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
```

```
 73   la   $t9,printf
 74   jal  $ra,$t9
 75   la   $a0,$LC8
 76   la   $t9,printf
 77   jal  $ra,$t9
 78   move  $sp,$fp
 79   lw   $ra,32($sp)
 80   lw   $fp,28($sp)
 81   addu  $sp,$sp,40
 82   j $ra
 83   .end  printManual
 84   .size printManual, .-printManual
 85   .rdata
 86   .align  2
 87 $LC10:
 88   .ascii  "help\000"
 89   .align  2
 90 $LC11:
 91   .ascii  "version\000"
 92   .data
 93   .align  2
 94 $LC12:
 95   .word $LC10
 96   .word 0
 97   .word 0
 98   .word 104
 99   .word $LC11
100   .word 0
101   .word 0
102   .word 86
103   .word 0
104   .word 0
105   .word 0
106   .word 0
107   .globl  memcpy
108   .rdata
109   .align  2
110 $LC9:
111   .ascii  "hV\000"
112   .align  2
113 $LC13:
114   .ascii  " Version 1.0 del TP0\n\000"
115   .align  2
116 $LC14:
117   .ascii  "Error, el programa se cerrara.\n\000"
118   .text
119   .align  2
120   .globl  parsearOpciones
121   .ent  parsearOpciones
122 parsearOpciones:
123   .frame  $fp,112,$ra   # vars= 64, regs= 3/0, args= 24, extra= 8
124   .mask 0xd0000000,-8
125   .fmask  0x00000000,0
126   .set  noreorder
127   .cpload $t9
128   .set  reorder
129   subu  $sp,$sp,112
130   .cprestore 24
131   sw  $ra,104($sp)
132   sw  $fp,100($sp)
133   sw  $gp,96($sp)
134   move  $fp,$sp
135   sw  $a0,112($fp)
136   sw  $a1,116($fp)
137   la  $v0,$LC9
138   sw  $v0,36($fp)
139   addu  $v0,$fp,40
140   la  $v1,$LC12
141   move  $a0,$v0
142   move  $a1,$v1
143   li  $a2,48      # 0x30
144   la  $t9,memcpy
145   jal  $ra,$t9
146 $L19:
```

```
147   addu  $v0,$fp,40
148   sw  $zero,16($sp)
149   lw  $a0,112($fp)
150   lw  $a1,116($fp)
151   lw  $a2,36($fp)
152   move  $a3,$v0
153   la  $t9,getopt_long
154   jal $ra,$t9
155   sw  $v0,32($fp)
156   lw  $v0,32($fp)
157   sw  $v0,88($fp)
158   li  $v0,86      # 0x56
159   lw  $v1,88($fp)
160   beq $v1,$v0,$L24
161   lw  $v1,88($fp)
162   slt $v0,$v1,87
163   beq $v0,$zero,$L28
164   li  $v0,-1      # 0xffffffffffffffff
165   lw  $v1,88($fp)
166   beq $v1,$v0,$L21
167   b $L26
168 $L28:
169   li  $v0,104     # 0x68
170   lw  $v1,88($fp)
171   beq $v1,$v0,$L23
172   b $L26
173 $L23:
174   la  $t9,printManual
175   jal $ra,$t9
176   move  $a0,$zero
177   la  $t9,exit
178   jal $ra,$t9
179 $L24:
180   la  $a0,$LC13
181   la  $t9,printf
182   jal $ra,$t9
183   move  $a0,$zero
184   la  $t9,exit
185   jal $ra,$t9
186 $L26:
187   la  $a0,__sF+176
188   la  $a1,$LC14
189   la  $t9,fprintf
190   jal $ra,$t9
191   la  $t9,printManual
192   jal $ra,$t9
193   li  $a0,1     # 0x1
194   la  $t9,exit
195   jal $ra,$t9
196 $L21:
197   lw  $v1,32($fp)
198   li  $v0,-1      # 0xffffffffffffffff
199   bne $v1,$v0,$L19
200   move  $sp,$fp
201   lw  $ra,104($sp)
202   lw  $fp,100($sp)
203   addu  $sp,$sp,112
204   j $ra
205   .end  parsearOpciones
206   .size parsearOpciones, .-parsearOpciones
207   .align  2
208   .globl  alocarMatriz
209   .ent  alocarMatriz
210 alocarMatriz:
211   .frame  $fp,56,$ra    # vars= 16, regs= 4/0, args= 16, extra= 8
212   .mask 0xd0010000,-4
213   .fmask  0x00000000,0
214   .set  noreorder
215   .cpload $t9
216   .set  reorder
217   subu  $sp,$sp,56
218   .cprestore 16
219   sw  $ra,52($sp)
220   sw  $fp,48($sp)
```

```
221   sw   $gp,44($sp)
222   sw   $s0,40($sp)
223   move   $fp,$sp
224   sw   $a0,56($fp)
225   sw   $a1,60($fp)
226   lw   $v0,56($fp)
227   sll $v0,$v0,2
228   move   $a0,$v0
229   la   $t9,malloc
230   jal $ra,$t9
231   sw   $v0,24($fp)
232   lw   $v0,24($fp)
233   bne $v0,$zero,$L31
234   sw   $zero,36($fp)
235   b $L30
236 $L31:
237   sw   $zero,28($fp)
238 $L32:
239   lw   $v0,28($fp)
240   lw   $v1,56($fp)
241   slt $v0,$v0,$v1
242   bne $v0,$zero,$L35
243   b $L33
244 $L35:
245   lw   $v0,28($fp)
246   sll $v1,$v0,2
247   lw   $v0,24($fp)
248   addu   $s0,$v1,$v0
249   lw   $v0,60($fp)
250   sll $v0,$v0,3
251   move   $a0,$v0
252   la   $t9,malloc
253   jal $ra,$t9
254   sw   $v0,0($s0)
255   lw   $v0,28($fp)
256   sll $v1,$v0,2
257   lw   $v0,24($fp)
258   addu   $v0,$v1,$v0
259   lw   $v0,0($v0)
260   bne $v0,$zero,$L34
261   sw   $zero,32($fp)
262 $L37:
263   lw   $v0,32($fp)
264   lw   $v1,28($fp)
265   slt $v0,$v0,$v1
266   bne $v0,$zero,$L40
267   b $L38
268 $L40:
269   lw   $v0,32($fp)
270   sll $v1,$v0,2
271   lw   $v0,24($fp)
272   addu   $v0,$v1,$v0
273   lw   $a0,0($v0)
274   la   $t9,free
275   jal $ra,$t9
276   lw   $v0,32($fp)
277   addu   $v0,$v0,1
278   sw   $v0,32($fp)
279   b $L37
280 $L38:
281   lw   $a0,24($fp)
282   la   $t9,free
283   jal $ra,$t9
284   sw   $zero,36($fp)
285   b $L30
286 $L34:
287   lw   $v0,28($fp)
288   addu   $v0,$v0,1
289   sw   $v0,28($fp)
290   b $L32
291 $L33:
292   lw   $v0,24($fp)
293   sw   $v0,36($fp)
294 $L30:
```

```
295   lw  $v0,36($fp)
296   move  $sp,$fp
297   lw  $ra,52($sp)
298   lw  $fp,48($sp)
299   lw  $s0,40($sp)
300   addu  $sp,$sp,56
301   j $ra
302   .end  alocarMatriz
303   .size alocarMatriz, .-alocarMatriz
304   .align  2
305   .globl  strLength
306   .ent  strLength
307 strLength:
308   .frame  $fp,24,$ra    # vars= 8, regs= 2/0, args= 0, extra= 8
309   .mask 0x50000000,-4
310   .fmask  0x00000000,0
311   .set  noreorder
312   .cpload $t9
313   .set  reorder
314   subu  $sp,$sp,24
315   .cprestore 0
