
66.20 - Organización de Computadoras
Trabajo Práctico 1 - Conjunto de instrucciones MIPS

Nicolás Alvarez, Padrón 93503
Nicolás Fernandez, Padrón 88599
Andrew Parlane

26 de abril de 2018

# Índice

# 1. Resumen

El objetivo de este trabajo se basa en aplicar los conocimientos aprendidos en clase acerca de las instrucciones MIPS32 y el concepto de ABI. Para eso desarrollaremos un programa en C que utilizará una función desarrollada en lenguaje assembler MIPS32.

# 2. Introducción

El programa que desarrollaremos se encargará de tomar una matriz de números enteros y devolverá su matriz transpuesta. El programa recibirá como argumento el nombre del archivo donde está especificada la matriz, y dará el resultado por stdout o lo escribirá en un archivo, según las opciones de entrada. De haber errores, los mensajes de error saldrán exclusivamente por stderr.

# 3. Compilacion y ejecución del programa

Esta sección describe los mecanismos para la compilación y ejecución del programa, tanto en entorno MIPS como en Linux. En una primera etapa compilamos y trabajamos en Linux para poder realizar las pruebas. Adicionalmente a esto utilizamos Valgrind para chequear que no tengamos problemas de leaks o free en nuestro código.

Para compilar manualmente:

- Estando parados en la carpeta donde se encuentran los archivos fuente, ejecutamos el siguiente comando:

  ```
  gcc -Wall -o tp1 *.c
  ```

Para compilar usando el Makefile:

- Estando parados en la carpeta donde se encuentran los archivos fuente, ejecutamos el siguiente comando:

  ```
  host# make
  ```

  Y para la machina de MIPS:

  ```
  host# gmake
  ```

Para ejecutar:

- Estando parados en la carpeta donde se encuentran el archivo ejecutable, corremos para cada archivo o fragmento de prueba lo siguiente:

  ```
  ./tp1 [options] file
  ```

- Si se quiere ver el help, que especifica las opciones disponibles al momento de invocar el programa:

  ```
  ./tp1 -h
  ```

- Si se quiere ver la versión del programa:

  ```
  ./tp1 -V
  ```

# 4. Implementación

## 4.1. Implementación en C

El trabajo se estructuró en tres archivos, uno que poseerá la mayor parte del programa (procesamiento de argumentos, usage, etc.) y lo otros dos la función ***transponer*** en código C y en código assembler MIPS32:

- **main.c** : Define el proceso principal de ejecución , la validación de los parámetros pasados al programa y además los métodos para parseo y salida del programa.
- **transpose.c** : Define la función transponer en código c, que recibe una matriz y devuelve su transpuesta.

- **transpose.S** : Define la función transponer en código assembler, que recibe una matriz y devuelve su transpuesta.

A continuación enumeramos las funciones definidas en el programa que se usarán luego de la verificación y validación de los parámetros de entrada:

- **usage**
  parámetros: FILE *stream
  const char *nuestroNombre

  descripción: función que muestra el help de la aplicación.

- **leerLongLong**
  parámetros: FILE *f
  long long *ll
  bool *OK
  bool *eof
  bool *newLine

  descripción: función que lee un entero de la matriz.

- **leerLinea**
  parámetros: FILE *f
  long long *data
  uint columnasEsperados
  bool *eof

  descripción: función que lee una línea completa de la matriz.

- **leerEntrada**
  parámetros: const char *archivo
  uint *filas
  uint *columnas

  descripción: función lee el archivo de entrada e inicia el procesamiento de la matriz.

- **transponer**
  parámetros: unsigned int *filas
  unsigned int *filas
  uint *columnas
  long long *entrada
  long long *salida

  descripción: función lee una matriz y devuelva su matriz transpuesta.

# 5. Pruebas

Realizamos las pruebas en GXEmul para cada uno de los archivos pedidos.

