

# Informe TP1

Lautaro Javier Ituarte, *Padrón Nro. 93639*  
lautaro.javier.ituarte@gmail.com

Nicolás Longo, *Padrón Nro. 98271*  
longo.gnr@hotmail.com

Federico Soldo, *Padrón Nro. 98288*  
federicosoldo@hotmail.com

2do. Cuatrimestre de 2017  
66.20 Organización de Computadoras – Práctica Martes s  
Facultad de Ingeniería, Universidad de Buenos Aires

10 de octubre del 2017

## 1. Enunciado

### 1. Objetivos

Familiarizarse con el conjunto de instrucciones MIPS y el concepto de ABI, extendiendo un programa que resuelva el problema descrito en la sección 4.

### 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, por escrito, en la fecha estipulada, con una carátula que contenga los datos completos de todos los integrantes. Además, es necesario que el trabajo práctico incluya (entre otras cosas, ver sección 5), la presentación de los resultados obtenidos explicando, cuando corresponda, con fundamentos reales, las causas o razones de cada resultado obtenido.

### 4. Descripción

En este trabajo, se reimplementará parcialmente en assembly MIPS el programa desarrollado en el trabajo práctico anterior. Para esto, se requiere reescribir el programa, de forma tal que quede organizado de la siguiente forma:

- Arranque y configuración del programa: procesamiento de las opciones de línea de comandos, apertura y cierre de archivos (de ser necesario), y reporte de errores. Desde aquí se invocará a la función de procesamiento del stream de entrada.
- Procesamiento: contendrá el código MIPS32 assembly con la función `palindrome()`, encargada de identificar, procesar e imprimir los componentes léxicos que resulten ser palíndromos, de forma equivalente a lo realizado en el TP anterior.

La función MIPS32 `palindrome()` antes mencionada se corresponderá con el siguiente prototipo en C:

```
int palindrome(int ifd, size_t ibytes, int ofd, size_t obytes);
```

Esta función es invocada por el módulo de arranque y configuración del programa, y recibe en `ifd` y `ofd` los descriptores abiertos de los archivos de entrada y salida respectivamente. Los parámetros `ibytes` y `obytes` describen los tamaños en bytes de las unidades de transferencia de datos desde y hacia el kernel de NetBSD, y permiten implementar un esquema de buffering de estas operaciones de acuerdo al siguiente diagrama: INCLUIR DIAGRAMA + PIE DE IMAGEN Como puede verse en la figura 1, la lógica de procesamiento de la función `palindrome()` va leyendo los caracteres del buffer de entrada en forma individual. En el momento en el cual `palindrome()` intente extraer un nuevo carácter, y el buffer de entrada se encuentre vacío, deberá ejecutar una llamada al sistema operativo para

realizar una lectura en bloque y llenar completamente el buffer, siendo el tamaño de bloque igual a `ibytes` bytes. De forma análoga, `palindrome()` irá colocando uno a uno los caracteres de las palabras capicúa en el buffer de salida. En el momento en el que se agote su capacidad, deberá vaciarlo mediante una operación de escritura hacia el kernel de NetBSD para continuar luego con su procesamiento. Al finalizar la lectura y procesamiento de los datos de entrada, es probable que exista información esperando a ser enviada al sistema operativo. En ese caso `palindrome()` deberá ejecutar una última llamada al sistema con el fin de vaciar completamente el buffer de salida. Se sugiere encapsular la lógica de buffering de entrada/salida con funciones, `getch()` y `putch()`. Asimismo durante la clase del martes 19/9 explicaremos la función `mymalloc()` que deberá ser usada para reservar dinámicamente la memoria de los buffers.

