Trabajo Práctico Nro. 1: programación MIPS

Lucas Verón, *Padrón Nro. 89.341*
lucasveron86@gmail.com

Eliana Diaz, *Padrón Nro. 89.324*
diazeliana09@gmail.com

Alan Helouani, *Padrón Nro. 90.289*
alanhelouani@gmail.com

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

Resumen

El presente proyecto tiene por finalidad familiarizarnos con el conjunto de instrucciones MIPS y el concepto de ABI

1. Introducción

Se detallará el diseño e implementación de un programa en lenguaje C y MIPS que procesa archivos de texto por línea de comando, como así también la forma de ejecución del mismo y los resultados obtenidos en las distintas pruebas ejecutadas.

El programa recibe los archivos o streams de entrada y salida, e imprime aquellas palabras del archivo de entrada (componentes léxicos) que sean palíndromos.

Se define como palabra a aquellos componentes léxicos del stream de entrada compuestos exclusivamente por combinaciones de caracteres a-z, 0-9, - (signo menos) y _ (*guiónbajo*).

Por otro lado, se considera que una palabra, número o frase, es *palíndroma* cuando se lee igual hacía adelante que hacía atrás.

Se implementará una función "palindrome"la cual se encargará de verificar si efectivamente la palabra es o no palindroma. La función estará escrita en assembly MIPS.

Los streams serán leídos y escritos de a bloques de memoría configurables, los cuales serán almacenados en un "buffer"para luego ser leídos de a uno.

2. Diseño

Las funcionalidades requeridas son las siguientes:

- Ayuda (Help): Presentación un detalle de los comandos que se pueden ejecutar.
- Versión: Se debe indicar la versión del programa.
- Procesar los datos:
  - Con especificación sólo del archivo de entrada.
  - Con especificación sólo del archivo de salida.
  - Con especificación del archivo de entrada y de salida.
  - Sin especificación del archivo de entrada ni de salida.
- Setting del tamaño del buffer in y buffer out; indicando de a cuantos caracteres se debe leer y escribir.

En base a estas funcionalidades, se modularizó el código a fin de poder reutilizarlo y a su vez que cada método se encargue de ejecutar una única funcionalidad.

3. Implementación

3.1. Código fuente en lenguaje C

```
/*
 ============================================================================

 Name        : tp1.c
 Author      : Grupo orga 66.20
 Version     : 1
 Copyright   : Orga6620 - Tp1
 Description : Trabajo practico 1: Programacion MIPS
 ============================================================================

 */

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <getopt.h>
#include <unistd.h>
#include "process.h"

#define VERSION "1.0"

#define FALSE 0
#define TRUE 1

size_t ibytes = 1;
```

```
24 size_t obytes = 1;
25
26 enum ParameterState {
27          OKEY = 0, INCORRECT_QUANTITY_PARAMS = 1,
                INCORRECT_MENU = 2, ERROR_FILE = 3, ERROR_BYTES
                = 6
28 };
29
30 int executeHelp() {
31         fprintf(stdout, "Usage: \n");
32         fprintf(stdout, "       tp1 -h \n");
33         fprintf(stdout, "       tp1 -V \n");
34         fprintf(stdout, "       tp1 [options] \n");
35         fprintf(stdout, "Options: \n");
36         fprintf(stdout, "       -V, --version
               Print version and quit. \n");
37         fprintf(stdout, "       -h, --help
               Print this information. \n");
38         fprintf(stdout, "       -i, --input
               Location of the input file. \n");
39         fprintf(stdout, "       -o, --output
               Location of the output file. \n");
40         fprintf(stdout, "       -I, --ibuf-bytes        Byte
               -count of the input buffer. \n");
41         fprintf(stdout, "       -O, --obuf-bytes        Byte
               -count of the output buffer. \n");
42         fprintf(stdout, "Examples: \n");
43         fprintf(stdout, "       tp1 -i ~/input -o ~/output \
               n");
44
45         return OKEY;
46 }
47
48 int executeVersion() {
49         fprintf(stdout, "Version: \"%s\" \n", VERSION);
50
51         return OKEY;
52 }
53
54 int executeWithDefaultParameter(char * path, int
       isInputDefault, int isOutputDefault) {
55         FILE * fileInput = NULL;
56         FILE * fileOutput = NULL;
57
58         if (isInputDefault == TRUE && isOutputDefault ==
               TRUE) {
59                 fileInput = stdin;
60                 fileOutput = stdout;
61         } else {
62                 if (isInputDefault == TRUE) {
63                         fileInput = stdin;
64
65                         fileOutput = fopen(path, "w"); //
                               Opens a text file for writing.
```

```
                        Pace the content.
                    if (fileOutput == NULL) {
                        fprintf(stderr, "[Error] El
                            archivo de output no pudo
                             ser abierto para
                            escritura: %s \n", path);
                        return ERROR_FILE;
                    }
                } else {
                    fileInput = fopen(path, "r"); //
                        Opens an existing text file for
                        reading purpose.
                    if (fileInput == NULL) {
                        fprintf(stderr, "[Error] El
                            archivo de input no pudo
                            ser abierto para lectura:
                             %s \n", path);
                        return ERROR_FILE;
                    }

                    fileOutput = stdout;
                }
            }

            int ifd = fileno(fileInput);
            int ofd = fileno(fileOutput);

            int executeResult = palindrome(ifd, ibytes, ofd,
                obytes);

            if (isInputDefault == FALSE || isOutputDefault ==
                FALSE) {
                if (isInputDefault == TRUE) {
                    if (fileOutput != NULL) {
                        int result = fclose(
                            fileOutput);
                        if (result == EOF) {
                            fprintf(stderr, "[
                                Warning] El
                                archivo de output
                                 no pudo ser
                                cerrado
                                correctamente: %s
                                 \n", path);
                            return ERROR_FILE;
                        }
                    }
                } else {
                    if (fileInput != NULL) {
                        int result = fclose(
                            fileInput);
                        if (result == EOF) {
                            fprintf(stderr, "[
                                Warning] El
```

```
                                                archivo de input
                                                no pudo ser
                                                cerrado
                                                correctamente: %s
                                                 \n", path);
100                                         return ERROR_FILE;
101                                 }
102                         }
103                 }
104         }
105
106         return executeResult;
107 }
108
109 int executeWithParameters(char * pathInput, char *
        pathOutput) {
110         FILE * fileInput = fopen(pathInput, "r"); // Opens
                an existing text file for reading purpose.
111         if (fileInput == NULL) {
112                 fprintf(stderr, "[Error] El archivo de input
                         no pudo ser abierto para lectura: %s \n"
                        , pathInput);
113                 return ERROR_FILE;
114         }
115
116         FILE * fileOutput = fopen(pathOutput, "w"); // Opens
                 a text file for writing. Pace the content.
117         if (fileOutput == NULL) {
118                 fprintf(stderr, "[Error] El archivo de
                        output no pudo ser abierto para escritura
                        : %s \n", pathOutput);
119
120                 int result = fclose(fileInput);
121                 if (result == EOF) {
122                         fprintf(stderr, "[Warning] El
                                archivo de input no pudo ser
                                cerrado correctamente: %s \n",
                                pathInput);
123                 }
124
125                 return ERROR_FILE;
126         }
127
128         int ifd = fileno(fileInput);
129         int ofd = fileno(fileOutput);
130
131         int executeResult = palindrome(ifd, ibytes, ofd,
                obytes);
132
133         int resultFileInputClose = 0; // EOF = -1
134         if (fileInput != NULL) {
135                 resultFileInputClose = fclose(fileInput);
136                 if (resultFileInputClose == EOF) {
```

```
137                         fprintf(stderr, "[Warning] El
                                archivo de input no pudo ser
                                cerrado correctamente: %s \n",
                                pathInput);
138                 }
139         }
140
141         if (fileOutput != NULL) {
142                 int result = fclose(fileOutput);
143                 if (result == EOF) {
144                         fprintf(stderr, "[Warning] El
                                archivo de output no pudo ser
                                cerrado correctamente: %s \n",
                                pathOutput);
145                         return ERROR_FILE;
146                 }
147         }
148
149         if (resultFileInputClose) {
150                 return ERROR_FILE;
151         }
152
153         return executeResult;
154 }
155
156 int executeByMenu(int argc, char **argv) {
157         // Always begins with /
158         if (argc == 1) {
159                 // Run with default parameters
160                 return executeWithDefaultParameter(NULL,
                        TRUE, TRUE);
161         }
162
163         char * inputValue = NULL;
164         char * outputValue = NULL;
165         char * iBufBytes = NULL;
166         char * oBufBytes = NULL;
167
168         /* Una cadena que lista las opciones cortas validas
                */
169         const char* const smallOptions = "Vhi:o:I:O:";
170
171         /* Una estructura de varios arrays describiendo los
                valores largos */
172         const struct option longOptions[] = {
173                 {"version",             no_argument,
                                0,  'V' },
174                 {"help",                no_argument,
                                        0,  'h' },
175                 {"input",               required_argument,
                        0,  'i' }, // optional_argument
176                 {"output",              required_argument,
                        0,  'o' },
```

```
177                 {"ibuf-bytes",  required_argument,      0,
                        'I' },
178                 {"obuf-bytes",  required_argument,      0,
                        'O' },
179                 {0,                     0,
                                             0,    0  }
180         };
181
182         int incorrectOption = FALSE;
183         int finish = FALSE;
184         int result = OKEY;
185         int longIndex = 0;
186         char opt = 0;
187
188         while ((opt = getopt_long(argc, argv, smallOptions,
189                                   longOptions, &longIndex )
                                       ) != -1 &&
                                       incorrectOption ==
                                       FALSE && finish ==
                                       FALSE) {
190                 switch (opt) {
191                         case 'V' :
192                                 result = executeVersion();
193                                 finish = TRUE;
194                                 break;
195                         case 'h' :
196                                 result = executeHelp();
197                                 finish = TRUE;
198                                 break;
199                         case 'i' :
200                                 inputValue = optarg;
201                                 break;
202                         case 'o' :
203                                 outputValue = optarg;
204                                 break;
205                         case 'I' :
206                                 iBufBytes = optarg;
207                                 break;
208                         case 'O' :
209                                 oBufBytes = optarg;
210                                 break;
211                         default:
212                                 incorrectOption = TRUE;
213                 }
214         }
215
216         if (incorrectOption == TRUE) {
217                 fprintf(stderr, "[Error] Incorrecta option
                        de menu.\n");
218                 return INCORRECT_MENU;
219         }
220
221         if (finish == TRUE) {
222                 return result;
```

```
223         }
224
225         if (iBufBytes != NULL) {
226                 char *finalPtr;
227                 ibytes = strtoul(iBufBytes, &finalPtr, 10);
228                 if (ibytes == 0) {
229                         fprintf(stderr, "[Error] Incorrecta
                            cantidad de bytes para el buffer
                            de entrada.\n");
230                         return ERROR_BYTES;
231                 }
232         }
233
234         if (oBufBytes != NULL) {
235                 char *finalPtr;
236                 obytes = strtoul(oBufBytes, &finalPtr, 10);
237                 if (obytes == 0) {
238                         fprintf(stderr, "[Error] Incorrecta
                            cantidad de bytes para el buffer
                            de salida.\n");
239                         return ERROR_BYTES;
240                 }
241         }
242
243         if (inputValue == NULL && outputValue == NULL) {
244                 return executeWithDefaultParameter(NULL,
                        TRUE, TRUE);
245         }
246
247         // / -i fileInput
248         if (inputValue != NULL && outputValue == NULL) {
249                 if (strcmp("-",inputValue) == 0) {
250                         return executeWithDefaultParameter(
                                NULL, TRUE, TRUE);
251                 } else {
252                         return executeWithDefaultParameter(
                                inputValue, FALSE, TRUE);
253                 }
254         }
255
256         // / -o fileOutput
257         if (inputValue == NULL && outputValue != NULL) {
258                 if (strcmp("-",outputValue) == 0) {
259                         return executeWithDefaultParameter(
                                NULL, TRUE, TRUE);
260                 } else {
261                         return executeWithDefaultParameter(
                                outputValue, TRUE, FALSE);
262                 }
263         }
264
265         if (inputValue != NULL && outputValue != NULL) {
266                 if (strcmp("-",inputValue) == 0 && strcmp("-
                        ",outputValue) == 0) {
```

```
267                         return executeWithDefaultParameter(
                                NULL, TRUE, TRUE);
268                 }
269
270                 if (strcmp("-",inputValue) == 0 && strcmp("-
                        ",outputValue) != 0) {
271                         return executeWithDefaultParameter(
                                outputValue, TRUE, FALSE);
272                 }
273
274                 if (strcmp("-",inputValue) != 0 && strcmp("-
                        ",outputValue) == 0) {
275                         return executeWithDefaultParameter(
                                inputValue, FALSE, TRUE);
276                 }
277
278                 return executeWithParameters(inputValue,
                        outputValue);
279         }
280
281         fprintf(stderr, "[Error] Incorrecta option de menu.\
                n");
282         return INCORRECT_MENU;
283 }
284
285 int main(int argc, char **argv) {
286         // / -i lalala.txt -o pepe.txt -I 2 -O 3 => 9
                parameters as maximum
287         if (argc > 9) {
288                 fprintf(stderr, "[Error] Cantidad máxima de
                        parámetros incorrecta: %d \n", argc);
289                 return INCORRECT_QUANTITY_PARAMS;
290         }
291
292         return executeByMenu(argc, argv);
293 }
```