- **matrix1**
- **matrix2**
- **matrix3**

```
$ ./tp1 -o - pruebas/matrix1
7 1
1
2
3
4
5
6
7

$ ./tp1 -o - pruebas/matrix2
Not enough entries on line. Expecting 5, found 4

$ ./tp1 -o - pruebas/matrix3
Found invalid character .
```

También se realizó unas pruebas con otros archivos para detectar otros casos posibles en el archivo de entrada.

```
$ cat matrix_tabs
4   3
1   2   3
                4           5           6
7 8 9
10                                          11 12

$ ./tp1 -o - pruebas/matrix_tabs
3 4
1 4 7 10
2 5 8 11
3 6 9 12

$ cat matrix_negativo
4 2
2 1
0 -1
-2 -3
-4 -5

$ ./tp1 -o - pruebas/matrix_tabs
2 4
2 0 -2 -4
1 -1 -3 -5

$ cat matrix_long_long
5 4
9223372036854775807 0 1234567891011121314 1
1516171819202122232 4252627282930313233 2 3
4 3435363738394041424 3444546474849505152 6
5354555657585960616 2636465666768697071 7273747576777787980 8182838485868788899
0 0 0 0

$ ./tp1 -o - pruebas/matrix_long_long
4 5
9223372036854775807 1516171819202122232 4 5354555657585960616 0
0 4252627282930313233 3435363738394041424 2636465666768697071 0
1234567891011121314 2 3444546474849505152 7273747576777787980 0
1 3 6 8182838485868788899 0
```

Se incluirán en la entrega más archivos que fueron usados para probar la robustez del programa.

# 6. Diagrama del stack del programa

A continuación se podrá ver el diagrama de cómo quedaría el stack cuando la función main llama a la función transponer. Asumimos que la función main usará algunos valores temporales por lo cual decidimos dejar reservado la sección de LTA del main con 16 bytes para las variables temporales ***nuestroNombre***, ***oArchivo***, ***option_index*** y ***c***

```
#   +---------------+   |------------MAIN------------
# 52    |       |   \
#   +---------------+   |
# 48    |   ra  |   |
#   +---------------+   |   SRA MAIN
# 44    |   fp  |   |
#   +---------------+   |
# 40    |   gp  |   /
#   +---------------+
# 36    |   c   |   \
#   +---------------+   |
# 32 |option_index| |
#   +---------------+   |   LTA MAIN
# 28   |oArchivo|   |
#   +---------------+   |
# 24 |nuestroNombre|/
#   +---------------+
# 20    |   a3  |   \
#   +---------------+   |
# 16    |   a2  |   |
#   +---------------+   |   ABA MAIN
# 12    |   a1  |   |
#   +---------------+   |
# 08    |   a0  |   /
#   +---------------+   |----------TRANSPONER----------
# 04    |   fp  |   \
#   +---------------+   |   SRA TRANSPONER
# 00    |   gp  |   /
#   +---------------+   |
```

# 7. Conclusiones

El desarrollo de este trabajo nos permitió llevar a la práctica los conocimientos adquiridos acerca de la estructura MIPS32. Debimos respetar la ABI de esta arquitectura, respetando los tamaños de las distintas secciones: SRA (Saved Register Area), LTA (Local and Temp Area) y ABA (Arg Building Area).

Para otorgar portabilidad a esta arquitectura desarrollamos la función transponer en lenguaje assembler MIPS32. Pudimos comprabar que fue un éxito al realizar la compilación de programa con la función ***transponer*** desarrollada en assembler en el archivo ***transpose.S***

Una observación que se puede apreciar en el diagrama del stack es que la función transponer posee un stack de 8 bytes ya que es una funcion *hoja* por lo cual no tendrá una ABA ni una LTA y no se deberá guardar el registro RA, mientras que el stack del main mide 48 bytes ya que reserva espacio de ABA para los argumentos que se usarán en la función de transposición, y como no es una función *hoja* guarda algunas variables temporales en la LTA