## 5. Ejemplos

Primero, usamos la opción `-h` para ver el mensaje de ayuda: `tp0 -h`

Usage: `tp0 -h tp0 -V tp0 [options]` Options: `-V`, `-version` Print version and quit. `-h`, `-help` Print this information. `-i`, `-input` Location of the input file. `-o`, `-output` Location of the output file. `-I`, `-ibuf-bytes` Byte-count of the input buffer. `-O`, `-obuf-bytes` Byte-count of the output buffer. Examples: `tp0 -i /input -o /output` Codificamos un archivo vacío (cantidad de bytes nula): `touch /tmp/zero.txt tp0 -i /tmp/zero.txt -o /tmp/out.txt ls -l /tmp/out.txt -rw-r--r-- 1 user group 0 2017-03-19 15:14 /tmp/out.txt` Leemos un stream cuyo único contenido es el carácter ASCII M, `echo Hola M` — `tp0 M` Observar que la salida del programa contiene aquellas palabras de la entrada que sean palíndromos (M en este caso). Veamos que sucede al procesar archivo de mayor complejidad: `cat entrada.txt` Somos los primeros en completar el TP 0. Ojo que La fecha de entrega del TP0 es el martes 12 de septiembre. `tp0 -i entrada.txt -o -` Somos Ojo

## 6. Informe

El informe deberá incluir al menos las siguientes secciones:

- 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, el cual también deberá entregarse en formato digital compilable (incluyendo archivos de entrada y salida de pruebas).
- Este enunciado.

El informe deberá entregarse en formato impreso y digital.

## 7. Fechas

- Entrega: 26/9/2017.
- Vencimiento: 10/10/2017.

Referencias [1] GXemul, <http://gavare.se/gxemul/>. [2] The NetBSD project,

<http://www.netbsd.org/>.

## 2. Implementación del programa

El TP consta de 2 grandes partes:

- Un primer fragmento de código programado en C.
- Una segunda parte, programada en lenguaje Assembly.

La parte del trabajo programada en C tiene como objetivo preparar el entorno de trabajo (y las respectivas variables) para poder pasarle el control a la función `palindrome` programada en Assembly (MIPS32) y que es la encargada de detectar los palíndromos propiamente dichos para un texto de entrada dado. En primera instancia, es el fragmento de código en C el que, una vez ejecutado, se encarga de interpretar cada uno de los parámetros (y toda combinación posible de estos) para que, de ésta forma, el comportamiento sea el esperado: si se pide la versión, o el mensaje de ayuda, los muestra; si se especifica un archivo de entrada y/o salida se encarga de cargarlos en memoria, ya sean estos archivos de texto o la entrada estándar y/o salida estándar. Una vez interpretados los parámetros y abierto los archivos se utiliza el file descriptor de estos y se le otorga el control a la función `palindrome`. Si la secuencia de parámetros recibidos por el programa no son los esperados, es el `main` el que se encarga de imprimir un mensaje de error terminando la ejecución del programa y devolviendo -1. De la misma forma, si el manejo de archivos falla, el programa también devuelve -1 y utilizando la variable `errno` y la función `strerror` se imprime un mensaje por `stderr` con el código de error correspondiente. Por último, es necesario mencionar que ante el fallo de una operación de escritura (utilizando las funciones `printf` y `fprintf`) el programa se interrumpe devolviendo el mismo valor que en los casos anteriormente mencionados. La parte del trabajo programada en Assembly prepara, en primera instancia, un fragmento de memoria (buffer) para almacenar en éste los caracteres a leer desde el archivo de entrada especificado. A la hora de pedir memoria al sistema operativo usamos la función `mymalloc.S`, provista por la cátedra, que devuelve la posición inicial de memoria asignada, cuyo tamaño es el especificado (este es uno de los argumentos que recibe). De la misma forma, nos reservamos una porción de memoria que utilizaremos para ir reservando, en tiempo de ejecución, las palabras que vamos detectando en el archivo de entrada. Desde este "buffer de palabras" evaluaremos si son o no palíndromos. Para leer el archivo en si implementamos una función, `getch`, que se encarga de tomar el próximo carácter del archivo y entregárselo al control del programa en `palindrome`. Esta es la rutina que detecta el fin del archivo, ante el cual se da por terminado el procesamiento de data. Si la lectura del carácter no corresponde al final del archivo, entonces se evalúa si el mismo es o no un espacio, de acuerdo con los parámetros establecidos por la cátedra. De esto se encarga la función `esEspacio`, la cual devuelve un 0 sino es espacio, o un 1 si lo es. Una vez lleno con una palabra el buffer de palabras, se llama a la función `palindromeString`, la cual recibe la dirección de comienzo de la palabra y su `len` menos 1, y analiza si es palíndromo, en cuyo caso realiza el `sys call` correspondiente para escribir el archivo de output especificado, y en caso de no serlo simplemente vuelve a la función que lo llamo, sin hacer nada.

