donde x e y son los indices de fila y columna respectivamente. El fin de linea se delimita con el caracter *newline*.

Por cada par de matrices que se presentan en la entrada, el programa en primer lugar, se encarga de cargarlas, luego verifica que las matrices cumplan con la condición para que la multiplicación sea posible (Se verifica que la cantidad de columnas de la primer matríz sea igual a la cantidad de filas de la segunda matríz), y en el caso que la cumplan, se procede a multiplicarlas. El resultado obtenido lo presenta por salida estándar (`stdout`), en el mismo formato mencionado anteriormente. Este proceso se repite hasta que llegue al final del archivo de entrada (`EOF`). Si se encontrara con un error, el programa lo informa por `stderr` y se detiene su ejecución.

3. Comandos para compilar el programa

Para facilitar la compilacion se utiliza un *Makefile*, para invocar la compilación del programa ejecutar desde una consola, dentro del mismo directorio que el código fuente: `make`.

Asimismo se proviciona un script que realiza pruebas con un set de datos preexistente, para invocarlo: `./pruebas.sh`.

4. Pruebas

En esta sección se detallarán las pruebas realizadas. Los archivos utilizados se encuentran el el directorio *test_files*.

4.1. Casos Exitosos

Los casos exitosos están comprendidos por los set de datos *test1* y textittest2. En el caso del primero:

```
1x2 1 2
2x3 1 0 4 5 1 3
```

Representa la operación

$$\begin{pmatrix}1 & 2\end{pmatrix} * \begin{pmatrix}1 & 0 & 4\\ 5 & 1 & 3\end{pmatrix} = \begin{pmatrix}11 & 2 & 10\end{pmatrix}$$

cuya salida por consola mediante el script de pruebas es
`1x3 11 2 10`
como es esperable.

El segundo set de datos es:

```
3x1 1.000 2.00 3.00
1x3 0.0 3.000 1.000

1x2 1 3
2x3 1 0 4 5 1 0
```

representando las operaciones

$$\begin{pmatrix}1\\ 2\\ 3\end{pmatrix} * \begin{pmatrix}0 & 3 & 1\end{pmatrix} = \begin{pmatrix}0 & 3 & 1\\ 0 & 6 & 2\\ 0 & 9 & 3\end{pmatrix}$$

$$\begin{pmatrix}1 & 3\end{pmatrix} * \begin{pmatrix}1 & 0 & 4\\ 5 & 1 & 0\end{pmatrix} = \begin{pmatrix}16 & 3 & 4\end{pmatrix}$$

cuya salida se obtuvo correctamente

```
3x3 0 3 1 0 6 2 0 9 3
1x3 16 3 4
```

4.2. Casos de error

Como casos de error del programa probamos matrices incompatibles para su multiplicacion y matrices mal definidas.

Uno de estos casos es el de tener dos matrices cuyas dimensiones hacen incompatibles la multiplicación entre sí. Este es el caso del set *test3*.

```
1x2 1 2
1x3 1 0 4
```

$$\begin{pmatrix}1 & 2\end{pmatrix} * \begin{pmatrix}1 & 0 & 4\end{pmatrix}$$

Al ejecutar dicha prueba, el programa termina con el siguiente mensaje:

```
Dimensiones no compatibles para multiplicar
```

Otra prueba es tener una cantidad impar de matrices, por lo cuál una no podrá ser multiplicada. Por ejemplo *test4*.

```
3x1 1.000 2.00 3.00
1x3 0.0 3.000 1.000

1x2 1 3
```

Como resultado arroja

```
3x3 0 3 1 0 6 2 0 9 3
Fallo al leer dimensiones
```

Otros casos de prueba, consisten en definir dimensiones de matrices inconsistentes con la cantidad de elementos leidos. Al ver *test5*

```
3x1 1.000 2.00
1x3 0.0 3.000 1.000

1x2 1 3
2x3 1 0 4 5 1 0
```

$$\begin{pmatrix}1\\ 2\\ X\end{pmatrix} * \begin{pmatrix}0 & 3 & 1\end{pmatrix}$$

si bien las dimensiones declaradas son compatibles para su multiplicación, los elementos provistos son inconsistentes. Dicha prueba arroja:

```
Cantidad elementos distinta a dimensiones de matriz
```

5. Codigo fuente del programa

5.1. En lenguaje C

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

//Funcion que imprime el manual del TP0
void printManual(){
  printf("Usage:\n tp0 -h\n");
  printf(" tp0 -V\n");
  printf("tp0 < in_file > out_file\n");
  printf("Options:\n");
  printf(" -V, --version   Print version and quit.\n");
  printf(" -h, --help      Print this information and quit.\n");
  printf("Examples:\n");
  printf(" tp0 < in.txt > out.txt\n");
  printf("cat in.txt | tp0 > out.txt\n");
}