# 8. Código C

## 8.1. main.c

```c
#include <stdio.h>
#include <stdlib.h>
#include <libgen.h>
```

```c
#include <getopt.h>
#include <ctype.h>
#include <string.h>
#include <stdint.h>
#include <stdbool.h>
#include <errno.h>

#define MAJOR_VERSION   0
#define MINOR_VERSION   1

// declaracion adelante
// transponer es en transponer.c o transponer.s
extern int transponer(unsigned int filas,
                      unsigned int columnas,
                      long long *entrada,
                      long long *salida);

static const struct option long_options[] =
{
    {"help",    no_argument,        0, 'h' },
    {"version", no_argument,        0, 'V' },
    {"output",  required_argument,  0, 'o' },
    {0,         0,                  0,  0  }
};

static void usage(FILE *stream, const char *nuestroNombre)
{
    fprintf(stream,
            "Usage:\n"
            "  %s -h\n"
            "  %s -V\n"
            "  %s [options] filename\n"
            "Options:\n"
            "  -h, --help Prints usage information.\n"
            "  -V, --version Prints version information.\n"
            "  -o, --output Path to output file.\n"
            "\n"
            "Examples:\n"
            "  %s -o - mymatrix\n",
            nuestroNombre, nuestroNombre, nuestroNombre, nuestroNombre);
    // necesitamos usar nuestroNombre, nuestroNombre, nuestroNombre, nuestroNombre
    // porque no soportamos %1$s
}

// lea caracter por caracter deshaciendo whitespace
// hasta encontrar [0-9-]. Despues comenzar a leer numeros
// [0-9]. Para cuando obtenemos EOF, \n, \r, ' ', \t.
// Es un error si encontramos algun otro caracter.
// devolver no 0 si hay un error
// *OK = 1 -> hay un integer valido en *ll
// *eof = 1 -> no hay mas a leer
// *newLine = 1 -> encontramos nueva linea
bool leerLongLong(FILE *f, long long *ll, bool *OK, bool *eof, bool *newLine)
{
    *OK = false;
    *eof = false;
    *newLine = false;

    // soportamos signed 64 bits:
    // max = 0x7FFF_FFFF_FFFF_FFFF =  9223372036854775807
    // min = 0x8000_0000_0000_0000 = -9223372036854775808
    // asi max input legal es 20 caracters +1 por NULL terminator
#define MAX_CHARS 20
    char buff[MAX_CHARS + 1];
    uint idx = 0;

    bool comenzandoLeerInt = false;
    while (1)
    {
        int res = fgetc(f);
        if (res == EOF)
        {
            *eof = true;
```

```
77              // si tenemos algo en buff, convertimos ahora
78              if (*OK)
79              {
80                  buff[idx] = '\0';
81                  *ll = strtoll(buff, NULL, 10);
82                  if (errno != 0)
83                  {
84                      fprintf(stderr, "Failed to convert %s to long long, error: %s\n", buff,
    strerror(errno));
85                      return false;
86                  }
87                  return true;
88              }
89              else if (comenzandoLeerInt)
90              {
91                  // solo podriamos estar aqui si leemos
92                  // un '-' y despues nada, eso es un error
93                  fprintf(stderr, "Found invalid entry \"-\"\n");
94                  return false;
95              }
96              else
97              {
98                  // eof but no error
99                  return true;
100             }
101         }
102
103         char c = (char)res;
104         if (c == '\r' || c == '\n')
105         {
106             *newLine = true;
107         }
108
109         if (!comenzandoLeerInt)
110         {
111             // Todavia no cemenzamos a leer el int
112             if (c == ' ' || c == '\t')
113             {
114                 // ignoramos
115                 continue;
116             }
117             else if (*newLine)
118             {
119                 // nuevo linea, terminamos.
120                 return true;
121             }
122             else if (c >= '0' && c <= '9')
123             {
124                 // valido
125                 buff[idx++] = c;
126                 comenzandoLeerInt = true;
127                 *OK = true;
128             }
129             else if (c == '-')
130             {
131                 // tambien valido pero el int todavia no es OK
132                 // porque necesitamos un niumero despues de un -
133                 buff[idx++] = c;
134                 comenzandoLeerInt = true;
135             }
136             else
137             {
138                 // error
139                 fprintf(stderr, "Found invalid character %c\n", c);
140                 return false;
141             }
142         }
143         else
144         {
145             // ya estamos leyendo data
146             if (c == ' ' || c == '\t' ||
147                 *newLine)
148             {
```

```
149                 // terminamos
150                 if (*OK)
151                 {
152                     buff[idx] = '\0';
153                     *ll = strtoll(buff, NULL, 10);
154                     if (errno != 0)