### 3. Comandos para compilar el programa

- `~ gcc -Wall tp1.c palindrome.S palindromeString.S esEspacio.S mymalloc.S -o nombreExecutable`

### 4. Código fuente

- `tp1.c`:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>
#include <unistd.h>
#include <errno.h>
#include "palindrome.h"

int palindrome(int ifd, size_t ibytes, int ofd, size_t obytes);

int aperturaDeArchivos(char* inName, FILE** input_file, char* outName, FILE**
output_file){
    if (inName == NULL){
        *input_file = stdin;
    }
    else{
        if ((*input_file = fopen(inName, "rt")) == NULL){
            if (fprintf(stderr, "No se pudo abrir el archivo el archivo de entrada: %s\n",
strerror(errno)) < 0){
                fprintf(stderr, "Fallo en la ejecucion de la funcion fprintf o printf");
            }
            return -1;
        }
    }
    if (outName == NULL){
        *output_file = stdout;
    }
    else{
        if ((*output_file = fopen(outName, "wt")) == NULL){
            if (fprintf(stderr, "No se pudo abrir el archivo el archivo de salida: %s\n",
strerror(errno)) < 0){
                fprintf(stderr, "Fallo en la ejecucion de la funcion fprintf o printf");
            }
            return -1;
        }
    }
    return 0;
}
```

```

char* seIngresoParametro_io(char* par, int len, char** argv){
    if (strcmp(par, "-i") == 0){
        int i;
        for (i = 1; i < len; i++){
            if (strcmp(argv[i], "-i") == 0){
                return argv[i+1];
            }
        }
    }
    else{
        int j;
        for (j = 1; j < len; j++){
            if (strcmp(argv[j], "-o") == 0){
                return argv[j+1];
            }
        }
    }
    return NULL;
}

```

```

size_t seIngresoParametro_buf(char* par, int len, char** argv){
    char* ptr;
    if (strcmp(par, "-I") == 0){
        int i;
        for (i = 1; i < len; i++){
            if (strcmp(argv[i], "-I") == 0){
                return (size_t) strtol(argv[i+1], &ptr, 10);
            }
        }
    }
    else{
        int j;
        for (j = 1; j < len; j++){
            if (strcmp(argv[j], "-O") == 0){
                return (size_t) strtol(argv[j+1], &ptr, 10);
            }
        }
    }
    return 1;
}

```

```

int verificarParametrosInvalidos(int len, char** argv){
    char* ptr;
    int i;
    for (i = 1; i < len; i+=2){ //SALTO DE PARAMETRO EN PARAMETRO, PREVIAMENTE EN EL MAIN
        VERIFIQUE
        //EL PODER REALIZAR ESTOS SALTOS.
        if (strcmp(argv[i], "-i") == 0 || strcmp(argv[i], "-o") == 0 ||
            strcmp(argv[i], "-I") == 0 || strcmp(argv[i], "-O") == 0){

```

```

//ANALIZO SI -I Y -O SON NUMEROS VALIDOS
if (strcmp(argv[i], "-I") == 0 || strcmp(argv[i], "-O") == 0){
if (strtoul(argv[i+1], &ptr, 10) <= 0){
return 1;
}
}
}
else{
return 1;
}
}
return 0;
}

int mostrarMensajeVersion()
{
    if (printf("%s\n", "Version 1.0") < 0){
fprintf(stderr, "Fallo en la ejecucion de la funcion fprintf o printf");
return -1;
}
    return 0;
}

int mostrarMensajeAyuda()
{
    if (printf("%s\n", "Usage:\ntp0 -h\ntp0 -V\ntp0 [options]\nOptions:\n-V, --version
Print version and quit.\n-h, --help Print this information.\n-i, --input
Location of the input file.\n-o, --output Location of the output file.\n
-I, --ibuf-bytes Byte-count of the input buffer.\n
-O, --obuf-bytes Byte-count of the output buffer.\n
Examples:\ntp0 -i ~/input -o ~/output -I ~/buf_in_size -O ~/buf_out_size") < 0){
fprintf(stderr, "Fallo en la ejecucion de la funcion fprintf o printf");
return -1;
}
    return 0;
}

int mostrarMensajeErrorParametrosInvalidos()
{
    if (fprintf(stderr, "Los parámetros ingresados no son válidos.\n") < 0){
fprintf(stderr, "Fallo en la ejecucion de la funcion fprintf o printf");
}
    return -1;
}

int main(int argc, char** argv){

char* input_fileName;
char* output_fileName;

```

```

size_t bufferIn, bufferOut;