3.2. Código MIPS32 (process.S)

```
1  #include <mips/regdef.h>
2  #include <sys/syscall.h>
3
4  #STATICS VAR DEFINITIONS FUNCTION PALINDROME
5
6  #define FALSE                   0
7  #define TRUE                    1
8  #define LEXICO_BUFFER_SIZE      10
9  #define DIR_NULL                0
10 #define FILE_DESCRIPTOR_STDERR  2
11 #define LINE_BREAK              10
12
13 # Resultados de funciones posibles
```

```
#define OKEY                    0
#define ERROR_MEMORY    2
#define ERROR_READ              3
#define ERROR_WRITE             4
#define LOAD_I_BUFFER   5

# Size mensajes
#define BYTES_MENSAJE_ERROR_MEMORIA_LEXICO      45
#define BYTES_MENSAJE_ERROR_MEMORIA_OBUFFER     60
#define BYTES_MENSAJE_ERROR_MEMORIA_IBUFFER 60
#define BYTES_MENSAJE_ERROR_MEMORIA_AMOUNT_SAVED 64
#define BYTES_MENSAJE_ERROR_LECTURA_ARCHIVO     60

# Para mymalloc y myfree

#define MYMALLOC_SIGNATURE 0xdeadbeef

#ifndef PROT_READ
#define PROT_READ 0x01
#endif

#ifndef PROT_WRITE
#define PROT_WRITE 0x02
#endif

#ifndef MAP_PRIVATE
#define MAP_PRIVATE 0x02
#endif

#ifndef MAP_ANON
#define MAP_ANON 0x1000
#endif


##----- mymalloc -----##

        .text
        .align  2
        .globl  mymalloc
        .ent    mymalloc
mymalloc:
        subu    sp, sp, 56
        sw      ra, 48(sp)
        sw      $fp, 44(sp)
        sw      a0, 40(sp)  # Temporary: original allocation
             size.
        sw      a0, 36(sp)  # Temporary: actual allocation
            size.
        li      t0, -1
        sw      t0, 32(sp)  # Temporary: return value (
            defaults to -1).
#if 0
        sw      a0, 28(sp)  # Argument building area (#8?).
        sw      a0, 24(sp)  # Argument building area (#7?).
```

```
65          sw      a0, 20(sp)  # Argument building area (#6).
66          sw      a0, 16(sp)  # Argument building area (#5).
67          sw      a0, 12(sp)  # Argument building area (#4, a3
                ).
68          sw      a0,  8(sp)  # Argument building area (#3, a2
                ).
69          sw      a0,  4(sp)  # Argument building area (#2, a1
                ).
70          sw      a0,  0(sp)  # Argument building area (#1, a0
                ).
71  #endif
72          move    $fp, sp
73
74          # Adjust the original allocation size to a 4-byte
                boundary.
75          #
76          lw      t0, 40(sp)
77          addiu   t0, t0, 3
78          and     t0, t0, 0xfffffffc
79          sw      t0, 40(sp)
80
81          # Increment the allocation size by 12 units, in
                order to
82          # make room for the allocation signature, block size
                 and
83          # trailer information.
84          #
85          lw      t0, 40(sp)
86          addiu   t0, t0, 12
87          sw      t0, 36(sp)
88
89          # mmap(0, sz, PROT_READ|PROT_WRITE, MAP_PRIVATE|
                MAP_ANON, -1, 0)
90          #
91          li      v0, SYS_mmap
92          li      a0, 0
93          lw      a1, 36(sp)
94          li      a2, PROT_READ|PROT_WRITE
95          li      a3, MAP_PRIVATE|MAP_ANON
96
97          # According to mmap(2), the file descriptor
98          # must be specified as -1 when using MAP_ANON.
99          #
100         li      t0, -1
101         sw      t0, 16(sp)
102
103         # Use a trivial offset.
104         #
105         li      t0, 0
106         sw      t0, 20(sp)
107
108         # XXX TODO.
109         #
110         sw      zero, 24(sp)
```

```
111         sw      zero, 28(sp)
112
113         # Excecute the syscall, save the return value.
114         #
115         syscall
116         sw      v0, 32(sp)
117         beqz    v0, mymalloc_return
118
119         # Success. Check out the allocated pointer.
120         #
121         lw      t0, 32(sp)
122         li      t1, MYMALLOC_SIGNATURE
123         sw      t1, 0(t0)
124
125         # The actual allocation size goes right after the
              signature.
126         #
127         lw      t0, 32(sp)
128         lw      t1, 36(sp)
129         sw      t1,  4(t0)
130
131         # Trailer information.
132         #
133         lw      t0, 36(sp) # t0: actual allocation size.
134         lw      t1, 32(sp) # t1: Pointer.
135         addu    t1, t1, t0 # t1 now points to the trailing
              4-byte area.
136         xor     t2, t0, MYMALLOC_SIGNATURE
137         sw      t2, -4(t1)
138
139         # Increment the result pointer.
140         #
141         lw      t0, 32(sp)
142         addiu   t0, t0, 8
143         sw      t0, 32(sp)
144
145 mymalloc_return:
146         # Restore the return value.
147         #
148         lw      v0, 32(sp)
149
150         # Destroy the stack frame.
151         #
152         move    sp, $fp
153         lw      ra, 48(sp)
154         lw      $fp, 44(sp)
155         addu    sp, sp, 56
156
157         j       ra
158         .end    mymalloc
159
160
161 ##----- myfree -----##
162
```

```
163         .globl  myfree
164         .ent    myfree
165 myfree:
166         subu    sp, sp, 40
167         sw      ra, 32(sp)
168         sw      $fp, 28(sp)
169         sw      a0, 24(sp)  # Temporary: argument pointer.
170         sw      a0, 20(sp)  # Temporary: actual mmap(2)
                pointer.
171         move    $fp, sp
172
173         # Calculate the actual mmap(2) pointer.
174         #
175         lw      t0, 24(sp)
176         subu    t0, t0, 8
177         sw      t0, 20(sp)
178
179         # XXX Sanity check: the argument pointer must be
                checked
180         # in before we try to release the memory block.
181         #
182         # First, check the allocation signature.
183         #
184         lw      t0, 20(sp) # t0: actual mmap(2) pointer.
185         lw      t1, 0(t0)
186         bne     t1, MYMALLOC_SIGNATURE, myfree_die
187
188         # Second, check the memory block trailer.
189         #
190         lw      t0, 20(sp) # t0: actual mmap(2) pointer.
191         lw      t1, 4(t0)  # t1: actual mmap(2) block size.
192         addu    t2, t0, t1 # t2: trailer pointer.
193         lw      t3, -4(t2)
194         xor     t3, t3, t1
195         bne     t3, MYMALLOC_SIGNATURE, myfree_die
196
197         # All checks passed. Try to free this memory area.
198         #
199         li      v0, SYS_munmap
200         lw      a0, 20(sp) # a0: actual mmap(2) pointer.
201         lw      a1, 4(a0)  # a1: actual allocation size.
202         syscall
203
204         # Bail out if we cannot unmap this memory block.
205         #
206         bnez    v0, myfree_die
207
208         # Success.
209         #
210         j myfree_return
211
212 myfree_die:
213         # Generate a segmentation fault by writing to the
                first
```

```
214         # byte of the address space (a.k.a. the NULL pointer
                ).
215         #
216         sw t0, 0(zero)
217
218 myfree_return:
219         # Destroy the stack frame.
220         #
221         move    sp, $fp
222         lw      ra, 32(sp)
223         lw      $fp, 28(sp)
224         addu    sp, sp, 40
225
226         j       ra
227         .end    myfree
228
229
230 ##----- toLowerCase -----##
231
232         .align          2
233         .globl          toLowerCase
234         .ent            toLowerCase
235 toLowerCase:
236         .frame          $fp,24,ra
237         .set            noreorder
238         .cpload         t9
239         .set            reorder
240
241         #Stack frame creation
242         subu            sp,sp,24
243
244         .cprestore 0
245         sw              $fp,20(sp)
246         sw              gp,16(sp)
247         move            $fp,sp
248
249         move            v0,a0
                        # word (this is the character)
250         sb              v0,8($fp)
251         lb              v0,8($fp)
252         slt             v0,v0,65
253         bne             v0,zero,$IfNotLower
                        # if !(word >= 65) goto IfNotLower
254         lb              v0,8($fp)
255         slt             v0,v0,91
256         beq             v0,zero,$IfNotLower
                        # if !(word <= 90) goto IfNotLower
257         lbu             v0,8($fp)
258         addu            v0,v0,32
                        # word += 32
259         sb              v0,8($fp)
260 $IfNotLower:
261         lb              v0,8($fp)
262         move            sp,$fp
```

```
263
264         #Stack frame destruction.
265         lw              $fp,20(sp)
266         addu            sp,sp,24
267         j               ra
                        # Jump and return
268         .end            toLowerCase
269
270
271 ##----- verifyPalindromic -----##
272
273         .align          2
274         .globl          verifyPalindromic
275         .ent            verifyPalindromic
276 verifyPalindromic:
277         .frame          $fp,72,ra
278         .set            noreorder
279         .cpload         t9
280         .set            reorder
281
282         #Stack frame creation
283         subu            sp,sp,72
284
285         .cprestore      16
286         sw              ra,64(sp)
287         sw              $fp,60(sp)
288         sw              gp,56(sp)
289         move            $fp,sp
290
291         sw              a0,72($fp)
                        # char * word
292         sw              a1,76($fp)
                        # int quantityCharacterInWord
293
294         lw              v0,72($fp)
295         beq             v0,zero,$IfPalindromicFalse
                        # if (word == NULL) goto
                    IfPalindromicFalse
296         lw              v0,76($fp)
297         blez            v0,$IfPalindromicFalse
                        # if (quantityCharacterInWord <= 0) goto
                    IfPalindromicFalse
298         b               $VerifyWhenOneCharacter
                        # Salta siempre - goto
                    VerifyWhenOneCharacter
299 $IfPalindromicFalse:
300         sw              zero,52($fp)
                                # Guardo FALSE (= 0)
301         b               $ReturnVerifyPalindromic
                        # Salta siempre - goto