155                     {
156                         fprintf(stderr, "Failed to convert %s to long long, error: %s\n", buff
    , strerror(errno));
157                         return false;
158                     }
159                     return true;
160                 }
161                 else
162                 {
163                     // solo podriamos estar aqui si leemos
164                     // un '-' y despues nada, eso es un error
165                     fprintf(stderr, "Found invalid entry \"-\"\n");
166                     return false;
167                 }
168             }
169             else if (c == '-')
170             {
171                 // un - aqui no es valido porque estamos en el medio
172                 // de un int.
173                 fprintf(stderr, "Found \"-\" in the middle of an integer\n");
174                 return false;
175             }
176             else if (c >= '0' && c <= '9')
177             {
178                 // valido
179                 if (idx >= MAX_CHARS)
180                 {
181                     fprintf(stderr, "Integer read was too large to fit into a long long\n");
182                     return false;
183                 }
184                 buff[idx++] = c;
185                 *OK = true;
186             }
187             else
188             {
189                 // error
190                 fprintf(stderr, "Found invalid character %c\n", c);
191                 return false;
192             }
193         }
194     }
195 }
196
197 bool leerLinea(FILE *f, long long *data, uint columnasEsperados, bool *eof)
198 {
199     uint32_t count = 0;
200     while (1)
201     {
202         bool OK;
203         bool newLine;
204         long long ll;
205         if (!leerLongLong(f, &ll, &OK, eof, &newLine))
206         {
207             // error
208             return false;
209         }
210
211         if (OK)
212         {
213             // leemos un integer
214             if (count >= columnasEsperados)
215             {
216                 // error - hay mas columnas de las esperadas
217                 fprintf(stderr, "Too many entries on line. Expecting %u\n", columnasEsperados)
    ;
218                 return false;
219             }
```

```
220
221             data[count] = ll;
222             count++;
223         }
224
225         if (*eof)
226         {
227             if (count != columnasEsperados)
228             {
229                 // error - hay menos columnas de las esperadas
230                 fprintf(stderr, "Not enough entries on line. Expecting %u, found %u\n",
    columnasEsperados, count);
231                 return false;
232             }
233             else
234             {
235                 return true;
236             }
237         }
238
239         if (newLine)
240         {
241             if (count == 0) // permitimos newLines antes de data comenzando
242             {
243                 continue;
244             }
245             else if (count != columnasEsperados)
246             {
247                 // error - hay menos columnas de las esperadas.
248                 fprintf(stderr, "Not enough entries on line. Expecting %u, found %u\n",
    columnasEsperados, count);
249                 return false;
250             }
251             else
252             {
253                 return true;
254             }
255         }
256     }
257     return true;
258 }
259
260 long long *leerEntrada(const char *archivo, uint *filas, uint *columnas)
261 {
262     FILE *f = fopen(archivo, "r");
263     if (f == NULL)
264     {
265         fprintf(stderr, "%s: No such file or directory\n", archivo);
266         return NULL;
267     }
268
269     long long primerLinea[2];
270
271     bool eof;
272     if (!leerLinea(f, primerLinea, 2, &eof))
273     {
274         fclose(f);
275         return NULL;
276     }
277
278     long long *llFilas = &primerLinea[0];
279     long long *llColumnas = &primerLinea[1];
280
281     // Validar filas y columnas
282     // no pueden ser menor a cero
283     // ni mas grande que 0xFFFFFFFF
284     if (*llFilas < 0 || *llColumnas < 0 ||
285         *llFilas > 0xFFFFFFFF ||
286         *llColumnas > 0xFFFFFFFF)
287     {
288         fprintf(stderr, "Invalid number of rows / columns\n");
289         fclose(f);
290         return NULL;
```

```
291     }
292
293     *filas = *(uint *)llFilas;
294     *columnas = *(uint *)llColumnas;
295
296     // numero de elementos = filas * columnas
297     // cada uno es un long long, asi:
298     long long *entrada = malloc(*filas * *columnas * sizeof(long long));
299     if (entrada == NULL)
300     {
301         fprintf(stderr, "Failed to malloc %u bytes\n", (unsigned int)(*filas * *columnas *
    sizeof(long long)));
302         fclose(f);
303         return NULL;
304     }
305
306     uint i;
307     for (i = 0; i < *filas; i++)
308     {