316   sw  $fp,20($sp)
317   sw  $gp,16($sp)
318   move  $fp,$sp
319   sw  $a0,24($fp)
320   sw  $zero,8($fp)
321 $L42:
322   lw  $v1,24($fp)
323   lw  $v0,8($fp)
324   addu  $v0,$v1,$v0
325   lb  $v0,0($v0)
326   bne $v0,$zero,$L44
327   b $L43
328 $L44:
329   lw  $v0,8($fp)
330   addu  $v0,$v0,1
331   sw  $v0,8($fp)
332   b $L42
333 $L43:
334   lw  $v0,8($fp)
335   move  $sp,$fp
336   lw  $fp,20($sp)
337   addu  $sp,$sp,24
338   j $ra
339   .end  strLength
340   .size strLength, .-strLength
341   .align  2
342   .globl  getLine
343   .ent  getLine
344 getLine:
345   .frame  $fp,160,$ra   # vars= 120, regs= 3/0, args= 16, extra= 8
346   .mask 0xd0000000,-8
347   .fmask  0x00000000,0
348   .set  noreorder
349   .cpload $t9
350   .set  reorder
351   subu  $sp,$sp,160
352   .cprestore 16
353   sw  $ra,152($sp)
354   sw  $fp,148($sp)
355   sw  $gp,144($sp)
356   move  $fp,$sp
357   addu  $a0,$fp,24
358   li  $a1,100     # 0x64
359   la  $a2,__sF
360   la  $t9,fgets
361   jal $ra,$t9
362   bne $v0,$zero,$L47
363   sw  $zero,140($fp)
364   b $L46
365 $L47:
366   addu  $a0,$fp,24
367   la  $t9,strLength
368   jal $ra,$t9
```

```
369   sw   $v0,128($fp)
370   lw   $v1,128($fp)
371   addu   $v0,$fp,24
372   addu   $v0,$v0,$v1
373   addu   $v0,$v0,-1
374   lb   $v1,0($v0)
375   li   $v0,10       # 0xa
376   bne $v1,$v0,$L54
377   lw   $v0,128($fp)
378   sltu   $v0,$v0,2
379   bne $v0,$zero,$L49
380   lw   $v1,128($fp)
381   addu   $v0,$fp,24
382   addu   $v0,$v0,$v1
383   addu   $v0,$v0,-1
384   sb   $zero,0($v0)
385 $L49:
386   lw   $a0,128($fp)
387   la   $t9,malloc
388   jal $ra,$t9
389   sw   $v0,132($fp)
390   sw   $zero,136($fp)
391 $L50:
392   lw   $v0,136($fp)
393   lw   $v1,128($fp)
394   sltu   $v0,$v0,$v1
395   bne $v0,$zero,$L53
396   b $L51
397 $L53:
398   lw   $v1,132($fp)
399   lw   $v0,136($fp)
400   addu   $a0,$v1,$v0
401   lw   $v1,136($fp)
402   addu   $v0,$fp,24
403   addu   $v0,$v0,$v1
404   lbu $v0,0($v0)
405   sb   $v0,0($a0)
406   lw   $v0,136($fp)
407   addu   $v0,$v0,1
408   sw   $v0,136($fp)
409   b $L50
410 $L51:
411   lw   $v0,132($fp)
412   sw   $v0,140($fp)
413   b $L46
414 $L54:
415   sw   $zero,140($fp)
416 $L46:
417   lw   $v0,140($fp)
418   move   $sp,$fp
419   lw   $ra,152($sp)
420   lw   $fp,148($sp)
421   addu   $sp,$sp,160
422   j $ra
423   .end   getLine
424   .size getLine, .-getLine
425   .rdata
426   .align   2
427 $LC15:
428   .ascii   "%lf%c\000"
429   .text
430   .align   2
431   .globl   llenarMatriz
432   .ent   llenarMatriz
433 llenarMatriz:
434   .frame   $fp,80,$ra     # vars= 40, regs= 3/0, args= 16, extra= 8
435   .mask 0xd0000000,-8