                    ReturnVerifyPalindromic (con return FALSE)
302 $VerifyWhenOneCharacter:
303         lw              v1,76($fp)
                        # Cargo quantityCharacterInWord
```

```
304         li              v0,1
                        # Cargo en v0, el valor 1, para luego
            hacer la comparacion
305         bne             v1,v0,$VerifyWhenTwoCharacteres # if
             (quantityCharacterInWord != 1) goto
            VerifyWhenTwoCharacteres
306         li              v0,TRUE
                        # Cargo resultado (TRUE es igual a 1)
307         sw              v0,52($fp)
308         b               $ReturnVerifyPalindromic
                        # Salta siempre - goto
            ReturnVerifyPalindromic (con return TRUE)
309 $VerifyWhenTwoCharacteres:
310         lw              v1,76($fp)
                        # Cargo quantityCharacterInWord
311         li              v0,2
                        # Cargo en v0, el valor 2, para luego
            hacer la comparacion
312         bne             v1,v0,
            $VerifyWhenMoreThanOneCharacter   # if (
            quantityCharacterInWord != 2) goto
            VerifyWhenMoreThanOneCharacter
313
314         # Paso a minuscula el primer caracter del lexico
315         lw              v0,72($fp)
                        # Cargo * word
316         lb              v0,0(v0)
                        # Cargo el primer caracter apuntado por
            word
317         move            a0,v0
                        # Cargo el primer caracter que estaba en
            v0, en a0. Voy a enviarlo por parametro a la
            funcion toLoweCase
318         la              t9,toLowerCase
                        # Cargo la direccion de la funcion
            toLowerCase
319         jal             ra,t9
                        # Salto a la funcion toLowerCase
320         sb              v0,24($fp)
                        # Cargo el resultado en v0 en 24($fp).
321
322         # Paso a minuscula el segundo caracter del lexico
323         lw              v0,72($fp)
                        # Cargo la direccion de memoria en donde
            esta word
324         addu            v0,v0,1
                        # Sumo uno a la direccion de memoria, me
            corro un lugar.
325         lb              v0,0(v0)
                        # Cargo el segundo caracter apuntado por
            word (solo habian dos caracteres)
326         move            a0,v0
                        # Cargo el segundo caracter que estaba en
             v0, en a0. Voy a enviarlo por parametro a la
```

```
            funcion toLoweCase
327         la              t9,toLowerCase
                        # Cargo la direccion de la funcion
            toLowerCase
328         jal             ra,t9
                        # Salto a la funcion toLowerCase
329         sb              v0,25($fp)
                        # Cargo el resultado en v0 en 25($fp).
330
331         lb              v1,24($fp)
                        # Cargo el primer caracter en minuscula
            en v1
332         lb              v0,25($fp)
                        # Cargo el segundo caracter en minuscula
            en v0
333         beq             v1,v0,$IfPalindromicTrue
                        # if (firstCharacter == lastCharacter)
            goto IfPalindromicTrue
334         sw              zero,52($fp)                    #
            Guardo FALSE (= 0)
335         b               $ReturnVerifyPalindromic
                        # Salta siempre - goto
            ReturnVerifyPalindromic (con return FALSE)
336 $IfPalindromicTrue:
337         li              v0,TRUE
                        # TRUE es igual a 1
338         sw              v0,52($fp)
339         b               $ReturnVerifyPalindromic
                        # Salta siempre - goto
            ReturnVerifyPalindromic (con return TRUE)
340 $VerifyWhenMoreThanOneCharacter:
341         l.s             $f0,76($fp)
                        # Cargo quantityCharacterInWord
342         cvt.d.w $f2,$f0                                 #
            Convierto el integer quantityCharacterInWord a
            double
343         l.d             $f0,doubleWord                  #
            Cargo en f0 el valor 2.
344         div.d   $f0,$f2,$f0                             #
            Division con Double (double)
            quantityCharacterInWord / 2; - Sintaxis: div.d
            FRdest, FRsrc1, FRsrc2
345         s.d             $f0,32($fp)
                        # Guarda el resultado de la division en
            32($fp). O sea, middle (double middle = (double)
            quantityCharacterInWord / 2;)
346         sw              zero,40($fp)                    # En
             40($fp) se encuentra idx (int idx = 0;).
347         li              v0,TRUE
                        # En v0 esta la variable validPalindromic
             en TRUE, que es igual a 1 (int validPalindromic
            = TRUE;).
348         sw              v0,44($fp)
                        # Guarda en la direccion 44($fp) el valor
```

```
             de validPalindromic.
349         lw              v0,76($fp)
                        # Cargo quantityCharacterInWord en v0.
350         addu            v0,v0,-1
                        # Le resto 1 a quantityCharacterInWord y
                    lo guardo en v0 (int last =
                    quantityCharacterInWord - 1;).
351         sw              v0,48($fp)
                        # Guardo en la direccion 48($fp) la
                    variable last.
352 $WhileMirror:
353         l.s             $f0,40($fp)
                        # Cargo idx en f0.
354         cvt.d.w $f2,$f0                                 #
                    Convierto el integer idx a double y lo guardo en
                    f2 para poder hacer la comparacion.
355         l.d             $f0,32($fp)
                        # Cargo en a0 la variable middle.
356         c.lt.d  $f2,$f0                                 #
                    Compara la variable idx con la variable middle, y
                     setea el condition flag en true si el primero (
                    idx) es mas chico que el segundo (middle).
357         bc1t    $WhileMirrorConditionLastWithMiddle     # Si
                     el condition flag es true, continua haciendo las
                     comparaciones.
358         b               $WhileMirrorFinalized
                        # Si el condition flag es false, salta al
                     final de la funcion, devolviendo el valor de la
                    variable validPalindromic que seria TRUE.
359 $WhileMirrorConditionLastWithMiddle:
360         l.s             $f0,48($fp)
                        # Cargo la variable last en f0.
361         cvt.d.w $f2,$f0                                 #
                    Convierto el integer last a double y lo guardo en
                     f2 para poder hacer la comparacion.
362         l.d             $f0,32($fp)
                        # Cargo en f0 el contenido de la variable
                     middle.
363         c.lt.d  $f0,$f2                                 #
                    Compara el contenido de la variable last con la
                    variable middle, y setea el condition flag en
                    true si
364 # el primero (last) es mas chico que el segundo (middle).
365         bc1t    $WhileMirrorConditionValidPalindromicTrue
                       # Si el condition flag es true, continua
                    haciendo las comparaciones.
366         b               $WhileMirrorFinalized
                        # Si el condition flag es false, salta al
                     final de la funcion, devolviendo el valor de la
                    variable validPalindromic que seria TRUE.
367 $WhileMirrorConditionValidPalindromicTrue:
368         lw              v1,44($fp)
                        # Cargo el contenido de la variable
                    validPalindromic, que esta en la direccion 44($fp
```

```
            ), en v1.
369         li              v0,TRUE
                        # Cargo TRUE (que seria 1) en v0.
370         beq             v1,v0,$WhileMirrorContent
                        # If validPalindromic == TRUE goto
                    WhileMirrorContent (entro al while).
371         b               $WhileMirrorFinalized
                        # Salto para salir del while (bucle).
372 $WhileMirrorContent:
373         # Voy a pasar a minuscula el caracter apuntado desde
                 la izquierda.
374         lw              v1,72($fp)
                        # Cargo en v1 el contenido de la variable
                 * word.
375         lw              v0,40($fp)
                        # Cargo en v0 el contenido de la variable
                 idx.
376         addu            v0,v1,v0
                        # En v0 coloco el puntero a word corrido
                    la cantidad indicada por la variable idx.
377         lb              v0,0(v0)
                        # Cargo en v0 el contenido de la
                    direccion de memoria (cero corrimiento).
378         move            a0,v0
                        # Paso a a0 el contenido de v0, que seria
                 un unico caracter.
379         la              t9,toLowerCase
                        # Cargo la direccion de la funcion
                    toLowerCase
380         jal             ra,t9
                        # Salto a la funcion toLowerCase para
                    pasar el caracter a minuscula.
381         sb              v0,25($fp)
                        # Cargo el caracter contenido en v0 a la
                    direccion de memoria 25($fp) - char
                    firstCharacter = toLowerCase(word[idx]);
382
383         # Voy a pasar a minuscula el caracter apuntado desde
                 la derecha.
384         lw              v1,72($fp)
                        # Cargo en v1 el contenido de la variable
                 * word.
385         lw              v0,48($fp)
                        # Cargo en v0 el contenido de la variable
