

66.20 Organización de Computadoras: Trabajo Práctico 1

Eirik Harald Lund, *Padrón Nro. 103081*,
`eirikharald@hotmail.com`
Julian Quino, *Padrón Nro. 94224*,
`julian.quino2@gmail.com`

Grupo Nro. ? - 1er. Cuatrimestre de 2018
66.20 Organización de Computadoras
Facultad de Ingeniería, Universidad de Buenos Aires

09/04/2018

Resumen

Usando el lenguaje C, desarrollamos un programa que devuelve el traspuesto de una matriz de entrada. La salida es escribe a un archivo. Escribimos también un programa en la versión del lenguaje Assembly de MIPS32. Además de la matriz, la primera línea del archivo debe tener dos números que dan el tamaño de la matriz.

1. Documentación

El programa necesita abrir y leer un archivo. Después de hacer la transposición, el programa debe o imprimir el resultado por *stdout* o escribirlo a un archivo, dado por los argumentos. Todo eso se puede hacer con la biblioteca standard de C.

Los pasos del desarrollo eran:

1. Leer los argumentos y abrir el archivo
2. Leyendo el archivo, guardar las dimensiones de la matriz y recorrerla
3. Guardar la matriz en forma traspuesta

4. O imprimir por *stdout* o guardarla en un archivo
5. Durante todo, observar si el archivo de entrada tiene la forma correcta. Si no, manejarlo como un error

Los primeros dos pasos son bastante standard cuando se trabaja con C. El recorrido de la matriz implica comparar los datos leídos con la forma esperada. Si se encuentra algo erróneo, necesita terminar en una manera controlada.

Mientras se recorre el archivo, guarda los datos que se encuentren en la forma traspuesta. Entonces, cuando se termine, tiene la matriz nueva y puede continuar.

El cuarto paso se realiza usando métodos standard de C.

Finalmente, el quinto paso es parte de todo. El programa debía estar escrito para un formate específico de matrices, así que es importante que tenga verificaciones de los datos encontrados.

2. Comandos

Para compilar el programa se usa un *makefile*. Así que, en el directorio con los archivos del código fuente, el comando es simplemente:

```
$ make
```

Que es equivalente a:

```
$ gcc -Wall -g -o tp1 tp1-orga.c
$ gcc -Wall -g -o tp1-MIPS trasponer.S tp1-orga-MIPS.c
```

Esto da dos programas: *tp1* y *tp1-MIPS*. El primero sólo usa C, mientras la segunda usa la función de la transposición escrita en MIPS32 Assembly.

Para usar el programa, se puede usar un comando así:

```
$ ./tp1 -o - matriz
```

Esto imprimirá la matriz de salida por *stdout*, sin escribirla a un archivo. Si se quiere usar un archivo, puede usar un comando así:

```
$ ./tp1 -o salida matriz
```

Se puede por supuesto también reemplazar *tp1* con *tp1-MIPS*:

```
$ ./tp1-MIPS -o salida matriz
```

Para imprimir la información de ayuda, se usa uno de los siguientes:

```
$ ./tp1 -h
$ ./tp1 --help
```

Finalmente, los siguientes imprimirán la información de la versión:

```
$ ./tp1 -V
$ ./tp1 --version
```

3. Corridas de prueba

Se hicieron unas corridas de prueba para asegurar la exactitud y funcionalidad del programa. Usamos el mismo comando para cada una:

```
$ ./tp1 -o - matrix[#/]
```

3.1. matrix1

El archivo original es

```
$ 1 7
$ 1 2 3 4 5 6 7
```

Usando el comando:

```
$ ./tp1 -o - matrix1
$ salida por stdout
$ 7 1
$ 1
$ 2
$ 3
$ 4
$ 5
$ 6
$ 7
```

Los datos están traspuestos, prueba exitosa.

3.2. matrix2

Entrada:

```
$ 3 5
$ 1234 5678 910 1112 1314
$ 20 300 4000 50000
$ 5 4 3 2
```

Salida:

```
$ salida por stdout
$ Error: faltan columnas en fila 3
```

El archivo de entrada dice que va a haber 5 columnas por tres filas, pero en la tercera fila sólo hay 4. Esto es un error y el programa lo encuentra. Prueba exitosa.