436   .fmask   0x00000000,0
437   .set   noreorder
438   .cpload $t9
439   .set   reorder
440   subu   $sp,$sp,80
441   .cprestore 16
442   sw   $ra,72($sp)
```

```
443   sw   $fp,68($sp)
444   sw   $gp,64($sp)
445   move   $fp,$sp
446   sw   $a0,80($fp)
447   sw   $a1,84($fp)
448   sw   $a2,88($fp)
449   sw   $zero,36($fp)
450   sw   $zero,24($fp)
451   sw   $zero,28($fp)
452   li   $v0,1      # 0x1
453   sb   $v0,40($fp)
454 $L56:
455   lbu  $v0,40($fp)
456   beq  $v0,$zero,$L57
457   lw   $v0,24($fp)
458   lw   $v1,84($fp)
459   slt  $v0,$v0,$v1
460   bne  $v0,$zero,$L58
461   b $L57
462 $L58:
463   addu   $v0,$fp,48
464   addu   $v1,$fp,32
465   la   $a0,$LC15
466   move   $a1,$v0
467   move   $a2,$v1
468   la   $t9,scanf
469   jal  $ra,$t9
470   sw   $v0,56($fp)
471   lw   $v1,56($fp)
472   li   $v0,-1       # 0xffffffffffffffff
473   beq  $v1,$v0,$L60
474   lw   $v1,56($fp)
475   li   $v0,2      # 0x2
476   bne  $v1,$v0,$L60
477   lw   $v0,24($fp)
478   sll  $v1,$v0,2
479   lw   $v0,80($fp)
480   addu   $a0,$v1,$v0
481   lw   $v0,28($fp)
482   sll  $v1,$v0,3
483   lw   $v0,0($a0)
484   addu   $v0,$v1,$v0
485   l.d  $f0,48($fp)
486   s.d  $f0,0($v0)
487   lw   $v0,36($fp)
488   addu   $v0,$v0,1
489   sw   $v0,36($fp)
490   lw   $v0,88($fp)
491   addu   $v1,$v0,-1
492   lw   $v0,28($fp)
493   bne  $v0,$v1,$L61
494   sw   $zero,28($fp)
495   lw   $v0,24($fp)
496   addu   $v0,$v0,1
497   sw   $v0,24($fp)
498   b $L56
499 $L61:
500   lw   $v0,28($fp)
501   addu   $v0,$v0,1
502   sw   $v0,28($fp)
503   b $L56
504 $L60:
505   sb   $zero,40($fp)
506   b $L56
507 $L57:
508   lw   $v1,84($fp)
509   lw   $v0,88($fp)
510   mult   $v1,$v0
511   mflo   $v1
512   lw   $v0,36($fp)
513   bne  $v0,$v1,$L65
514   lb   $v1,32($fp)
515   li   $v0,10       # 0xa
516   bne  $v1,$v0,$L65
```

```
517   b $L64
518 $L65:
519   li  $v0,1     # 0x1
520   sw  $v0,60($fp)
521   b $L55
522 $L64:
523   sw  $zero,60($fp)
524 $L55:
525   lw  $v0,60($fp)
526   move  $sp,$fp
527   lw  $ra,72($sp)
528   lw  $fp,68($sp)
529   addu  $sp,$sp,80
530   j $ra
531   .end  llenarMatriz
532   .size llenarMatriz, .-llenarMatriz
533   .align  2
534   .globl  liberarMatriz
535   .ent  liberarMatriz
536 liberarMatriz:
537   .frame  $fp,48,$ra    # vars= 8, regs= 3/0, args= 16, extra= 8
538   .mask 0xd0000000,-8
539   .fmask  0x00000000,0
540   .set  noreorder
541   .cpload $t9
542   .set  reorder
543   subu  $sp,$sp,48
544   .cprestore 16
545   sw  $ra,40($sp)
546   sw  $fp,36($sp)
547   sw  $gp,32($sp)
548   move  $fp,$sp
549   sw  $a0,48($fp)
550   sw  $a1,52($fp)
551   sw  $zero,24($fp)
552 $L67:
553   lw  $v0,24($fp)