                 last.
386         addu            v0,v1,v0
                        # En v0 coloco el puntero a word corrido
                    la cantidad indicada por la variable last.
387         lb              v0,0(v0)
                        # Cargo en v0 el contenido de la
                    direccion de memoria (cero corrimiento).
388         move            a0,v0
                        # Paso a a0 el contenido de v0, que seria
                 un unico caracter.
```

```
389         la              t9,toLowerCase
                        # Cargo la direccion de la funcion
                toLowerCase
390         jal             ra,t9
                        # Salto a la funcion toLowerCase para
                pasar el caracter a minuscula.
391         sb              v0,24($fp)
                        # Cargo el caracter contenido en v0 a la
                direccion de memoria 24($fp) - char lastCharacter
                 = toLowerCase(word[last]);
392
393         lb              v1,25($fp)
                        # Cargo en v1 el contenido de la variable
                 firstCharacter.
394         lb              v0,24($fp)
                        # Cargo en v1 el contenido de la variable
                 lastCharacter.
395         beq             v1,v0,$ContinuedInWhileMirror
                        # If (firstCharacter == lastCharacter)
                goto ContinuedInWhileMirror
396         sw              zero,44($fp)
397 $ContinuedInWhileMirror:
398         lw              v0,40($fp)
                        # Cargo en v0 el contenido de la variable
                 idx.
399         addu            v0,v0,1
                        # Incremento en uno el valor de la
                variable idx (idx ++).
400         sw              v0,40($fp)
                        # Guardo el contenido de la variable idx
                en la direccion de memoria 40($fp).
401         lw              v0,48($fp)
                        # Cargo en v0 el contenido de la variable
                 last.
402         addu            v0,v0,-1
                        # Decremento en uno el valor de la
                variable last (last --).
403         sw              v0,48($fp)
                        # Guardo el contenido de la variable last
                 en la direccion de memoria 48($fp).
404         b               $WhileMirror
                        # Vuelvo a entrar en el bucle.
405 $WhileMirrorFinalized:
406         lw              v0,44($fp)
                        # Cargo en v0 el contenido de la variable
                 validPalindromic, que se encuentra en la
                direccion de memoria 44($fp).
407         sw              v0,52($fp)
                        # Guardo en la direccion de memoria 52(
                $fp) el resultado de la funcion verifyPalindromic
                .
408 $ReturnVerifyPalindromic:
409         lw              v0,52($fp)
410         move            sp,$fp
```

```
411         lw              ra,64(sp)
412         lw              $fp,60(sp)
413         addu            sp,sp,72
414         j               ra
                        # Jump and return
415         .end            verifyPalindromic
416
417
418 ##----- isKeywords -----##
419
420         .align          2
421         .globl          isKeywords
422         .ent            isKeywords
423 isKeywords:
424         .frame          $fp,24,ra
425         .set            noreorder
426         .cpload         t9
427         .set            reorder
428
429         #Stack frame creation
430         subu            sp,sp,24
431         .cprestore 0
432         sw              $fp,20(sp)
433         sw              gp,16(sp)
434         move            $fp,sp
435
436         move            v0,a0
                        # Muevo de a0 a v0 el parametro de la
                            funcion (char character).
437         sb              v0,8($fp)
                        # Guardo en la direccion de memoria 8($fp
                            ) el contenido de la variable character que se
                            encuentra en el registro v0.
438         lb              v0,8($fp)
                        # Cargo el byte character en v0 que
                            estaba en la direccion de memoria 8($fp).
439
440         # character >= 65 && character <= 90    ---   A - Z
                            = [65 - 90]
441         slt             v0,v0,65
                        # Compara el contenido de la variable
                            character con el literal 65, y guarda true en v0
                            si
442 # el primero (character) es mas chico que el segundo (65).
443         bne             v0,FALSE,$VerifyCharacterOfaToz
                        # Si no es igual a FALSE, o sea,
                            character < 65, salta a VerifyCharacterOfaToz.
444         lb              v0,8($fp)
                        # Cargo el byte character en v0 que
                            estaba en la direccion de memoria 8($fp).
445         slt             v0,v0,91
                        # Compara el contenido de la variable
                            character con el literal 91, y guarda true en v0
                            si el
```

```
446 # primero (character) es mas chico que el segundo (91).
447         bne             v0,FALSE,$ReturnIsKeywordsTrue
                        # Si no es igual a FALSE, o sea,
                    character > 91, salta a ReturnIsKeywordsTrue.
448 $VerifyCharacterOfaToz:
449         lb              v0,8($fp)
                        # Cargo el byte character en v0 que
                    estaba en la direccion de memoria 8($fp).
450
451         # character >= 97 && character <= 122    ---   a - z
                     = [97 - 122]
452         slt             v0,v0,97
                        # Compara el contenido de la variable
                    character con el literal 97, y guarda true en v0
                    si
453 # el primero (character) es mas chico que el segundo (97).
454         bne             v0,FALSE,$VerifyCharacterOf0To9
                        # Si no es igual a FALSE, o sea,
                    character < 65, salta a VerifyCharacterOf0To9.
455         lb              v0,8($fp)
                        # Cargo el byte character en v0 que
                    estaba en la direccion de memoria 8($fp).
456         slt             v0,v0,123
                        # Compara el contenido de la variable
                    character con el literal 123, y guarda true en v0
                     si el
457 # primero (character) es mas chico que el segundo (123).
458         bne             v0,FALSE,$ReturnIsKeywordsTrue
                        # Si no es igual a FALSE, o sea,
                    character > 123, salta a ReturnIsKeywordsTrue.
459 $VerifyCharacterOf0To9:
460         lb              v0,8($fp)
                        # Cargo el byte character en v0 que
                    estaba en la direccion de memoria 8($fp).
461
462         # character >= 48 && character <= 57    ---   0 - 9
                    = [48 - 57]
463         slt             v0,v0,48
                        # Compara el contenido de la variable
                    character con el literal 48, y guarda true en v0
                    si el
464 # primero (character) es mas chico que el segundo (48).
465         bne             v0,zero,$VerifyCharacterGuionMedio
                          # Si no es igual a FALSE, o sea,
                    character < 48, salta a VerifyCharacterGuionMedio
                    .
466         lb              v0,8($fp)
                        # Cargo el byte character en v0 que
                    estaba en la direccion de memoria 8($fp).
467         slt             v0,v0,58
                        # Compara el contenido de la variable
                    character con el literal 58, y guarda true en v0
                    si el
468 # primero (character) es mas chico que el segundo (58).
```

```
469         bne             v0,zero,$ReturnIsKeywordsTrue
                        # Si no es igual a FALSE, o sea,
                    character > 58, salta a ReturnIsKeywordsTrue.
470 $VerifyCharacterGuionMedio:
471         lb              v1,8($fp)
                        # Cargo el byte character en v1 que
                    estaba en la direccion de memoria 8($fp).
472
473         # character == 45    ---   - =  45
474         li              v0,45
                        # Cargo el literal 45 en v0 para hacer
                    luego la comparacion.
475         beq             v1,v0,$ReturnIsKeywordsTrue
                        # If (character == 45) goto
                    ReturnIsKeywordsTrue
476
477         lb              v1,8($fp)
                        # Cargo el byte character en v1 que
                    estaba en la direccion de memoria 8($fp).
478
479         # character == 95    ---   _ = 95
480         li              v0,95
                        # Cargo el literal 95 en v0 para hacer
                    luego la comparacion.
481         beq             v1,v0,$ReturnIsKeywordsTrue
                        # If (character == 95) goto
                    ReturnIsKeywordsTrue
482         b               $ReturnIsKeywordsFalse
                        # Salto incondicional para retornar FALSE
                     (character no es un keyword).
483 $ReturnIsKeywordsTrue:
484         li              v0,TRUE
                        # Cargo en v0 TRUE (que seria igual a 1).
485         sw              v0,12($fp)
                        # Guardo el resultado de la funcion TRUE
                    (v0) en la direccion de memoria 12($fp).
486         b               $ReturnIsKeywords
                        # Salto incondicional para retornar
                    resultado de las comparaciones.
487 $ReturnIsKeywordsFalse:
488         sw              zero,12($fp)
                        # Guardo FALSE (que seria igual a 0) en
                    la direccion de memoria 12($fp).
489 $ReturnIsKeywords:
490         lw              v0,12($fp)