3.3. matrix3

El último archivo de prueba contiene:

```
$ 5 3
$ 1 2 34.5
$ 6 7 hola
$ 8 9 10
$ 11 12 13
```

Al fin eso da:

```
$ salida por stdout
$ Error de conversion en fila: 1, columna: 3
```

El programa sólo acepta números enteros. Como 34,5 no es entero, causa un error. El programa no llega hasta la "hola". Prueba exitosa.



Ademas de las pruebas anteriores, se implemento un conjunto de pruebas automatizadas que se puede correr con el comando.

```
$ ./testing.sh
```

4. Diagramas

Para mostrar cómo funciona el código Assembly, hay unos diagramas del *stack*.

La función trasponer en assembler, tiene la siguiente ABI.

44	
40	gp
36	ra
32	\$fp
28	
24	j
20	contadorMatrixSalida
16	i
12	a3
8	a2
4	a1
0	a0

El stack de la función trasponer tiene un total de 48 bytes.

ABI de la función main es la siguiente:

68	
64	gp
60	ra
56	\$fp
52	&matrix
48	cantColumnas
44	cantFilas
40	ingresoGuion
36	&outputFile
32	&inputFile
28	&long_opt
24	&short_opt
20	&file
16	option
12	
8	
4	argv
0	argc

El stack de main tiene un total de 72 bytes.

5. Código fuente

5.1. Versión C

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

#define ERROR -1
#define SALIDA_EXITOSA 0
FILE* validacionDeArchivoEntrada(int argc, char *argv
    [], bool ingresoGuion){
    FILE *auxi = NULL;
    auxi = fopen( argv[argc - 1], "r" );
    if(auxi == NULL){
        fprintf(stderr, "Error en el archivo _
            Entrada:_%s\n", strerror(errno));
        return NULL;
    }
    return auxi;
}

void guardarMatriz(long long *matrix, int cantColumnas,
    int cantFilas, FILE *fp){
    fprintf( fp, "%d_%d\n", cantFilas, cantColumnas
        );
    int i=0;
    int contadorColumna = 0;
    for (i = 0; i < (cantColumnas*cantFilas); i++){
        if(contadorColumna == cantColumnas){
            fprintf( fp, "\n");
            contadorColumna = 0;
        }
        fprintf( fp, "%dli", matrix[i] );
        contadorColumna++;
        if(contadorColumna != cantColumnas){
            fputc( ' ', fp );
        }
    }
}
```

```

        }
    }
    if ( NULL != fp ) fputc( '\n', fp);
    else puts( "\n" );
}
int trasponer(unsigned int filas , unsigned int columnas
, long long *entrada , long long *salida){
    int i = 0;
    int contadorMatrixSalida = 0;
    for(i = 0; i < columnas; i++){
        int j = 0;
        for(j = 0; j < filas; j++){
            salida[contadorMatrixSalida] =
                entrada[j*columnas+i];
            contadorMatrixSalida++;
        }
    }
    return 1;
}

int llenarFila(char *fila ,long long *vectorFila ,int *
cantColumnas ,int contadorFilas){
    int contadorColumna = 0;
    char *auxi = malloc((*cantColumnas)*sizeof(char
));
    char *eptr;
    int contadorAuxi = 0;
    int contadorFila = 0;
    int tamanioFile = strlen(fila);
    while(contadorFila <= tamanioFile){
        if(fila[contadorFila] == '_' ||
            contadorFila == tamanioFile){
            auxi[contadorAuxi] = '\0';
            long long numero = strtoll(auxi
, &eptr , 10);
            if (*eptr){
                fprintf(stderr , "Error de _
conversion en _fila: _%a _ ,
columna: _%a\n" ,
                contadorFilas ,*
                cantColumnas);
            }
        }
    }
}

```