// No puede ser mayor que nueve porque sólo se pueden pasar 8 parametros al
// programa más
// el "nombre del programa" que se encuentra en el arc
// No puede tener ni 4 ni 6 ni 8 párametros porque significaria que "-i" o "-o"
// o "-I" o
// "-O" no tienen el nombre del archivo especificado o el
// tamaño de los buffers
if (argc > 9 || argc == 4 || argc == 6 || argc == 8)
{
    return mostrarMensajeErrorParametrosInvalidos();
}

// Si se recibió un solo parámetro
if (argc == 2)
{
    if (strcmp(argv[1], "-V") == 0)
    {
        return mostrarMensajeVersion();
    }
    else if (strcmp(argv[1], "-h") == 0)
    {
        return mostrarMensajeAyuda();
    }
    else
    {
        return mostrarMensajeErrorParametrosInvalidos();
    }
}

if (verificarParametrosInvalidos(argc, argv)){
return mostrarMensajeErrorParametrosInvalidos();
}
//ESTO FUE UNA PRUEBA PARA VER SI ANDABA Todo.
input_fileName = seIngresoParametro_io("-i", argc, argv);
output_fileName = seIngresoParametro_io("-o", argc, argv);
bufferIn = seIngresoParametro_buf("-I", argc, argv);
bufferOut = seIngresoParametro_buf("-O", argc, argv);
FILE* input_file = NULL;
FILE* output_file = NULL;
if (aperturaDeArchivos(input_fileName, &input_file, output_fileName, &output_file)
== -1) {
    if (fprintf(stderr, "Alguno de los archivos ingresados no pudo ser abierto.\n")
< 0){
fprintf(stderr, "Fallo en la ejecucion de la funcion fprintf o printf");
}
    return -1;
}
int ifd = fileno(input_file);

```



```

    int ofd = fileno(output_file);

    palindrome(ifd, bufferIn, ofd, bufferOut); //ACA LLAMAMOS A LA FUNCION PALINDROME
    DE MIPS
    fclose(input_file);
    fclose(output_file);
    return 0;
}

```

■ palindrome.S:

```

#include <mips/regdef.h>
#include <sys/syscall.h>
.text
.abicalls
.globl palindrome
.ent palindrome

palindrome:
    .frame $fp, 48, ra
    .set noreorder
    .cload t9
    .set reorder

    subu sp, sp, 48 # pido espacio para mi Stack Frame
    .cprestore 36 # salvo gp en 36
    sw $fp, 32(sp) # salvo fp en 32
    sw ra, 40(sp) # salvo ra en 40
    move $fp, sp # a partir de acá trabajo con fp

    # me guardo los parámetros tp1.c (por convención de ABI)
    sw a0, 48($fp) # salvo el file descriptor del input file
    sw a1, 52($fp) # salvo el tamaño del buffer de entrada
    sw a2, 56($fp) # salvo el file descriptor del output file
    sw a3, 60($fp) # salvo el tamaño del buffer de salida

    # me guardo los parámetros como variables globales
    sw a0, FDESCRIPTOR_DE_LECTURA
    sw a1, IBYTES
    sw a2, FDESCRIPTOR_DE_ESCRITURA
    sw a3, OBYTES

    # reservo memoria para el buffer de entrada
    lw a0, IBYTES # preparo a0 para pasarselo a mymalloc
    jal mymalloc
    sw v0, 16($fp) # salvo la posición inicial del buffer en el stack frame
    sw v0, POS_INICIAL_IB # lo guardo como variable global

```