554   lw  $v1,52($fp)
555   slt $v0,$v0,$v1
556   bne $v0,$zero,$L70
557   b $L68
558 $L70:
559   lw  $v0,24($fp)
560   sll $v1,$v0,2
561   lw  $v0,48($fp)
562   addu  $v0,$v1,$v0
563   lw  $a0,0($v0)
564   la  $t9,free
565   jal $ra,$t9
566   lw  $v0,24($fp)
567   addu  $v0,$v0,1
568   sw  $v0,24($fp)
569   b $L67
570 $L68:
571   lw  $a0,48($fp)
572   la  $t9,free
573   jal $ra,$t9
574   move  $sp,$fp
575   lw  $ra,40($sp)
576   lw  $fp,36($sp)
577   addu  $sp,$sp,48
578   j $ra
579   .end  liberarMatriz
580   .size liberarMatriz, .-liberarMatriz
581   .rdata
582   .align  2
583 $LC16:
584   .ascii  "%ix%i \000"
585   .align  2
586 $LC17:
587   .ascii  "%g \000"
588   .align  2
589 $LC18:
590   .ascii  "\n\000"
```

```
591   .text
592   .align  2
593   .globl  multiplicar
594   .ent  multiplicar
595 multiplicar:
596   .frame  $fp,64,$ra    # vars= 24, regs= 3/0, args= 16, extra= 8
597   .mask 0xd0000000,-8
598   .fmask  0x00000000,0
599   .set  noreorder
600   .cpload $t9
601   .set  reorder
602   subu  $sp,$sp,64
603   .cprestore 16
604   sw  $ra,56($sp)
605   sw  $fp,52($sp)
606   sw  $gp,48($sp)
607   move  $fp,$sp
608   sw  $a0,64($fp)
609   sw  $a1,68($fp)
610   sw  $a2,72($fp)
611   sw  $a3,76($fp)
612   la  $a0,$LC16
613   lw  $a1,68($fp)
614   lw  $a2,80($fp)
615   la  $t9,printf
616   jal $ra,$t9
617   sw  $zero,24($fp)
618 $L72:
619   lw  $v0,24($fp)
620   lw  $v1,68($fp)
621   slt $v0,$v0,$v1
622   bne $v0,$zero,$L75
623   b $L73
624 $L75:
625   sw  $zero,28($fp)
626 $L76:
627   lw  $v0,28($fp)
628   lw  $v1,80($fp)
629   slt $v0,$v0,$v1
630   bne $v0,$zero,$L79
631   b $L74
632 $L79:
633   sw  $zero,40($fp)
634   sw  $zero,44($fp)
635   sw  $zero,32($fp)
636 $L80:
637   lw  $v0,32($fp)
638   lw  $v1,72($fp)
639   slt $v0,$v0,$v1
640   bne $v0,$zero,$L83
641   b $L81
642 $L83:
643   lw  $v0,24($fp)
644   sll $v1,$v0,2
645   lw  $v0,64($fp)
646   addu  $a0,$v1,$v0
647   lw  $v0,32($fp)
648   sll $v1,$v0,3
649   lw  $v0,0($a0)
650   addu  $a1,$v1,$v0
651   lw  $v0,32($fp)
652   sll $v1,$v0,2
653   lw  $v0,76($fp)
654   addu  $a0,$v1,$v0
655   lw  $v0,28($fp)
656   sll $v1,$v0,3
657   lw  $v0,0($a0)
658   addu  $v0,$v1,$v0
659   l.d $f2,0($a1)
660   l.d $f0,0($v0)
661   mul.d $f2,$f2,$f0
662   l.d $f0,40($fp)
663   add.d $f0,$f0,$f2
664   s.d $f0,40($fp)
```

```
665   lw  $v0,32($fp)
666   addu  $v0,$v0,1
667   sw  $v0,32($fp)
668   b $L80
669 $L81:
670   la  $a0,$LC17
671   lw  $a2,40($fp)
672   lw  $a3,44($fp)
673   la  $t9,printf
674   jal $ra,$t9
675   lw  $v0,28($fp)
676   addu  $v0,$v0,1
677   sw  $v0,28($fp)
678   b $L76
679 $L74:
680   lw  $v0,24($fp)