void parsearOpciones(int argc, char* argv[]) {
  int next_option;
  const char* const short_options = "hV";
  const struct option long_options[] = {
    { "help",     0, NULL, 'h' },
    { "version",  0, NULL, 'V' },
    { NULL,       0, NULL, 0   } //Necesario al final del array
  };
  //Procesamiento de los parametros de entrada.
  do {
    next_option = getopt_long(argc, argv, short_options, long_options, NULL);
    switch (next_option){
      case 'h':   // -h, --help
        printManual();
        exit(EXIT_SUCCESS);
        break;
      case 'V':   // -V, --version
        printf(" Version 1.0 del TP0\n");
        exit(EXIT_SUCCESS);
        break;
      case -1:   // Se terminaron las opciones
        break;
      default:    // Opcion incorrecta
        fprintf(stderr, "Error, el programa se cerrara.\n");
        printManual();
        exit(EXIT_FAILURE);
    }
  } while (next_option != -1);
}

double** alocarMatriz( int filas, int columnas) {
  double** matriz;
  matriz = (double**)malloc(filas*sizeof(double*));
  if (!matriz) {
    return NULL;
  }
  int i; //Recorre filas
  for(i=0;i<filas;i++){
    matriz[i] = (double*)malloc(columnas*sizeof(double));
    //Si falla el malloc, libero todo lo reservado anteriormente
    if(!matriz[i]) {
      int j;
      for (j=0;j<i;j++) {
        free(matriz[j]);
      }
      free(matriz);
      return NULL;
    }
  }
  return matriz;
```

```
70 }
71
72 size_t strLength(char* s){
73     size_t i;
74     for(i = 0; s[i] != 0; i++);
75     return i;
76 }
77
78 int llenarMatriz(double** matriz, int fila, int columna) {
79     int i;
80     int j;
81     char c;
82     int cantidadElementos = 0;
83     i = 0;
84     j = 0;
85     bool exito = true;
86     double valor;
87     int flag;
88     while (exito && i<fila) {
89       flag = scanf("%lf%c",&valor,&c);
90       if (flag != EOF && flag == 2) {
91         //printf("Leo: %lf y %c\n",valor,c);
92         matriz[i][j] = valor;
93         cantidadElementos++;
94         if (j==columna-1) {
95           j=0;
96           i++;
97         } else {
98           j++;
99         }
100      } else {
101        exito = false;
102      }
103    }
104    if (cantidadElementos != ((fila)*(columna)) || (c!='\n')) {
105      return EXIT_FAILURE;
106    }
107  return EXIT_SUCCESS;
108 }
109
110 void liberarMatriz(double** matriz, int fila) {
111   int i;
112   for(i=0;i<fila;++i) {
113     free(matriz[i]);
114   }
115   free(matriz);
116 }
117 void multiplicar(double** matriz1, int fila1, int columna1, double** matriz2, int columna2) {
118   int i;
119   int j;
120   int k;
121   double accum;
122   printf("%dx%d ",fila1,columna2);
123   for(i=0;i<fila1;i++) {
124     for(j=0;j<columna2;j++) {
125       accum = 0;
126       for(k=0;k<columna1;k++) {
127         accum = accum + (matriz1[i][k] * matriz2[k][j]);
128       }
129       printf("%g ",accum);
130     }
131   }
132   printf("\n");
133 }
134 int main(int argc, char *argv[]) {
135   parsearOpciones(argc,argv);
136   //Construyo la primera matriz
137   double** matriz1;
138   int fila1;
139   int columna1;
140   int cant;
141   cant =  scanf("%d %*c %d ",&fila1,&columna1);
142   do {
143     if (cant != 2) {
```

```
144       fprintf(stderr, "Fallo al leer dimensiones\n");
145       return EXIT_FAILURE;
146     }
147     matriz1 = alocarMatriz(fila1,columna1);
148     if (!matriz1) {
149       fprintf(stderr, "Fallo en malloc\n");
150       return EXIT_FAILURE;
151     }
152     int llenar;
153     llenar = llenarMatriz(matriz1,fila1,columna1);
154     if (llenar) {
155       liberarMatriz(matriz1,fila1);
156       fprintf(stderr, "Cantidad elementos distinta a dimensiones de matriz\n");
157       return EXIT_FAILURE;
158     }
159     //Repito para segunda matriz
160     double** matriz2;
161     int fila2;
162     int columna2;
163     cant = scanf("%i %*c %i ",&fila2,&columna2);
164     if (cant != 2) {
165       liberarMatriz(matriz1,fila1);
166       fprintf(stderr, "Fallo al leer dimensiones\n");
167       return EXIT_FAILURE;
168     }
169     matriz2 = alocarMatriz(fila2,columna2);
170     if (!matriz2) {
171       liberarMatriz(matriz1,fila1);
172       fprintf(stderr, "Fallo en malloc\n");
173       return EXIT_FAILURE;
174     }
175     llenar =  llenarMatriz(matriz2,fila2,columna2);
176     if (llenar) {
177       liberarMatriz(matriz1,fila1);
178       liberarMatriz(matriz2,fila2);
179       fprintf(stderr, "Cantidad elementos distinta a dimensiones de matriz\n");
180       return EXIT_FAILURE;
181     }
182     if(columna1 == fila2) {
183       //Multiplicar
184       multiplicar(matriz1,fila1,columna1,matriz2,columna2);
185       liberarMatriz(matriz1,fila1);
186       liberarMatriz(matriz2,fila2);
187     } else {
188       liberarMatriz(matriz1,fila1);
189       liberarMatriz(matriz2,fila2);
190       fprintf(stderr, "Dimensiones no compatibles para multiplicar\n");
191       return EXIT_FAILURE;
192     }
193     cant =  scanf("%i %*c %i ",&fila1,&columna1);
194   } while(cant != EOF);
195     //Repetir
196     return EXIT_SUCCESS;
197 }
```

5.2. Codigo MIPS32 generado por el compilador