```

        return ERROR;
    }
    if(contadorAuxi > 0){
        vectorFila[
            contadorColumna] =
            numero;
        contadorColumna++;
        contadorAuxi = 0;
    }
    if(contadorColumna > (*
        cantColumnas)){
        free(aux);
        fprintf(stderr, "Error:
            _exceso_de_columnas_
            en_fila_%a\n",
            contadorFilas);
        return ERROR;
    }
} else{
    aux[contadorAuxi] = fila[
        contadorFila];
    contadorAuxi++;
}
    contadorFila++;
}
if(contadorColumna < (*cantColumnas)){
    free(aux);
    fprintf(stderr, "Error: _faltan _columnas
        _en_fila_%a\n", contadorFilas);
    return ERROR;
}
free(aux);
return SALIDA_EXITOSA;
}

```

```

long long *parsearMatriz(FILE *inputFile, int *
    cantFilas, int *cantColumnas){
    fscanf(inputFile, "%a_%a\n", cantFilas,
        cantColumnas);
    long long *matrix = malloc(((cantFilas)*(cantColumnas))*sizeof(long long));
}

```

```

int contadorMatrix = 0;
int contadorFilas = 0;
char *fila = NULL;
int contadorCadenaFila = 0;
char caracter = fgetc(inputFile);
while (!feof(inputFile)){
    while((!feof(inputFile)) && (caracter!=
        '\n')){
        if(fila == NULL){
            fila = malloc((
                contadorCadenaFila
                +1)*sizeof(char));
            fila[contadorCadenaFila
                ] = caracter;
        }else{
            fila = realloc(fila , (
                contadorCadenaFila
                +1)* sizeof(char));
            fila[contadorCadenaFila
                ]=caracter;
        }
        contadorCadenaFila++;
        caracter = fgetc(inputFile);
    }
    if(fila != NULL){
        contadorFilas++;
        if(contadorFilas > (*cantFilas)
            ){
            fprintf(stderr , "Error:
                _exceso_de_filas\n")
                ;
            return NULL;
        }

        fila = realloc(fila , (
            contadorCadenaFila+1)*
            sizeof(char));
        fila[contadorCadenaFila]='\0';

        long long *vectorFila = malloc
            ((*cantColumnas)*sizeof(long

```

```

        long));
    if(llenarFila(fila, vectorFila,
        cantColumnas, contadorFilas)
        == ERROR){
        return NULL;
    }
    int i= 0;
    for(i = 0; i < *cantColumnas; i
        ++){
        matrix[contadorMatrix]
            = vectorFila[i];
        contadorMatrix++;
    }
    free(vectorFila);
    free(fila);
}
fila = NULL;
contadorCadenaFila = 0;
if(!feof(inputFile)){
    caracter = fgetc(inputFile);
}
}
if(contadorFilas < (*cantFilas)){
    fprintf(stderr, "Error: _faltan _filas\n"
        );
    return NULL;
}
return matrix;
}

int main(int argc, char *argv[])
{
    int option = 0;
    char *file;
    const char *short_opt = ":o:hV";
    struct option long_opt[] = {
        {"version", no_argument, NULL, 'V'},
        {"help", no_argument, NULL, 'h'},
        {"output", optional_argument, NULL, 'o'},
        {NULL, 0, NULL, 0}
    };
};

```

```

FILE *inputFile = NULL;
FILE *outputFile = NULL;
bool ingresoGuion = false;
while ((option = getopt_long(argc, argv, short_opt,
    long_opt, NULL)) != -1) {
    switch (option) {
        case 'V':
            printf("TP_#0_de_la_materia_
                Organizacion_de_Computadoras
                _\n");
            printf("Alumno: _\n");
            printf(" _ _ _ _ _ Quino_Lopez_
                Julian_\n");
            printf(" _ _ _ _ _ Lund_Eirik_
                Harald_\n");
            return SALIDA_EXITOSA;
        case 'h':
            printf("Usage: _\n");
            printf(" _ _ _ _ _ %s_-h_\n", argv
                [0]);
            printf(" _ _ _ _ _ %s_-V_\n", argv
                [0]);
            printf(" _ _ _ _ _ %s_[options]_
                file_\n", argv[0]);
            printf("Options: _\n");
            printf(" _ _ _ _ _-V, _ _ _ _ _version _ _
                Print_version_and_quit._\n")
                ;
            printf(" _ _ _ _ _-h, _ _ _ _ _help _ _ _ _ _
                Print_this_information._\n")
                ;
            printf(" _ _ _ _ _-o, _ _ _ _ _output _ _ _ _ _
                Path_to_output_file._\n");
            printf("\n_Examples: _\n");
            printf(" _ _ _ _ _ %s_- _mymatrix_\n
                ", argv[0]);
            printf(" _ _ _ _ _ %s_-o _ _
                mymatrix_\n", argv[0]);
            return SALIDA_EXITOSA;
        case 'o':
            if(strcmp(optarg, "-") != 0){

```