```

lw a0, IBYTES # utilizamos tamaño que el buff in
jal mymalloc # en v0 tengo la posición de memoria del buffer para
palabras
sw v0, 20($fp) # pos_actual del buffwords
sw v0, POS_INICIAL_BUFF_PAL

lw a0, POS_INICIAL_IB # preparo los argumentos para getch, paso la posición actual
del buffin (que resulta ser la inicial)
li a1, 1 # tiene que llenar el buffer
lecturaArchivo:
jal getch # empiezo a leer
beqz v0, finDeLectura # si el char es 0, EOF. acá estamos evaluando si el puntero
es 0. esto no debería cumplirse nunca
sw v1, 16($fp) # me guardo la posición actual del buff in
sw v0, 24($fp) # me guardo el puntero al primer carácter

move a0, v0 # guardo el último char leído
lb a0, 0(a0)
jal esEspacio # me fijo si el carácter es un espacio
# en v0 está si es un espacio = 1, sino = 0
beq v0, 0, _definirLargoDePalabra

_noEsPalabra:
lw a0, 16($fp) # si no es palabra, cargo la posición actual del buff
in y vuelvo
li a1, 0
b lecturaArchivo # sigo leyendo

_definirLargoDePalabra:

lw t0, 24($fp) # cargo el puntero en t0
lb t0, 0(t0) # cargo el dato en t0
lw t1, POS_INICIAL_BUFF_PAL # cargo la posición del primer carácter
sb t0, 0(t1) # guardo el dato
addu t1, t1, 1 # adelanto el índice en buff pal
sw t1, 20($fp) # salvo el índice, pos actual del buff pal
li t7, 0 # inicializo t7 en 0 porque ya detecté el primer
carácter

_loopEsPalabra:
lw a0, 16($fp) # preparo los argumentos para getch, paso la posición
actual del buffin
li a1, 0 # no es la primera lectura
jal getch # guardarse v1 en el stack
sw v1, 16($fp) # actualizo pos_actual del buffin
lb a0, 0(v0) # preparo a0 para esEspacio
beq v0, zero, _analizarPalindromoFinal # si el char leído es un 0, EOF.
move t3, a0

```

```

jal esEspacio # me fijo si el caracter es un espacio
beq v0, 1, _analizarPalindromo # si el caracter es un espacio no escribo ni sumo
nada
addu t7, t7, 1 # escribí caracter más
lw t1, 20($fp) # me traigo la pos actual del buff pal
sb t3, 0(t1) # guardo el dato
addu t1, t1, 1 # adelanto el indice de buff pal
sw t1, 20($fp) # guardo la pos actual de buff pal
b _loopEsPalabra

# si salgo de acá, entonces ya tengo la palabra entera en el buffwords
_analizarPalindromo:
lw a0, POS_INICIAL_BUFF_PAL
move a1, t7
jal palindromeString
move t7, zero # pongo en cero mi registro t7 de nuevo
lw a0, 16($fp) # me preparo para seguir leyendo
li a1, 0
b lecturaArchivo

_analizarPalindromoFinal:
    beqz t7, finDeLectura # ésta línea casi seguro es inútil
    lw a0, POS_INICIAL_BUFF_PAL
    move a1, t7
jal palindromeString

finDeLectura:

lw ra, 40(sp)
lw $fp, 32(sp)
lw gp, 36(sp)
addu sp, sp, 48
jr ra

.end palindrome

.globl getch
.ent getch

getch:
.frame $fp, 40, ra
.set noreorder
.cpload t9
.set reorder

subu sp, sp, 40 # pido espacio para mi Stack Frame
.cprestore 28 # salvo gp en 28
sw $fp, 24(sp) # salvo fp en 24
sw ra, 32(sp) # salvo ra en 32
move $fp, sp # a partir de acá trabajo con fp

```

```

# me guardo los parámetros que no guardo la caller (por convención de ABI)
sw a0, 40($fp) # salvo posicion actual del buffer
sw a1, 44($fp) # salvo condición de lectura inicial

_if:
li t0, 1
beq t0, a1, _lectura_inicial

#compruebo que quedan caracteres por leer
la t2, POS_INICIAL_IB
lw t1, 0(t2)
subu t0, a0, t1 # le resto la pos_inicial a la pos_actual
lw t1, IBYTES
subu t0, t0, t1 # al resultado, le resto el tamaño. Si son iguales, tendré que pasar
al syscall
beq t0, 0, _rellenar_buffer