309         if (!leerLinea(f, &entrada[i * *columnas], *columnas, &eof))
310         {
311             fclose(f);
312             free(entrada);
313             return NULL;
314         }
315     }
316
317     // deberia estar todo, comprobar que no hay mas data
318     while (!eof)
319     {
320         if (!leerLinea(f, NULL, 0, &eof))
321         {
322             fclose(f);
323             free(entrada);
324             return NULL;
325         }
326     }
327
328     fclose(f);
329     return entrada;
330 }
331
332 bool escribirSalida(const char *archivo, uint filas, uint columnas, long long *salida)
333 {
334     FILE *f;
335
336     if (archivo == NULL)
337     {
338         // stdout
339         f = stdout;
340     }
341     else
342     {
343         // archivo
344         f = fopen(archivo, "w");
345         if (f == NULL)
346         {
347             fprintf(stderr, "Failed to open %s for writing\n", archivo);
348             return NULL;
349         }
350     }
351
352     fprintf(f, "%u %u\n", filas, columnas);
353     uint i;
354     for (i = 0; i < filas; i++)
355     {
356         uint c;
357         for (c = 0; c < columnas; c++)
358         {
359             fprintf(f, "%lld ", salida[(i * columnas) + c]);
360         }
361         fprintf(f, "\n");
362     }
```

```
363
364     if (archivo != NULL)
365     {
366         fclose(f);
367     }
368
369     return true;
370 }
371
372 int main(int argc, char **argv)
373 {
374     // usamos argv[0] como el nombre del aplicacion
375     // pero solo queremos el archivo, no la ruta
376     const char *nuestroNombre = basename(argv[0]);
377
378     // escribir la salida a un archivo si vemos -o (y el argumento no es -)
379     const char *oArchivo = NULL;
380
381     // clear errors
382     opterr = 0;
383
384     // parse short options
385     while (1)
386     {
387         // obtener el siguiente argumento
388         int option_index = 0;
389         int c = getopt_long(argc, argv, "hVo:", long_options, &option_index);
390
391         if (c == -1)
392         {
393             // no hay mas
394             break;
395         }
396
397         switch (c)
398         {
399             case 'h':
400             {
401                 usage(stdout, nuestroNombre);
402                 // no seguimos despues de -h
403                 return 0;
404             }
405             case 'V':
406             {
407                 printf("%s: Version %u.%u\n", nuestroNombre, MAJOR_VERSION, MINOR_VERSION);
408                 // no siguimos desupes de -V
409                 return 0;
410             }
411             case 'o':
412             {
413                 // si vemos "-o -" la salida es stdout
414                 // si no, la salida es el archivo en optarg
415                 if (strcmp(optarg, "-") != 0)
416                 {
417                     oArchivo = optarg;
418                 }
419                 break;
420             }
421             case '?':
422             {
423                 if (optopt == 'o')
424                 {
425                     fprintf(stderr, "Option '-%c' requires an argument.\n\n", optopt);
426                 }
427                 else if (isprint(optopt))
428                 {
429                     // es un argumento, pero no es uno que esperamos
430                     fprintf (stderr, "Unknown option '-%c'.\n\n", optopt);
431                 }
432                 else
433                 {
434                     // solo muestra el usage
435                 }
```

```
                usage(stderr, nuestroNombre);
                return 1;
            }
            default:
            {
                usage(stderr, nuestroNombre);
                return 1;
            }
        }
    }

    if (optind == argc)
    {
        fprintf(stderr, "filename is required\n\n");
        usage(stderr, nuestroNombre);
        return 1;
    }

    if ((optind + 1) != argc)
    {
        fprintf(stderr, "Too many arguments\n\n");
        usage(stderr, nuestroNombre);
        return 1;
    }

    // leer archivo
    uint filas;
    uint columnas;
    long long *entrada = leerEntrada(argv[optind], &filas, &columnas);
    if (entrada == NULL)
    {
        // falla, leerArchivo escribi el error
        return 1;
    }

    // malloc la salida
    long long *salida = malloc(filas * columnas * sizeof(long long));
    if (salida == NULL)
    {
        fprintf(stderr, "Failed to allocate %u bytes for output\n", (unsigned int)(filas *
    columnas * sizeof(long long)));
        free(entrada);
        return 1;
    }

    // transponer
    if (transponer(filas, columnas, entrada, salida) != 0)
    {
        fprintf(stderr, "Failed to transpose the matrix\n");
        free(entrada);
        free(salida);
        return 1;
    }

    // escribir el resultado
    bool res = escribirSalida(oArchivo, columnas, filas, salida);
    free(entrada);
    free(salida);

    return res ? 0 : 1;
}
```