```

        file = optarg;
        outputFile = fopen(
            file , "w" );
        if (outputFile == NULL)
        {
            fprintf(stderr ,
                "Error _
                archivo _
                Salida: _ %s\n
                " , strerror(
                    errno));
            return ERROR;
        }
    }
    else{
        ingresoGuion = true;
        printf ("salida _por _
            stdout\n");
    }
    break;
default:
    fprintf(stderr , "Parametro _
        invalido , _use _el _comando _-h\
        n");
    return ERROR;
    break;
}

}

inputFile = validacionDeArchivoEntrada(argc , argv ,
    ingresoGuion);
if(inputFile==NULL){
    return ERROR;
}
int cantFilas = 0;
int cantColumnas = 0;
long long *matrix = parsearMatriz(inputFile , &
    cantFilas , &cantColumnas);
if(matrix==NULL){
    return ERROR;
}

```

```

long long *matrixSalida = malloc((cantFilas*
    cantColumnas)*sizeof(long long));

trasponer(cantFilas, cantColumnas, matrix,
    matrixSalida);

    if ( NULL == outputFile ) outputFile = stdout;
    guardarMatriz(matrixSalida, cantFilas,
        cantColumnas, outputFile);

    if(outputFile != stdout){
        fclose(outputFile);
    }

    fclose(inputFile);
    free(matrix);
    free(matrixSalida);
    return SALIDA_EXITOSA;
}

```

5.2. Versión MIPS

```

#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>
#include <getopt.h>
#include <unistd.h>
#include <errno.h>
#include <string.h>

#define ERROR -1
#define SALIDA_EXITOSA 0
FILE* validacionDeArchivoEntrada(int argc, char *argv
    [], bool ingresoGuion){
    FILE *auxi = NULL;
    auxi = fopen( argv[argc - 1], "r" );
    if(auxi == NULL){
        fprintf(stderr, "Error en el archivo _
            Entrada: _%s\n", strerror(errno));
        return NULL;
    }
}

```

```

    }
    return auxi;
}
void guardarMatriz(long long *matrix, int cantColumnas,
    int cantFilas, FILE *fp){
    fprintf( fp, "%d_%d\n", cantFilas, cantColumnas
    );
    int i=0;
    int contadorColumna = 0;
    for (i = 0; i < (cantColumnas*cantFilas); i++){
        if(contadorColumna == cantColumnas){
            fprintf( fp, "\n");
            contadorColumna = 0;
        }
        fprintf( fp, "%dli", matrix[i] );
        contadorColumna++;
        if(contadorColumna != cantColumnas){
            fputc( '_', fp );
        }
    }
    if ( NULL != fp ) fputc( '\n', fp);
    else puts( "\n" );
}
extern int trasponer(unsigned int filas, unsigned int
    columnas, long long *entrada, long long *salida);

int llenarFila(char *fila, long long *vectorFila, int *
    cantColumnas, int contadorFilas){
    int contadorColumna = 0;
    char *auxi = malloc((*cantColumnas)*sizeof(char
    ));
    char *eptr;
    int contadorAuxi = 0;
    int contadorFila = 0;
    int tamanioFile = strlen(fila);
    while(contadorFila <= tamanioFile){
        if(fila[contadorFila] == '_' ||
            contadorFila == tamanioFile){
            auxi[contadorAuxi] = '\0';
            long long numero = strtoll(auxi
            , &eptr, 10);

```