                        # Cargo en v0 el resultado de la funcion
                    isKeywords guardado en la direccion de memoria
                    12($fp).
491         move            sp,$fp
492         lw              $fp,20(sp)
493         addu            sp,sp,24
494         j               ra
                        # Jump and return
495         .end            isKeywords
```

```
496
497
498 ##----- myRealloc -----##
499
500         .align          2
501         .globl          myRealloc
502         .ent            myRealloc
503 myRealloc:
504         .frame          $fp,64,ra
505         .set            noreorder
506         .cpload         t9
507         .set            reorder
508
509         #Stack frame creation
510         subu            sp,sp,64
511
512         .cprestore 16
513         sw              ra,56(sp)
514         sw              $fp,52(sp)
515         sw              gp,48(sp)
516         move            $fp,sp
517
518         # Parameters
519         sw              a0,64($fp)
                        # Guardo en la direccion de memoria 64(
                    $fp) la variable ptr (void * ptr).
520         sw              a1,68($fp)
                        # Guardo en la direccion de memoria 68(
                    $fp) la variable tamanyoNew (size_t tamanyoNew).
521         sw              a2,72($fp)
                        # Guardo en la direccion de memoria 72(
                    $fp) la variable tamanyoOld (int tamanyoOld).
522
523         lw              v0,68($fp)
                        # Cargo en v0 el contenido de la variable
                     tamanyoNew, que esta en la direccion de memoria
                    68($fp)
524         bne             v0,zero,
                    $MyReallocContinueValidations   # If (tamanyoNew
                    != 0) goto MyReallocContinueValidations
525
526         # If (tamanyoNew == 0)
527         lw              a0,64($fp)
                        # Cargo en a0 la direccion de memoria
                    guardada en la direccion 64($fp), o sea, la
                    variable * ptr.
528         la              t9,myfree
                                # Cargo la direccion de la
                    funcion myfree.
529         jal             ra,t9
                        # Ejecuto la funcion myfree.
530         sw              zero,64($fp)                    #
                    Coloco el puntero apuntando a NULL (ptr = NULL;).
```

```
531         sw              zero,40($fp)                    #
                Coloco en la direccion de memoria 40($fp) NULL,
                que seria el resultado de la funcion myRealloc.
532         b               $MyReallocReturn
                        # Salto incondicional para retornar
                resultado de myRealloc.
533 $MyReallocContinueValidations:
534         lw              a0,68($fp)
                        # Cargo en a0 el contenido guardado en la
                 direccion 68($fp), o sea, la variable tamanyoNew
                .
535         la              t9,mymalloc
                        # Cargo la direccion de la funcion
                mymalloc.
536         jal             ra,t9
                        # Ejecuto la funcion mymalloc.
537         sw              v0,24($fp)
                        # Guardo en la direccion 24($fp) el
                contenido de v0, que seria la direccion de la
                memoria asignada con mymalloc.
538         lw              v0,24($fp)
                        # Cargo en v0 la direccion de la memoria
                asignada con mymalloc (void * ptrNew = (void *)
                mymalloc(tamanyoNew);).
539
540         # (ptrNew == NULL) ?
541         bne             v0,DIR_NULL,
                $MyReallocContinueValidationsWithMemory # If (
                ptrNew != NULL) goto
                MyReallocContinueValidationsWithMemory
542         lw              a0,64($fp)
                        # Cargo en a0 la direccion de memoria
                guardada en la direccion 64($fp), o sea, la
                variable * ptr.
543         la              t9,myfree
                            # Cargo la direccion de la
                funcion myfree.
544         jal             ra,t9
                        # Ejecuto la funcion myfree.
545         sw              zero,64($fp)                    #
                Coloco el puntero apuntando a NULL (ptr = NULL;).
546         sw              zero,40($fp)                    #
                Coloco en la direccion de memoria 40($fp) NULL,
                que seria el resultado de la funcion myRealloc.
547         b               $MyReallocReturn
                        # Salto incondicional para retornar
                resultado de myRealloc.
548 $MyReallocContinueValidationsWithMemory:
549         lw              v0,64($fp)
                        # Cargo en v0 la direccion de memoria
                guardada en la direccion 64($fp), o sea, la
                variable * ptr.
550         bne             v0,DIR_NULL,
                $MyReallocContinueWithLoadCharacters # If (ptr !=
```

```
             NULL) goto MyReallocContinueWithLoadCharacters
551
552         # (ptr == NULL) ?
553         lw              v0,24($fp)
                        # Cargo en v0 la direccion de memoria
                    guardada en la direccion 24($fp), o sea, la
                    variable * ptrNew,
554 # que seria la direccion de la memoria asignada con mymalloc
        .
555         sw              v0,40($fp)
                        # Coloco en la direccion de memoria 40(
                    $fp) el contenido de v0 (* ptrNew), que seria el
                    resultado de la funcion myRealloc.
556         b               $MyReallocReturn
                        # Salto incondicional para retornar
                    resultado de myRealloc.
557 $MyReallocContinueWithLoadCharacters:
558         lw              v0,68($fp)
                        # Cargo en v0 el contenido guardado en la
                     direccion 68($fp), o sea, la variable tamanyoNew
                    .
559         sw              v0,28($fp)
                        # Guardo en la direccion de memoria 28(
                    $fp) la variable tamanyoNew guardada en v0 (int
                    end = tamanyoNew;).
560
561         lw              v1,72($fp)
                        # Cargo en v1 el contenido guardado en la
                     direccion 72($fp), o sea, la variable tamanyoOld
                    .
562         lw              v0,68($fp)
                        # Cargo en v0 el contenido guardado en la
                     direccion 68($fp), o sea, la variable tamanyoNew
                    , para poder luego hacer comparacion.
563
564         # (tamanyoOld < tamanyoNew) ?
565         sltu            v0,v1,v0
                        # Compara el contenido de la variable
                    tamanyoOld (v1) con tamanyoNew (v0), y guarda
                    true en v0 si
566 # el primero (tamanyoOld) es mas chico que el segundo (
    tamanyoNew).
567         beq             v0,FALSE,$MyReallocLoadCharacters
                      # If (tamanyoOld >= tamanyoNew) goto
                    MyReallocLoadCharacters
568         lw              v0,72($fp)
                        # Cargo en v0 el contenido guardado en la
                     direccion 72($fp), o sea, la variable tamanyoOld
                    .
569         sw              v0,28($fp)
                        # Guardo en la direccion 28($fp), que
                    seria la variable end, el contenido de la
                    variable tamanyoOld (end = tamanyoOld;).
570 $MyReallocLoadCharacters:
```

```
571         lw              v0,24($fp)
                    # Cargo en v0 el contenido guardado en la
                direccion 24($fp), o sea, la variable ptrNew.
572         sw              v0,32($fp)
                    # Guardo en la direccion de memoria 24(
               $fp) el contenido de v0 (char *tmp = ptrNew;).
573         lw              v0,64($fp)
                    # Cargo en v0 el contenido guardado en la
                direccion 64($fp), o sea, la variable ptr.
574         sw              v0,36($fp)
                    # Guardo en la direccion de memoria 36(
               $fp) el contenido de v0 (const char *src   = ptr
               ;).
575 $MyReallocWhileLoadCharacter:
576         lw              v0,28($fp)
                    # Cargo en v0 el contenido guardado en la
                direccion 28($fp), o sea, la variable end.
577         addu            v0,v0,-1
                    # Decremento en 1 el contenido de v0 (end
                --).
578         move            v1,v0
                    # Muevo el contenido de v0 a v1.
579         sw              v1,28($fp)
                    # Guardo en la direccion de memoria 28(
               $fp), que seria en donde estaba end, el nuevo
               valor de end (habia sido decrementado en 1).
580         li              v0,-1
                    # Cargo en v0 el literal -1.
581         bne             v1,v0,$MyReallocContinueWhileLoad
                   # If ( end != -1) goto
               MyReallocContinueWhileLoad.
582         b               $MyReallocFinalizedWhileLoad
                    # Salto incondicional fuera del while,
               porque la variable end es -1.
583 $MyReallocContinueWhileLoad:
584         # *tmp = *src;
585         lw              v1,32($fp)
                    # Cargo en v1 el contenido guardado en la
                direccion 32($fp), que seria *tmp.
586         lw              v0,36($fp)
                    # Cargo en v0 el contenido guardado en la
                direccion 36($fp), que seria *src.
587         lbu             v0,0(v0)
                    # Cargo la direccion de memoria en v0 de
               src.
588         sb              v0,0(v1)
                    # Guardo en la direccion apuntada por el
               contenido de v1, la direccion de memoria guardada
                en v0 (*tmp = *src;).
589
590         # tmp ++
