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

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 char* getLine() {
79   char buff[100];
80   if (!fgets(buff,100, stdin)) {
81       return NULL;
82   }
83   size_t l = strLength(buff);
84   if (buff[l-1] == '\n')  {
85     if (l > 1)  {
86       buff[l-1] = 0;
87       }
88     char* concat;
89     concat = (char*)malloc(l*sizeof(char));
90     int i;
91     for (i=0;i<l;i++)
92       concat[i] = buff[i];
93     return concat;
94   } else {
95     //EN ESTE CASO, EL BUFFER ES MENOR A LA CANTIDAD DE CARACTERES DE LA LINEA
96     //DEBO ARMAR UN BUFFER MAS GRANDE Y SEGUIR LEYENDO HASTA ENCONTRAR
97     //EL FIN DE LINEA
98     /*
99     char buff2[100];
100    char* concat;
101    concat = NULL;
102    while (fgets(buff2,100,stdin)) {
103      free(concat);
104      size_t l2 = strLength(buff2);
105      concat = (char*)malloc((l+l2+1)*sizeof(char));
106      int i;
107      for (i = 0; i < l; i++)
108          concat[i] = buff[i];
109      for (i = 0; i < l2; i++){
110          int j = i + l;
111          concat[j] = buff2[i];
112          }
113      concat[l+l2] = 0;
114    }
115    return concat;
116    */
117  }
118  return NULL;
119 }
120
121 int llenarMatriz(double** matriz, int fila, int columna) {
122     int i;
123     int j;
124     char c;
125     int cantidadElementos = 0;
126     i = 0;
127     j = 0;
128     bool exito = true;
129     double valor;
130     int flag;
131     while (exito && i<fila) {
132       flag = scanf("%lf%c",&valor,&c);
133       if (flag != EOF && flag == 2) {
134         //printf("Leo: %lf y %c\n",valor,c);
135         matriz[i][j] = valor;
136         cantidadElementos++;
137         if (j==columna-1) {
138           j=0;
139           i++;
140         } else {
141           j++;
142         }
143       } else {
```

```
144         exito = false;
145       }
146     }
147     if (cantidadElementos != ((fila)*(columna)) || (c!='\n')) {
148       return EXIT_FAILURE;
149     }
150   return EXIT_SUCCESS;
151 }
152
153 void liberarMatriz(double** matriz, int fila) {
154   int i;
155   for(i=0;i<fila;++i) {
156     free(matriz[i]);
157   }
158   free(matriz);
159 }
160 void multiplicar(double** matriz1, int fila1, int columna1, double** matriz2, int columna2) {
161   int i;
162   int j;
163   int k;
164   double accum;
165   printf("%dx%d ",fila1,columna2);
166   for(i=0;i<fila1;i++) {
167     for(j=0;j<columna2;j++) {
168       accum = 0;
169       for(k=0;k<columna1;k++) {
170         accum = accum + (matriz1[i][k] * matriz2[k][j]);
171       }
172       printf("%g ",accum);
173     }
174   }
175   printf("\n");
176 }
177 int main(int argc, char *argv[]) {
178   parsearOpciones(argc,argv);
179   //Construyo la primera matriz
180   double** matriz1;
181   int fila1;
182   int columna1;
183   int cant;
184   cant =  scanf("%d %*c %d ",&fila1,&columna1);
185   do {
186     if (cant != 2) {
187       fprintf(stderr, "Fallo al leer dimensiones\n");
188       return EXIT_FAILURE;
189     }
190     matriz1 = alocarMatriz(fila1,columna1);
191     if (!matriz1) {
192       fprintf(stderr, "Fallo en malloc\n");
193       return EXIT_FAILURE;
194     }
195     int llenar;
196     llenar = llenarMatriz(matriz1,fila1,columna1);
197     if (llenar) {
198       liberarMatriz(matriz1,fila1);
199       fprintf(stderr, "Cantidad elementos distinta a dimensiones de matriz\n");
200       return EXIT_FAILURE;
201     }
202     //Repito para segunda matriz
203     double** matriz2;
204     int fila2;
205     int columna2;
206     cant = scanf("%d %*c %d ",&fila2,&columna2);
207     if (cant != 2) {
208       liberarMatriz(matriz1,fila1);
209       fprintf(stderr, "Fallo al leer dimensiones\n");
210       return EXIT_FAILURE;
211     }
212     matriz2 = alocarMatriz(fila2,columna2);
213     if (!matriz2) {
214       liberarMatriz(matriz1,fila1);
215       fprintf(stderr, "Fallo en malloc\n");
216       return EXIT_FAILURE;
217     }
```

```
218     llenar =  llenarMatriz(matriz2,fila2,columna2);
219     if (llenar) {
220       liberarMatriz(matriz1,fila1);
221       liberarMatriz(matriz2,fila2);
222       fprintf(stderr, "Cantidad elementos distinta a dimensiones de matriz\n");
223       return EXIT_FAILURE;
224     }
225     if(columna1 == fila2) {
226       //Multiplicar
227       multiplicar(matriz1,fila1,columna1,matriz2,columna2);
228       liberarMatriz(matriz1,fila1);
229       liberarMatriz(matriz2,fila2);
230     } else {
231       liberarMatriz(matriz1,fila1);
232       liberarMatriz(matriz2,fila2);
233       fprintf(stderr, "Dimensiones no compatibles para multiplicar\n");
234       return EXIT_FAILURE;
235     }
236     cant =  scanf("%i %*c %i ",&fila1,&columna1);
237   } while(cant != EOF);
238     //Repetir
239     return EXIT_SUCCESS;
240 }
```

5.2. Codigo MIPS32 generado por el compilador

```
... (Codigo de MIPS32)
```

6. Enunciado

Acá iría el enunciado, no se si se anexa aparte o lo agregamos acá

7. Conclusiones

... Conclusiones ...