681   addu  $v0,$v0,1
682   sw  $v0,24($fp)
683   b $L72
684 $L73:
685   la  $a0,$LC18
686   la  $t9,printf
687   jal $ra,$t9
688   move  $sp,$fp
689   lw  $ra,56($sp)
690   lw  $fp,52($sp)
691   addu  $sp,$sp,64
692   j $ra
693   .end  multiplicar
694   .size multiplicar, .-multiplicar
695   .rdata
696   .align  2
697 $LC19:
698   .ascii  "%i %*c%i \000"
699   .align  2
700 $LC20:
701   .ascii  "Fallo al leer dimensiones\n\000"
702   .align  2
703 $LC21:
704   .ascii  "Fallo en malloc\n\000"
705   .align  2
706 $LC22:
707   .ascii  "Cantidad elementos distinta a dimensiones de matriz\n\000"
708   .align  2
709 $LC23:
710   .ascii  "Dimensiones no compatibles para multiplicar\n\000"
711   .text
712   .align  2
713   .globl  main
714   .ent  main
715 main:
716   .frame  $fp,88,$ra    # vars= 40, regs= 3/0, args= 24, extra= 8
717   .mask 0xd0000000,-8
718   .fmask  0x00000000,0
719   .set  noreorder
720   .cpload $t9
721   .set  reorder
722   subu  $sp,$sp,88
723   .cprestore 24
724   sw  $ra,80($sp)
725   sw  $fp,76($sp)
726   sw  $gp,72($sp)
727   move  $fp,$sp
728   sw  $a0,88($fp)
729   sw  $a1,92($fp)
730   lw  $a0,88($fp)
731   lw  $a1,92($fp)
732   la  $t9,parsearOpciones
733   jal $ra,$t9
734   addu  $v0,$fp,36
735   addu  $v1,$fp,40
736   la  $a0,$LC19
737   move  $a1,$v0
738   move  $a2,$v1
```

```
739   la   $t9,scanf
740   jal  $ra,$t9
741   sw   $v0,44($fp)
742 $L85:
743   lw   $v1,44($fp)
744   li   $v0,2      # 0x2
745   beq  $v1,$v0,$L88
746   la   $a0,__sF+176
747   la   $a1,$LC20
748   la   $t9,fprintf
749   jal  $ra,$t9
750   li   $v0,1      # 0x1
751   sw   $v0,64($fp)
752   b $L84
753 $L88:
754   lw   $a0,36($fp)
755   lw   $a1,40($fp)
756   la   $t9,alocarMatriz
757   jal  $ra,$t9
758   sw   $v0,32($fp)
759   lw   $v0,32($fp)
760   bne  $v0,$zero,$L89
761   la   $a0,__sF+176
762   la   $a1,$LC21
763   la   $t9,fprintf
764   jal  $ra,$t9
765   li   $v0,1      # 0x1
766   sw   $v0,64($fp)
767   b $L84
768 $L89:
769   lw   $a0,32($fp)
770   lw   $a1,36($fp)
771   lw   $a2,40($fp)
772   la   $t9,llenarMatriz
773   jal  $ra,$t9
774   sw   $v0,48($fp)
775   lw   $v0,48($fp)
776   beq  $v0,$zero,$L90
777   lw   $a0,32($fp)
778   lw   $a1,36($fp)
779   la   $t9,liberarMatriz
780   jal  $ra,$t9
781   la   $a0,__sF+176
782   la   $a1,$LC22
783   la   $t9,fprintf
784   jal  $ra,$t9
785   li   $v0,1      # 0x1
786   sw   $v0,64($fp)
787   b $L84
788 $L90:
789   addu   $v0,$fp,56
790   addu   $v1,$fp,60
791   la   $a0,$LC19
792   move   $a1,$v0
793   move   $a2,$v1
794   la   $t9,scanf
795   jal  $ra,$t9
796   sw   $v0,44($fp)
797   lw   $v1,44($fp)
798   li   $v0,2      # 0x2