```

        if (*eptr){
            fprintf(stderr, "Error de _
            conversion en fila: _%d _
            columna: _%d \n",
            contadorFilas, *
            cantColumnas);
            return ERROR;
        }
        if(contadorAuxi > 0){
            vectorFila[
                contadorColumna] =
                numero;
            contadorColumna++;
            contadorAuxi = 0;
        }
        if(contadorColumna > (*
            cantColumnas)){
            free(auxi);
            fprintf(stderr, "Error:
            _exceso de _columnas _
            en _fila _%d \n",
            contadorFilas);
            return ERROR;
        }
    }else{
        auxi[contadorAuxi] = fila[
            contadorFila];
        contadorAuxi++;
    }
    contadorFila++;
}
if(contadorColumna < (*cantColumnas)){
    free(auxi);
    fprintf(stderr, "Error: _faltan _columnas
    _en _fila _%d \n", contadorFilas);
    return ERROR;
}
free(auxi);
return SALIDA_EXITOSA;
}

```



```

long long *parsearMatriz(FILE *inputFile, int *
cantFilas, int *cantColumnas){
    fscanf(inputFile, "%d %d\n", cantFilas,
cantColumnas);
    long long *matrix = malloc(((cantFilas)*(cantColumnas))*sizeof(long long));
    int contadorMatrix = 0;
    int contadorFilas = 0;
    char *fila = NULL;
    int contadorCadenaFila = 0;
    char caracter = fgetc(inputFile);
    while (!feof(inputFile)){
        while ((!feof(inputFile)) && (caracter!=
'\n')){
            if(fila == NULL){
                fila = malloc((
                    contadorCadenaFila
                    +1)*sizeof(char));
                fila[contadorCadenaFila
                    ] = caracter;
            }else{
                fila = realloc(fila, (
                    contadorCadenaFila
                    +1)* sizeof(char));
                fila[contadorCadenaFila
                    ]=caracter;
            }
            contadorCadenaFila++;
            caracter = fgetc(inputFile);
        }
        if(fila != NULL){
            contadorFilas++;
            if(contadorFilas > (cantFilas)
                ){
                fprintf(stderr, "Error:
                _exceso_de_filas\n");
                ;
                return NULL;
            }

            fila = realloc(fila, (

```

```

        contadorCadenaFila+1)*
        sizeof(char));
    fila [contadorCadenaFila]='\0';

    long long *vectorFila = malloc
        ((*cantColumnas)*sizeof(long
        long));
    if (llenarFila (fila , vectorFila ,
        cantColumnas , contadorFilas)
        == ERROR){
        return NULL;
    }
    int i= 0;
    for (i = 0; i < *cantColumnas; i
        ++){
        matrix [contadorMatrix]
            = vectorFila [i];
        contadorMatrix++;
    }
    free (vectorFila);
    free (fila);
}
fila = NULL;
contadorCadenaFila = 0;
if (!feof(inputFile)){
    caracter = fgetc(inputFile);
}
}
if (contadorFilas < (*cantFilas)){
    fprintf(stderr , "Error: _faltan _filas\n"
        );
    return NULL;
}
return matrix;
}

int main(int argc , char *argv [])
{
    int option = 0;
    char *file;
    const char *short_opt = ":o:hV";

```

```

struct option long_opt [] = {
    {"version", no_argument,      NULL, 'V'},
    {"help",    no_argument,      NULL, 'h'},
    {"output",  optional_argument, NULL, 'o'},
    {NULL, 0,   NULL, 0}
};
FILE *inputFile = NULL;
FILE *outputFile = NULL;
bool ingresoGuion = false;
while ((option = getopt_long(argc, argv, short_opt,
    long_opt, NULL)) != -1) {
    switch (option) {
        case 'V':
            printf("TP_#0_de_la_materia_
                Organizacion_de_Computadoras
                _\n");
            printf("Alumno:\n");
            printf("_____Quino_Lopez_
                Julian_\n");
            printf("_____Lund_Eirik_
                Harald_\n");
            return SALIDA_EXITOSA;
        case 'h':
            printf("Usage:\n");
            printf("_____ %s -h\n", argv
                [0]);
            printf("_____ %s -V\n", argv
                [0]);
            printf("_____ %s [options] _
                file\n", argv[0]);
            printf("Options:\n");
            printf("_____ -V, --version _
                Print_version_and_quit._\n")
                ;
            printf("_____ -h, --help _
                Print_this_information._\n")
                ;
            printf("_____ -o, --output _
                Path_to_output_file._\n");
            printf("\n_Examples:\n");
            printf("_____ %s _mymatrix_\n

```