_lectura:
lw v0, 40($fp) # en v0 guardo el char (leído) que es lo que voy a devolver
lb t3, 0(v0) # para gdb
addu v1, v0, 1 # en v1, la nueva posicion actual (la anterior, más el tamaño de un
char)

b _return

_rellenar_buffer:
li v0, SYS_read
la t0, FDESCRIPTOR_DE_LECTURA
lw a0, 0(t0)
la t0, POS_INICIAL_IB
lw a1, 0(t0)
sw a1, 16($fp) # me guardo la posición inicial en LTA por si tengo que hacer una
relectura
lw a2, IBYTES # y acá está el tamaño
syscall

# Casos posibles:
# 1. El syscall puede leer todo, devuelve 0 y ibytes
# 2. El syscall puede leer parcialmente. Devuelve 0 (porque no hubo error) y un número
positivo menor a ibytes
# 3. El syscall devuelve un número negativo (hubo error). Hay código de error y está
en v0
# 4. El syscall devuelve 0 y 0 ----> eof
_comprobacion:
bltz a3, _error_en_syscall
addu t5, v0, a3
beqz t5, _eof
la t2, IBYTES
lw t1, 0(t2)

```

```

subu t0, v0, t1
bgtz t0, _relectura # si pasa ésta línea, entonces a3=0 y v0=ibytes

la t2, POS_INICIAL_IB
lw t0, 0(t2)
sw t0, 40($fp) # mi pos_actual es pos_inicial
b _lectura

_error_en_syscall: # devuelvo en v0 un -1 y en v1 el código de error
(generado por el syscall)
move v1, v0
li v0, -1
b _return

_eof: # devuelvo en v0 un 0 y en v1 un 0
li v0, 0
li v1, 0
b _return

_relectura:
lw t0, 16($fp) # mi posición inicial estaba en LTA. la levanto
add t0, t0, v0 # mi nueva posición inicial, será la anterior + los caracteres leídos
sw t0, 16($fp) # me guardo la nueva posición inicial temporal en LTA
move a1, t0
la t3, FDESCRIPTOR_DE_LECTURA
lw a0, 0(t3)
lw t1, 48($fp)
subu t0, t1, v0 # mi nuevo tamaño será el tamaño anterior - los caracteres
leídos
move a2, t0
syscall
b _comprobacion

_lectura_inicial:
b _rellenar_buffer

_return:
lw ra, 32(sp)
lw $fp, 24(sp)
lw gp, 28(sp)
addu sp, sp, 40
jr ra

.end getch

.data

AUX: .word 0
FDESCRIPTOR_DE_LECTURA: .word 0
FDESCRIPTOR_DE_ESCRITURA: .word 0

```

```

IBYTES: .word 0
OBYTES: .word 0
POS_INICIAL_IB: .word 0
POS_INICIAL_BUFF_PAL: .word 0

```

■ palindromeString.S:

```

#include <mips/regdef.h>
#include <sys/syscall.h>
.text
.abicalls
.globl palindromeString
.ent palindromeString

palindromeString:
.frame $fp, 40, ra
.set noreorder
.cpload t9
.set reorder

subu sp, sp, 40 # pido espacio para mi Stack Frame
.cprestore 28 # salvo gp en 28
sw $fp, 24(sp) # salvo fp en 24
sw ra, 32(sp) # salvo ra en 32
move $fp, sp # a partir de acá trabajo con fp

# me guardo los parámetros (por convención de ABI)
sw a0, 40($fp) # salvo el string
sw a1, 44($fp) # longitud string menos uno

# guardo en t0 el comienzo del string
move t0, a0
# guardo en t1 el final del string
addu t1, a0, a1
# guardo en t2 la mitad del string (le sumo uno porque trunca)
div t2, a1, 2
addu t2, t2, 1
# guardo en t3 las posiciones que revise (inicializo en 0)
xor t3, t3, t3
# sigo revisando si no recorri la mitad del string
_palindrome_loop:
# si ya compare todo el string finalizo
beq t2, t3, _palindrome_true
# cargo t0 y t1
lb t4, 0(t0)
lb t5, 0(t1)
# si los caracteres espejo no son iguales entonces no es palindromo
bne t4, t5, _palindrome_false

```