799   beq  $v1,$v0,$L91
800   lw   $a0,32($fp)
801   lw   $a1,36($fp)
802   la   $t9,liberarMatriz
803   jal  $ra,$t9
804   la   $a0,__sF+176
805   la   $a1,$LC20
806   la   $t9,fprintf
807   jal  $ra,$t9
808   li   $v0,1      # 0x1
809   sw   $v0,64($fp)
810   b $L84
811 $L91:
812   lw   $a0,56($fp)
```

```
813   lw   $a1,60($fp)
814   la   $t9,alocarMatriz
815   jal  $ra,$t9
816   sw   $v0,52($fp)
817   lw   $v0,52($fp)
818   bne  $v0,$zero,$L92
819   lw   $a0,32($fp)
820   lw   $a1,36($fp)
821   la   $t9,liberarMatriz
822   jal  $ra,$t9
823   la   $a0,__sF+176
824   la   $a1,$LC21
825   la   $t9,fprintf
826   jal  $ra,$t9
827   li   $v0,1      # 0x1
828   sw   $v0,64($fp)
829   b $L84
830 $L92:
831   lw   $a0,52($fp)
832   lw   $a1,56($fp)
833   lw   $a2,60($fp)
834   la   $t9,llenarMatriz
835   jal  $ra,$t9
836   sw   $v0,48($fp)
837   lw   $v0,48($fp)
838   beq  $v0,$zero,$L93
839   lw   $a0,32($fp)
840   lw   $a1,36($fp)
841   la   $t9,liberarMatriz
842   jal  $ra,$t9
843   lw   $a0,52($fp)
844   lw   $a1,56($fp)
845   la   $t9,liberarMatriz
846   jal  $ra,$t9
847   la   $a0,__sF+176
848   la   $a1,$LC22
849   la   $t9,fprintf
850   jal  $ra,$t9
851   li   $v0,1      # 0x1
852   sw   $v0,64($fp)
853   b $L84
854 $L93:
855   lw   $v1,40($fp)
856   lw   $v0,56($fp)
857   bne  $v1,$v0,$L94
858   lw   $v0,60($fp)
859   sw   $v0,16($sp)
860   lw   $a0,32($fp)
861   lw   $a1,36($fp)
862   lw   $a2,40($fp)
863   lw   $a3,52($fp)
864   la   $t9,multiplicar
865   jal  $ra,$t9
866   lw   $a0,32($fp)
867   lw   $a1,36($fp)
868   la   $t9,liberarMatriz
869   jal  $ra,$t9
870   lw   $a0,52($fp)
871   lw   $a1,56($fp)
872   la   $t9,liberarMatriz
873   jal  $ra,$t9
874   b $L95
875 $L94:
876   lw   $a0,32($fp)
877   lw   $a1,36($fp)
878   la   $t9,liberarMatriz
879   jal  $ra,$t9
880   lw   $a0,52($fp)
881   lw   $a1,56($fp)
882   la   $t9,liberarMatriz
883   jal  $ra,$t9
884   la   $a0,__sF+176
885   la   $a1,$LC23
886   la   $t9,fprintf
```

```
887   jal $ra,$t9
888   li  $v0,1       # 0x1
889   sw  $v0,64($fp)
890   b $L84
891 $L95:
892   addu  $v0,$fp,36
893   addu  $v1,$fp,40
894   la  $a0,$LC19
895   move  $a1,$v0
896   move  $a2,$v1
897   la  $t9,scanf
898   jal $ra,$t9
899   sw  $v0,44($fp)
900   lw  $v1,44($fp)
901   li  $v0,-1        # 0xffffffffffffffff
902   bne $v1,$v0,$L85
903   sw  $zero,64($fp)
904 $L84:
905   lw  $v0,64($fp)
906   move  $sp,$fp
907   lw  $ra,80($sp)
908   lw  $fp,76($sp)
909   addu  $sp,$sp,88
910   j $ra
911   .end  main
912   .size main, .-main
913   .ident  "GCC: (GNU) 3.3.3 (NetBSD nb3 20040520)"
```

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.