```

        n", argv[0]);
        printf(" %s_o_o_
        mymatrix\n", argv[0]);
        return SALIDA_EXITOSA;
    case 'o':
        if(strcmp(optarg, "-") != 0){
            file = optarg;
            outputFile = fopen(
                file, "w" );
            if (outputFile == NULL)
            {
                fprintf(stderr,
                    "Error_
                    archivo_
                    Salida:_%s\n
                    ", strerror(
                        errno));
                return ERROR;
            }
        }
        else{
            ingresoGuion = true;
            printf ("salida_por_
                stdout\n");
        }
        break;
    default:
        fprintf(stderr, "Parametro_
            invalido ,_use_el_comando_-h\
            n");
        return ERROR;
        break;
    }

}

inputFile = validacionDeArchivoEntrada(argc, argv,
    ingresoGuion);
if(inputFile==NULL){
    return ERROR;
}
int cantFilas = 0;

```

```

    int cantColumnas = 0;
    long long *matrix = parsearMatriz(inputFile,&
        cantFilas , &cantColumnas);
    if(matrix==NULL){
        return ERROR;
    }
    long long *matrixSalida = malloc((cantFilas*
        cantColumnas)*sizeof(long long));

    trasponer(cantFilas , cantColumnas , matrix ,
        matrixSalida);

    if ( NULL == outputFile ) outputFile = stdout;
    guardarMatriz(matrixSalida , cantFilas ,
        cantColumnas , outputFile);

    if(outputFile != stdout){
        fclose(outputFile);
    }

    fclose(inputFile);
    free(matrix);
    free(matrixSalida);
    return SALIDA_EXITOSA;
}

```

5.3. Código Assembly

```

#include <mips/regdef.h>

    .text
    .align 2
    .globl trasponer
    .ent trasponer
trasponer:

    .frame $fp ,48 , ra
    .set noreorder
    .cpload t9
    .set reorder

```

```

subu          sp, sp, 48
.cprestore    16
sw            $fp, 32(sp)
sw            ra, 36(sp)
sw            gp, 40(sp)

sw            a0, 0(sp)
sw            a1, 4(sp)
sw            a2, 8(sp)
sw            a3, 12(sp)

move          $fp, sp

sw            zero, 16($fp)
             #t0 int i=0
sw            zero, 20($fp)
             #int contadorMatrixSalida = 0;
sw            zero, 24($fp)
#for(i = 0; i < columnas; i++){
forPrincipal:
lw            t0, 16($fp)
bge           t0, a1,
             trasponerMipsFin

sw            zero, 24($fp)
             #int j = 0;

             #for(j = 0; j < filas; j++)
forSecundario:
lw            t0, 24($fp)
bge           t0, a0,
             salirDeForPrincipal

#salida[contadorMatrixSalida] = entrada[j*
             columnas+i];
move          t0, a1
lw            t1, 24($fp)
sll           t1, t1, 3
mul           t1, t1, t0
             #j*columnas
lw            t0, 16($fp)

```

```

sll                                t0, t0, 3
addu                               t1, t0, t1
                                #j*columnas+i
addu                               t1, t1, a2
lw                                t3, (t1)
                                #entrada[j*columnas+
                                i]
lw                                t4, 4(t1)
                                #entrada[j*columnas+i]

lw                                t0, 20($fp)
sll                                t0, t0, 3
                                #
                                contadorMatrixSalida
addu                               t0, a3, t0

sw                                t3, (t0)
sw                                t4, 4(t0)
                                #salida[contadorMatrixSalida
                                ]
lw                                t0, 20($fp)
addu                               t0, t0, 1
                                #contadorMatrixSalida++;
sw                                t0, 20($fp)

lw                                t0, 24($fp)
addu                               t0, t0, 1
sw                                t0, 24($fp)
b                                forSecundario

salirDeForPrincipal:
lw                                t0, 16($fp)
addu                               t0, t0, 1
sw                                t0, 16($fp)
b                                forPrincipal

trasponerMipsFin:
li                                v0, 1
lw                                ra, 36($fp)
lw                                $fp, 32($fp)
lw                                gp, 40($fp)

```

```
addu          sp, sp, 48
j             ra
.end          trasponer
```