```

# seteo t0 y t1 para comparar los siguientes caracteres
addu t0, t0, 1
subu t1, t1, 1
# aumento contador
addu t3, t3, 1
b _palindrome_loop
# si es palindromo lo muestro con sys-write
_palindrome_true:
li v0, SYS_write
li a0, 1 #salida
lw a1, 40($fp) # guardo string
lw a2, 44($fp) # guardo la longitud menos uno
addu a2, a2, 1 # le sumo uno
syscall

li v0, SYS_write
li a0, 1 #salida
la a1, SALTO
syscall

# si no es palindromo simplemente termino
_palindrome_false:
lw ra, 32(sp)
lw $fp, 24(sp)
lw gp, 28(sp)
addu sp, sp, 40
jr ra

.end palindromeString

.rdata

SALTO: .asciiz "\n"

```

#### ■ esEspacio.S:

```

#include <mips/regdef.h>
#include <sys/syscall.h>
.text
.abicalls
.globl esEspacio
.ent esEspacio

esEspacio:
.frame $fp, 40, ra
.set noreorder
.cpload t9

```

```

.set reorder

subu sp, sp, 40 # pido espacio para mi Stack Frame
.cprestore 28 # salvo gp en 28
sw $fp, 24(sp) # salvo fp en 24
sw ra, 32(sp) # salvo ra en 32
move $fp, sp # a partir de acá trabajo con fp

# me guardo los parámetros (por convención de ABI)
sw a0, 40($fp) # salvo el caracter

# inicia el programa
sub t4, a0, 45 # analizo si es el guion medio
beq t4, zero, _noEs
sub t4, a0, 95 # analizo si es el guion bajo
beq t4, zero, _noEs
sub t4, a0, 48 # si es menor que 48 es un espacio
bltz t4, _Es
sub t4, a0, 122 # si es mayor que 122 es un espacio
bgtz t4, _Es
sub t4, a0, 57 # si es menor o igual que 57 seguro no es espacio [0,...,9]
blez t4, _noEs
sub t4, a0, 65 # si es menor a 65 seguro es espacio
bltz t4, _Es
sub t4, a0, 90 # si es menor o igual a 90 seguro no es espacio [A,...,Z]
blez t4, _noEs
sub t4, a0, 97 # si es menor que 97 seguro es espacio
bltz t4, _Es
b _noEs # esta entre 97 y 122 [a,...,z]
_noEs:
li v0, 0
b _return
_Es:
li v0, 1
b _return

_return:
lw ra, 32(sp)
lw $fp, 24(sp)
lw gp, 28(sp)
addu sp, sp, 40
jr ra

.end esEspacio

```



## 5. Comandos y corrida de archivos de pruebas

Con el fin de probar la robustez del programa se realizaron distintos archivos de prueba para ver si soportaba cualquier tipo de archivo de texto plano que se deseara introducir.

- Probamos con un archivo creado por nosotros, con algunos palíndromos:

```
~/ tp1 -i inputFile -I x
```

```
arribalabirra
a
fedef
MaLaM
vasoosav
liil
nico-ocin
1551
0330
```

- Con un archivo que no tenga salto de línea, y lo suficientemente largo (la salida se debe a que las palabras con tilde no están incluidas en el enunciado, por lo que corta la palabra cada vez que ve una palabra con tilde):

```
~/ tp1 -i sin_salto_de_linea -I x
```

```
r
y
s
f
y
0
y
n
y
y
narran
y
n
y
a
y
m
n
o
y
y
n
```

s  
o  
S  
a  
o  
n  
m  
s  
n  
y  
o  
y  
sus  
a  
y  
n  
n  
n  
o  
sus  
n  
a  
t  
o  
m  
s  
e  
m  
s  
n  
y  
a  
n  
y  
n  
y  
y  
y  
a  
y  
y  
o  
y  
b  
y  
m  
s  
y  
y

b  
s  
a  
y  
y  
y  
Y  
o  
a  
y  
a  
p  
l  
p  
q  
i  
l  
t  
n  
m  
s  
o  
2  
t  
c  
s  
n  
a  
o  
y  
y  
y  
m  
s  
y

- Con un archivo sin texto:

```
~/ tp1 -i nada -I x
```

- Luego se probó con un archivo con palabras largas (800 caracteres):

```
~/ tp1 -i palabrasLargas.txt -I x  
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa  
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa  
bbbbbb  
cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc  
cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
```