591         lw              v0,32($fp)
                    # Cargo en v0 el contenido guardado en la
                direccion 32($fp), que seria *tmp.
```

```
592         addu            v0,v0,1
                        # Incremento en 1 el contenido guardado
                    en v0 (tmp ++).
593         sw              v0,32($fp)
                        # Guardo en la direccion de memoria 32(
                    $fp) lo que tenia v0 (el resultado de hacer tmp
                    ++).
594
595         # src ++
596         lw              v0,36($fp)
                        # Cargo en v0 el contenido guardado en la
                     direccion 36($fp), que seria *src.
597         addu            v0,v0,1
                        # Incremento en 1 el contenido guardado
                    en v0 (src ++).
598         sw              v0,36($fp)
                        # Guardo en la direccion de memoria 36(
                    $fp) lo que tenia v0 (el resultado de hacer src
                    ++).
599
600         b               $MyReallocWhileLoadCharacter
                           # Vuelvo a entrar al while
601 $MyReallocFinalizedWhileLoad:
602         lw              a0,64($fp)
                        # Cargo en v0 el contenido guardado en la
                     direccion 64($fp), que seria *PTR.
603         la              t9,myfree
                               # Cargo la direccion de la
                    funcion myfree.
604         jal             ra,t9
                        # Ejecuto la funcion myfree.
605         sw              zero,64($fp)                    #
                    Coloco el puntero apuntando a NULL (ptr = NULL;).
606
607         lw              v0,24($fp)
                        # Cargo en v0 la direccion de memoria
                    guardada en la direccion 24($fp), o sea, la
                    variable * ptrNew, que seria la direccion de la
                    memoria asignada con mymalloc..
608         sw              v0,40($fp)
                        # Guardo en la direccion de memoria 40(
                    $fp) el contenido de v0 (* ptrNew), que seria el
                    resultado de la funcion myRealloc.
609 $MyReallocReturn:
610         lw              v0,40($fp)
                        # Cargo en v0 el resultado de la funcion
                    myRealloc guardado en la direccion de memoria 40(
                    $fp).
611         move            sp,$fp
612         lw              ra,56(sp)
613         lw              $fp,52(sp)
614         addu            sp,sp,64
615         j               ra
                        # Jump and return
```

```
616         .end            myRealloc
617
618
619 ##----- initializeBuffer -----##
620
621         .align          2
622         .globl          initializeBuffer
623         .ent            initializeBuffer
624 initializeBuffer:
625         .frame          $fp,24,ra
626         .set            noreorder
627         .cpload         t9
628         .set            reorder
629
630         #Stack frame creation
631         subu            sp,sp,24
632
633         .cprestore 0
634         sw              $fp,20(sp)
635         sw              gp,16(sp)
636         move            $fp,sp
637
638         # Parameters
639         sw              a0,24($fp)
                        # Guardo en la direccion de memoria 24(
                    $fp) la variable bytes (size_t bytes).
640         sw              a1,28($fp)
                        # Guardo en la direccion de memoria 28(
                    $fp) la variable buffer (char * buffer).
641
642         sw              zero,8($fp)
                        # Guardo en la direccion de memoria 8($fp
                    ) el contenido 0, que seria la variable i (int i
                    ;).
643 $ForInitializeBuffer:
644         lw              v0,8($fp)
                        # Cargo en v0 el contenido en la
                    direccion de memoria 8($fp), que seria la
                    variable i.
645         lw              v1,24($fp)
                        # Cargo en v1 el contenido en la
                    direccion de memoria 24($fp), que seria la
                    variable bytes.
646         sltu            v0,v0,v1
                        # Comparo i (v0) con bytes (v1). Si i <
                    bytes, guardo TRUE en v0, sino guardo FALSE.
647         bne             v0,FALSE,$ForInitializeCharacter
                       # If (i < bytes) goto
                    ForInitializeCharacter.
648         b               $InitializeBufferReturn
                        # Salto incondicional al return de la
                    funcion initializeBuffer.
649 $ForInitializeCharacter:
650         # buffer[i] = '\0';
```

```
651         lw              v1,28($fp)
                    # Cargo en v1 el contenido en la
                direccion de memoria 28($fp), que seria la
                variable * buffer.
652         lw              v0,8($fp)
                    # Cargo en v0 el contenido en la
                direccion de memoria 8($fp), que seria la
                variable i.
653         addu            v0,v1,v0
                    # Me corro en el buffer la cantidad
                estipulada por la variable i (buffer[i] = buffer
                + i), y lo guardo en v0.
654         sb              zero,0(v0)
                    # Guardo '\0' = 0 en la posicion del
                buffer estipulada previamente (buffer[i] = '\0';)
                .
655
656         # ++ i
657         lw              v0,8($fp)
                    # Cargo en v0 el contenido en la
                direccion de memoria 8($fp), que seria la
                variable i.
658         addu            v0,v0,1
                    # Incremento en 1 la variable i (i ++).
659         sw              v0,8($fp)
                    # Guardo en la direccion de memoria 8($fp
                ) el nuevo valor de la variable i.
660
661         b               $ForInitializeBuffer
                    # Vuelvo a entrar en el for (bucle).
662 $InitializeBufferReturn:
663         move            sp,$fp
664         lw              $fp,20(sp)
665         addu            sp,sp,24
666         j               ra
                    # Jump and return
667         .end            initializeBuffer
668
669
670 ##----- writeOBufferInOFile -----##
671
672         .align          2
673         .globl          writeOBufferInOFile
674         .ent            writeOBufferInOFile
675 writeOBufferInOFile:
676         .frame          $fp,64,ra
677         .set            noreorder
678         .cpload         t9
679         .set            reorder
680
681         #Stack frame creation
682         subu            sp,sp,64
683
684         .cprestore 16
```

```
685         sw              ra,56(sp)
686         sw              $fp,52(sp)
687         sw              gp,48(sp)
688         move            $fp,sp
689
690         # Parameter
691         sw              a0,64($fp)
                        # Guardo en la direccion de memoria 64(
                    $fp) la variable * amountSavedInOBuffer (int *
                    amountSavedInOBuffer).
692
693         sw              zero,24($fp)
                        # Guardo en la direccion de memoria 24(
                    $fp) la variable completeDelivery inicializada en
                     FALSE (int completeDelivery = FALSE;).
694         sw              zero,28($fp)
                        # Guardo en la direccion de memoria 28(
                    $fp) la variable bytesWriteAcum inicializada en 0
                     (int bytesWriteAcum = 0;).
695
696         lw              v0,64($fp)
                        # Cargo en v0 el contenido de la
                    direccion de memoria 64($fp), que seria la
                    variable * amountSavedInOBuffer.
697         lw              v0,0(v0)
                        # Cargo la direccion de memoria del
                    contenido en v0.
698         sw              v0,32($fp)
                        # Guardo en la direccion de memoria 32(
                    $fp) la direccion de memoria de la variable
699                                                                 #
                                                                        amountSavedInOBu
                                                                        (
                                                                        int
                                                                        bytesToWrite
                                                                        =
                                                                        (*
                                                                        amountSavedInOBu
                                                                        )
                                                                        ;)
                                                                        .
700 $WhileWriteOBufferInOFile:
701         lw              v0,24($fp)
                        # Cargo en v0 el contenido de la
                    direccion de memoria 24($fp), que seria la
                    variable completeDelivery.
702         beq             v0,FALSE,
                    $GoInWhileWriteOBufferInOFile  # Si
```

```
            completeDelivery es FALSE (todavia no se
            guardaron todos los datos cargados en el buffer
            en el archivo)
703                                                          #
                                                                 entro
                                                                 al
                                                                 while
                                                                 para
                                                                 continuar
                                                                 la
                                                                 bajada
                                                                 de