## 8.2. transpose.c

```
#include <stdio.h>
#include <stdint.h>
#include <stdlib.h>

int transponer(unsigned int filas, unsigned int columnas, long long *entrada, long long *
    salida)
{
    uint f;
    for (f = 0; f < filas; f++)
```

```
    {
        uint c;
        for (c = 0; c < columnas; c++)
        {
            salida[(c * filas) + f] = entrada[(f * columnas) + c];
        }
    }

    return 0;
}
```

# 9. Código Assembler

## 9.1. transpose.S

```
    #   +---------------+   |------------MAIN------------
    # 52    |       |   \
    #   +---------------+   |
    # 48    |   ra  |   |
    #   +---------------+   |   SRA MAIN
    # 44    |   fp  |   |
    #   +---------------+   |
    # 40    |   gp  |   /
    #   +---------------+
    # 36    |   c   |   \
    #   +---------------+   |
    # 32 |option_index| |
    #   +---------------+   |   LTA MAIN
    # 28   |oArchivo|   |
    #   +---------------+   |
    # 24 |nuestroNombre|/
    #   +---------------+
    # 20    |   a3  |   \
    #   +---------------+   |
    # 16    |   a2  |   |
    #   +---------------+   |   ABA MAIN
    # 12    |   a1  |   |
    #   +---------------+   |
    # 08    |   a0  |   /
    #   +---------------+   |----------TRANSPONER----------
    # 04    |   fp  |   \
    #   +---------------+   |   SRA TRANSPONER
    # 00    |   gp  |   /
    #   +---------------+   |


#include <mips/regdef.h>

    .text
    .align  2
    .globl  transponer
    .ent    transponer

# int transponer(unsigned int filas,
#                unsigned int columnas,
#                long long *entrada,
#                long long *salida)

transponer:
    subu    sp, sp, 8
    sw      gp, 0(sp)
    sw      $fp, 4(sp)

    # a0 = filas
    # a1 = columnas
    # a2 = &entrada[0]
    # a3 = &salida[0]

    move    v0, zero            # siempre devolvemos 0 no hay errores posibles

    beqz    a0, fin             # if (filas == 0) return 0;
    beqz    a1, fin             # if (columnas == 0) return 0;
```

```
    move    t0, zero            # uint f = 0;

filaLoop:                       # do {
    move    t1, zero            #   uint c = 0;

columnaLoop:                    #   do {
    ld      t2, 0(a2)           #     (t2,t3) = *entrada;
    addu    a2, a2, 8           #     entrada++;

    mul     t4, t1, a0          #     t4 = (c * filas);
    addu    t4, t4, t0          #     t4 += f;
    mul     t4, t4, 8           #     t4 = offset en salida
    addu    t4, t4, a3          #     t4 = &salida[(c * filas) + f]
    sd      t2, 0(t4)           #     salida[(c * filas) + f] = (t2,t3)

    addu    t1, t1, 1           #     c++;
    bne     t1, a1, columnaLoop #   } while (c != columnas)

    addu    t0, t0, 1           #   f++;
    bne     t0, a0, filaLoop    # } while (f != filas)

fin:
    lw      gp, 0(sp)
    lw      $fp, 4(sp)
    addu    sp, sp, 8
    jr      ra

    .end transponer
```