```
1   .file 1 "main.c"
2   .section .mdebug.abi32
3   .previous
4   .abicalls
5   .rdata
6   .align 2
7 $LC0:
8   .ascii "Usage:\n"
9   .ascii " tp0 -h\n\000"
10  .align 2
11 $LC1:
12  .ascii " tp0 -V\n\000"
13  .align 2
14 $LC2:
15  .ascii "tp0 < in_file > out_file\n\000"
16  .align 2
17 $LC3:
18  .ascii "Options:\n\000"
19  .align 2
20 $LC4:
21  .ascii " -V, --version \tPrint version and quit.\n\000"
22  .align 2
23 $LC5:
24  .ascii " -h, --help  \tPrint this information and quit.\n\000"
25  .align 2
26 $LC6:
27  .ascii "Examples:\n\000"
28  .align 2
29 $LC7:
30  .ascii " tp0 < in.txt > out.txt\n\000"
31  .align 2
32 $LC8:
33  .ascii "cat in.txt | tp0 > out.txt\n\000"
34  .text
35  .align 2
36  .globl printManual
37  .ent printManual
38 printManual:
39  .frame $fp,40,$ra    # vars= 0, regs= 3/0, args= 16, extra= 8
40  .mask 0xd0000000,-8
41  .fmask 0x00000000,0
42  .set noreorder
43  .cpload $t9
44  .set reorder
45  subu $sp,$sp,40
46  .cprestore 16
47  sw $ra,32($sp)
48  sw $fp,28($sp)
49  sw $gp,24($sp)
50  move $fp,$sp
51  la $a0,$LC0
52  la $t9,printf
53  jal $ra,$t9
54  la $a0,$LC1
55  la $t9,printf
56  jal $ra,$t9
57  la $a0,$LC2
58  la $t9,printf
59  jal $ra,$t9
60  la $a0,$LC3
61  la $t9,printf
62  jal $ra,$t9
63  la $a0,$LC4
64  la $t9,printf
65  jal $ra,$t9
66  la $a0,$LC5
67  la $t9,printf
68  jal $ra,$t9
69  la $a0,$LC6
70  la $t9,printf
71  jal $ra,$t9
72  la $a0,$LC7
```

5.3. Enunciado

Universidad de Buenos Aires - FIUBA
66.20 Organización de Computadoras
Trabajo práctico 0: Infraestructura básica
2^{do} cuatrimestre de 2015

`$Date: 2015/09/01 00:28:25 $`

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 de números reales, representados en punto flotante de doble precisión.

Las matrices a multiplicar ingresarán por entrada estándar (`stdin`), donde cáda línea describe una matriz completa en formato de texto, según el siguiente formato:

$NxM\ a_{1,1}\ a_{1,2}\ ...\ a_{1,M}\ \ a_{2,1}\ a_{2,2}\ ...\ a_{2,M}\ \ ...\ \ a_{N,1}\ a_{N,2}\ ...\ a_{N,M}$

La línea anterior representa a la matriz A de N filas y M columnas. Los elementos de la matriz A son los $a_{x,y}$, siendo x e y los indices de fila y columna respectivamente[1]. 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, dada la siguiente matriz:

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

Su representación sería:

```
2x3 1 2 3 4 5 6
```

Por cada par de matrices que se presenten en su entrada, el programa deberá multiplicarlas y presentar el resultado por su salida estándar (`stdout`) en el mismo formato presentado anteriormente, hasta que llegue al final del archivo de entrada (`EOF`). Ante un error, el progama deberá informar la situación inmediatamente (por `stderr`) y detener su ejecución. Tener en cuenta que también se condidera un error que a la entrada se presenten matrices de dimensiones incompatibles entre sí para su multiplicación.

[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 in.txt
2x3 1 2 3 4 5 6.1
3x2 1 0 0 0 0 1
3x3 1 2 3 4 5 6.1 3 2 1
3x1 1 1 0

$ cat in.txt | ./tp0
2x2 1 3 4 6.1
3x1 3 9 5
```

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`):

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

$$\begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6.1 \\ 3 & 2 & 1 \end{pmatrix} \times \begin{pmatrix} 1 \\ 1 \\ 0 \end{pmatrix} = \begin{pmatrix} 3 \\ 9 \\ 5 \end{pmatrix}$$

5.3. 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 22/9/2015.

Referencias

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

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

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