                                                                 los
                                                                 datos
                                                                 al
                                                                 buffer
                                                                 .
704         b               $WriteOBufferInOFileReturnOkey
                         # Salto incondicional para retornar OKEY
                 como resultado del proceso de escritura en el
            archivo de salida.
705 $GoInWhileWriteOBufferInOFile:
706         # obuffer + bytesWriteAcum
707         lw              v1,obuffer
                        # Cargo en v1 obuffer (variable global).
708         lw              v0,28($fp)
                        # Cargo en v0 el contenido de la
            direccion de memoria 28($fp), que es la variable
            bytesWriteAcum.
709         addu            v0,v1,v0
                        # Sumo la direccion de obuffer con el
            contenido de bytesWriteAcum, y lo guardo en v0.
710
711         lw              a0,oFileDescriptor
                        # Cargo en a0 la variable oFileDescriptor
            .
712
713         move            a1,v0
                        # Muevo el contenido de v0 (corrimiento
            de direccion de memoria sobre obuffer) en a1.
714
```

```
715         lw              a2,32($fp)
                    # Cargo en a2 el contenido de la
                direccion de memoria 32($fp), que seria la
                variable bytesToWrite.
716
717         li              v0, SYS_write
718         syscall                         # Seria write: int
                bytesWrite = write(oFileDescriptor, obuffer +
                bytesWriteAcum, bytesToWrite);
719
720         # Chequeo errores. v0 contiene el numero de
                caracteres escrito (es negativo si hubo error).
721         sw              v0,36($fp)
                    # Guardo en la direccion de memoria 36(
                $fp) la cantidad de bytes escritos efectivamente
                en el archivo de salida, que esta en v0.
722         lw              v0,36($fp)
                    # Cargo en v0 la cantidad de bytes
                escritos (bytesWrite).
723         bgez            v0,$ContinueWriteOBufferInOFile
                    # Si la cantidad de bytes escritos (
                bytesWrite) es mas grande que 0, salto a
                continuar escribiendo
724                                                                 #
                                                                        en
                                                                        el
                                                                        archivo
                                                                        si
                                                                        es
                                                                        necesario
                                                                        (
                                                                        ContinueWriteOBu
                                                                        )
                                                                        .
725
726         # Hubo un error (la cantidad de caracteres escritos
                es menor a 0, valor negativo).
727         li              v0,ERROR_WRITE
                    # Cargo en v0 el resultado de la funcion,
                 que seria un codigo de error (ERROR_WRITE).
728         sw              v0,40($fp)
                    # Guardo en la direccion de memoria 40(
                $fp) el resultado de la funcion que estaba en v0
                (ERROR_WRITE).
729         b               $WriteOBufferInOFileReturn
                    # Salto incondicional al final de la
```

```
            funcion, al return.
$ContinueWriteOBufferInOFile:
        # bytesWriteAcum += bytesWrite;
        lw              v1,28($fp)
                        # Cargo en v1 el contenido de la
            direccion de memoria 28($fp), que seria la
            variable bytesWriteAcum.
        lw              v0,36($fp)
                        # Cargo en v0 el contenido de la
            direccion de memoria 36($fp), que seria la
            variable bytesWrite.
        addu            v0,v1,v0
                        # Sumo el contenido de v1 (bytesWriteAcum
            ) y el contenido de v0 (bytesWrite), y guardo el
            resultado en v0.
        sw              v0,28($fp)
                        # Guardo en la direccion de memoria 28(
            $fp) el contenido de v0, que seria el resultado
            de la suma (bytesWriteAcum += bytesWrite;).

        # bytesToWrite = (*amountSavedInOBuffer) -
            bytesWriteAcum;
        lw              v0,64($fp)
                        # Cargo en v0 el contenido de la
            direccion de memoria 64($fp), que seria la
            variable *amountSavedInOBuffer (una direccion de
            memoria).
        lw              v1,0(v0)
                        # Cargo lo contenido en la direccion de
            memoria guardada en v0 en v1 (
            ammountSavedInOBuffer es un puntero).
        lw              v0,28($fp)
                        # Cargo en v0 el contenido de la
            direccion de memoria 28($fp), que seria la
            variable bytesWriteAcum.
        subu            v0,v1,v0
                        # Resto el contenido de v1 (*
            amountSavedInOBuffer) con el contenido de v0 (
            bytesWriteAcum), y guardo el resultado en v0.
        sw              v0,32($fp)
                        # Guardo el resultado de la resta en la
            direccion de memoria 32($fp), que seria la
            variable bytesToWrite.

        lw              v0,32($fp)
                        # Cargo en v0 el contenido de la
            direccion de memoria 32($fp), que seria la
            variable bytesToWrite.
        bgtz            v0,$WhileWriteOBufferInOFile
                        # Si bytesToWrite es mayor a cero, salto
            a WhileWriteOBufferInOFile (vuelvo a entrar al
            loop while).
        # (bytesToWrite <= 0) ? then:
```

```
747         li              v0,TRUE
                        # Cargo en v0 el literal TRUE (que es 1).
748         sw              v0,24($fp)
                        # Guardo en la direccion de memoria 24(
                $fp) el contenido de v0. O sea, completeDelivery
                = TRUE;
749         b               $WhileWriteOBufferInOFile
                        # Salto incondicional al inicio del loop.
750 $WriteOBufferInOFileReturnOkey:
751         sw              zero,40($fp)
                        # Guardo en la direccion de memoria 40(
                $fp) el resultado OKEY (resultado de la funcion
                writeOBufferInOFile).
752 $WriteOBufferInOFileReturn:
753         lw              v0,40($fp)
                        # Cargo el resultado de la funcion
                writeOBufferInOFile, que estaba en la direccion
                de memoria 40($fp), en el registro v0.
754         move            sp,$fp
755         lw              ra,56(sp)
756         lw              $fp,52(sp)
757         addu            sp,sp,64
758         j               ra
                        # Jump and return
759         .end            writeOBufferInOFile
760
761
762 ##----- executePalindromeWrite -----##
763
764         .align          2
765         .globl          executePalindromeWrite
766         .ent            executePalindromeWrite
767 executePalindromeWrite:
768         .frame          $fp,80,ra
769         .set            noreorder
770         .cpload         t9
771         .set            reorder
772
773         #Stack frame creation
774         subu            sp,sp,80
775
776         .cprestore 16
777         sw              ra,72(sp)
778         sw              $fp,68(sp)
779         sw              gp,64(sp)
780         move            $fp,sp
781
782         # Parameter
783         sw              a0,80($fp)
                        # Guardo en la direccion de memoria 80(
                $fp) la variable * ibuffer (char * ibuffer).
784         sw              a1,84($fp)
                        # Guardo en la direccion de memoria 84(
                $fp) la variable * amountSavedInOBuffer (int *
```

```
            amountSavedInOBuffer).
785
786         sw              zero,24($fp)
                    # Guardo en la direccion de memoria 24(
            $fp) el valor FALSE (que seria 0), representa la
            variable findEnd (int findEnd = FALSE).
787         sw              zero,28($fp)
                    # Guardo en la direccion de memoria 28(
            $fp) el valor FALSE (que seria 0), representa la
            variable loadIBuffer (int loadIBuffer = FALSE).
788         sw              zero,32($fp)
                    # Guardo en la direccion de memoria 32(
            $fp) el valor 0, representa la variable idx (int
            idx = 0).
789         sw              zero,36($fp)
                    # Guardo en la direccion de memoria 36(
            $fp) el valor OKEY (que seria 0), representa la
            variable rdo (int rdo = FALSE).
790 $WhileExecPalindromeWrite:
791         # findEnd == FALSE ?
792         lw              v0,24($fp)
                    # Cargo en v0 el contenido de la
            direccion de memoria 24($fp), que seria la
            variable findEnd.
793         bne             v0,FALSE,
            $LeaveWhileExecPalindromeWrite # If (findEnd !=
            FALSE) goto LeaveWhileExecPalindromeWrite.
794
795         # loadIBuffer == FALSE ?
796         lw              v0,28($fp)
                    # Cargo en v0 el contenido de la
            direccion de memoria 28($fp), que seria la
            variable loadIBuffer.
797         bne             v0,FALSE,
            $LeaveWhileExecPalindromeWrite # If (loadIBuffer
            != FALSE) goto LeaveWhileExecPalindromeWrite.