# 10. Código Makefile

## 10.1. Makefile

```
C_TARGET = tp1_c
ASM_TARGET = tp1_asm

MACHINE = $(shell uname -m)

LIBS =
CC = gcc
C_FLAGS = -Wall -g
ASM_FLAGS = -Wall -g

default: $(C_TARGET)
all: default

C_OBJECTS = main.c.o \
                    transpose.c.o

ASM_OBJECTS = main.c.o \
                          transpose.S.o

HEADERS = $(wildcard *.h)

ifeq ($(MACHINE), pmax)

define HACE_PRUEBA_ASM
        @echo Probando codigo ASM con pruebas/$(strip $(1))
        @./pruebaScript.sh $(ASM_TARGET) $(strip $(1)) $(2)
endef

else

define HACE_PRUEBA_ASM
        @#Hace nada porque no estamos pmax
endef

endif

# Macro con dos argumentos
#  1) El nombre de archivo a probar en pruebas/
```

```
#  2) Codigo de salida esperado
#
# Si el codigo de salida es igual al esperada,
# combrobamos la salida con un archivo que
# tiene el mismo nombre del input en resultados/

define HACE_PRUEBA

        @echo Probando codigo C con pruebas/$(strip $(1))
        @./pruebaScript.sh $(C_TARGET) $(strip $(1)) $(2)
        $(call HACE_PRUEBA_ASM, $(1), $(2))
endef


%.c.o: %.c $(HEADERS)
        $(CC) $(C_FLAGS) -c $< -o $@

%.S.o: %.S $(HEADERS)
        $(CC) $(ASM_FLAGS) -c $< -o $@

$(C_TARGET): $(C_OBJECTS)
        $(CC) $(C_OBJECTS) $(C_FLAGS) $(LIBS) -o $@

$(ASM_TARGET): $(ASM_OBJECTS)
        $(CC) $(ASM_OBJECTS) $(ASM_FLAGS) $(LIBS) -o $@

C: $(C_TARGET)

ifeq ($(MACHINE), pmax)

ASM: $(ASM_TARGET)

else

ASM:

endif

c: C

asm: ASM

prueba: C ASM
        @# Primero el basico
        -$(call HACE_PRUEBA, matrix1, 0)
        @# Con finales de lineas diferentes
        -$(call HACE_PRUEBA, matrix_crlf, 0)
        -$(call HACE_PRUEBA, matrix_cr_only, 0)
        -$(call HACE_PRUEBA, matrix_lf_only, 0)
        @# Espacio blanco extra
        -$(call HACE_PRUEBA, matrix_filas_blancas, 0)
        -$(call HACE_PRUEBA, matrix_tabs, 0)
        @# Numeros negativos o grandes, pero en el rango de signed long long
        -$(call HACE_PRUEBA, matrix_negativo, 0)
        -$(call HACE_PRUEBA, matrix_long_long, 0)
        @# Invalidos
        -$(call HACE_PRUEBA, matrix2, 1)
        -$(call HACE_PRUEBA, matrix3, 1)
        -$(call HACE_PRUEBA, matrix_filas_negativos, 1)
        -$(call HACE_PRUEBA, matrix_demasiado_columnos, 1)
        -$(call HACE_PRUEBA, matrix_demasiado_filas, 1)
        -$(call HACE_PRUEBA, matrix_demasiado_largo_int, 1)
        -$(call HACE_PRUEBA, matrix_demasiado_largo_int2, 1)
        -$(call HACE_PRUEBA, matrix_demasiado_negativo_int, 1)
        -$(call HACE_PRUEBA, matrix_demasiado_negativo_int2, 1)
        -$(call HACE_PRUEBA, matrix_no_suficiente_filas, 1)
        -$(call HACE_PRUEBA, archivo_que_no_existe, 1)

clean:
        -rm -f *.o
        -rm -f $(C_TARGET)
        -rm -f $(ASM_TARGET)
        -rm -f salida stdout stderr
```

```
.PHONY: default all C c ASM asm clean
```

# 11. Enunciado

**Ver hojas anexadas*