798
799         # Comienzo a ejecutar las intrucciones dentro del
            while
800
801         # char character = ibuffer[idx];
802         lw              v1,80($fp)
                    # Cargo en v1 lo guardado en la direccion
             de memoria 80($fp), que seria la variable
            ibuffer.
803         lw              v0,32($fp)
                    # Cargo en v0 lo guardado en la direccion
             de memoria 32($fp), que seria la variable idx.
804         addu            v0,v1,v0
                    # Me corro en la direccion de memoria: a
            la apuntada por v1 (ibuffer) me corro la cantidad
             de posiciones establecidas por idx.
805         lbu             v0,0(v0)
                    # Cargo en v0 la direccion de memoria
```

```
            guardada en v0 (calculada en el paso anterior).
806         sb              v0,40($fp)
                    # Guardo en la direccion 40($fp) lo
            guardado en v0. Representaria la variable
            character.
807
808         # character == '\0' ?
809         lb              v0,40($fp)
                    # Cargo en v0 el contenido de la
            direccion de memoria 40($fp), que seria la
            variable character.
810         bne             v0,zero,
            $VerifyCharacterToLoadInLexico  # If (character
            != '\0') goto VerifyCharacterToLoadInLexico.
811
812         # character is equal '\0'
813         li              v0,TRUE
                    # Cargo en v0 TRUE (que seria el literal
            1).
814         sw              v0,24($fp)
                    # Guardo en la direccion de memoria 24(
            $fp) el contenido de v0 (findEnd = TRUE).
815 $VerifyCharacterToLoadInLexico:
816         # findEnd != TRUE
817         lw              v1,24($fp)
                    # Cargo en v1 el contenido de la
            direccion de memoria 24($fp), que seria findEnd.
818         li              v0,TRUE
                    # Cargo en v0 el literal 1 (TRUE) para
            hacer luego una comparacion.
819         beq             v1,v0,
            $VerifyQuantityCharacterInLexico  # Si findEnd (
            v1) es igual a TRUE (v0), salto a
            VerifyQuantityCharacterInLexico.
820
821         # findEnd es igual a TRUE. Continuo validaciones
            para cargar caracter en lexico.
822         # Voy a verificar si el caracter es una keyword.
823         lb              v0,40($fp)
                    # Cargo en v0 el contenido en la
            direccion de memoria 40($fp), que seria character
            .
824         move            a0,v0
                    # Muevo el contenido de v0 a a0. Voy a
            pasar como parametro la variable character a la
            funcion isKeywords.
825         la              t9,isKeywords
                    # Cargo la direccion de memoria de
            isKeywords.
826         jal             ra,t9
                    # Ejecuto la funcion isKeywords.
827         move            v1,v0
                    # Muevo el resultado de la funcion
            isKeywords, que esta en v0, a v1.
```

```
828         li              v0,TRUE
                        # Cargo en v0 TRUE, que es el literal 1.
829         bne             v1,v0,
                $VerifyQuantityCharacterInLexico  # Si el
                resultado de la funcion isKeywords (v1) no es
                igual a TRUE (v0), salto a
                VerifyQuantityCharacterInLexico.
830
831         # El caracter es una keyword.
832         lw              v0,lexico
                        # Cargo en v0 lo guardado en lexico.
833         bne             v0,DIR_NULL,
                $VerifyMemorymyfreeInLexico   # Si el contenido
                de lexico no apunta a NULL, voy a verificar la
                memoria disponible para cargarle character (goto
                VerifyMemorymyfreeInLexico).
834
835         # Voy a asignar memoria a la variable lexico. lexico
                 es igual a NULL.
836         li              a0,LEXICO_BUFFER_SIZE
                        # Cargo en a0 la cantidad de bytes a
                asignar por buffer a lexico (es un literal).
837         la              t9,mymalloc
                        # Cargo en t9 la direccion de la funcion
                mymalloc.
838         jal             ra,t9
                        # Ejecuto la funcion mymalloc.
839         sw              v0,lexico
                        # Guardo en lexico la memoria asignada
                con mymalloc (que esta en v0).
840         li              v0,LEXICO_BUFFER_SIZE
                        # Cargo en v0 el literal
                LEXICO_BUFFER_SIZE (cantidad de bytes a asignar
                por buffer a lexico).
841         sw              v0,bytesLexico
                        # Guardo el contenido de vo (
                LEXICO_BUFFER_SIZE) en la variable global
                bytesLexico.
842         b               $LoadCharacterInLexico
                        # Salto incondicional, voy a cargar
                caracter en lexico.
843 $VerifyMemorymyfreeInLexico:
844         lw              v0,quantityCharacterInLexico
                        # Cargo en v0 el contenido de
                quantityCharacterInLexico.
845         lw              v1,bytesLexico
                        # Cargo en v1 el contenido de bytesLexico
                .
846         slt             v0,v0,v1
                        # Verifico si quantityCharacterInLexico
                es mas chico que bytesLexico. Guardo resultado en
                 v0 (TRUE o FALSE).
847         bne             v0,FALSE,$LoadCharacterInLexico
                        # Si quantityCharacterInLexico >=
```

```
            bytesLexico (el contenido de v0 es FALSE,
            resultado comparacion anterior), voy a
            LoadCharacterInLexico.
848
849         # Realloc para lexico
850         lw              v0,bytesLexico
                        # Cargo en v0 el contenido de bytesLexico
            .
851         sw              v0,44($fp)
                        # Guardo en la direccion de memoria 44(
            $fp) el contenido de v0, que seria bytesLexico.
            Representaria la variable bytesLexicoPreview.
852         lw              v0,bytesLexico
                        # Cargo en v0 el contenido de bytesLexico
            .
853         addu            v0,v0,LEXICO_BUFFER_SIZE
                        # Le sumo a bytesLexico
            LEXICO_BUFFER_SIZE y guardo resultado en v0.
854         sw              v0,bytesLexico
                        # Guardo el contenido de la suma en
            bytesLexico.
855         lw              a0,lexico
                        # Cargo en a0 lexico para enviarlo por
            parametro a myRealloc.
856         lw              a1,bytesLexico
                        # Cargo en a1 bytesLexico para enviarlo
            por parametro a myRealloc.
857         lw              a2,44($fp)
                        # Cargo en a2 lo guardado en la direccion
             de memoria 44($fp), que representaba a
            bytesLexicoPreview, para enviarlo por parametro a
             myRealloc.
858         la              t9,myRealloc
                        # Cargo en t9 la direccion de la funcion
            myRealloc.
859         jal             ra,t9
                        # Ejecuto la funcion myRealloc.
860         sw              v0,lexico
                        # Guardo en lexico la nueva direccion de
            memoria. En v0 esta el resultado de la funcion
            myRealloc.
861 $LoadCharacterInLexico:
862         lw              v0,lexico
                        # Cargo en v0 lexico.
863         bne             v0,DIR_NULL,$LoadCharacter
                        # If (lexico != NULL) goto LoadCharacter
864
865         # lexico is NULL => Mensaje de error
866         li              a0,FILE_DESCRIPTOR_STDERR
                        # Cargo en a0 FILE_DESCRIPTOR_STDERR.
867         la              a1,MENSAJE_ERROR_MEMORIA_LEXICO #
            Cargo en a1 la direccion de memoria donde se
            encuentra el mensaje a cargar.
```

```
868         li              a2,
                BYTES_MENSAJE_ERROR_MEMORIA_LEXICO   # Cargo en
                a2 la cantidad de bytes a escribir.
869         li              v0, SYS_write
870         syscall                                     # No
                 controlo error porque sale de por si de la
                funcion por error.
871
872         li              v0,ERROR_MEMORY
                        # Cargo en v0 el codigo de error.
873         sw              v0,60($fp)
                        # Guardo el codigo de error (en v0) en la
                 direccion de memoria 60($fp).
874         b               $ReturnExecutePalindromeWrite
                        # Salto incondicional al return de la
                funcion (goto ReturnExecutePalindromeWrite).
875 $LoadCharacter:
876         # lexico is not NULL
877         lw              v1,lexico
                        # Cargo en v1 lexico.
878         lw              v0,quantityCharacterInLexico
                        # Cargo en v0 quantityCharacterInLexico.
879         addu            v1,v1,v0
                        # Me desplazo en la direccion de memoria
                de lexico (v1) la cantidad especificada por
                quantityCharacterInLexico (v0) y
880                                                                 #
                                                                    guardo
                                                                    la
                                                                    nueva
                                                                    posicion
                                                                    en
                                                                    v1
                                                                    .
881         lbu             v0,40($fp)
                        # Cargo en v0 lo almacenado en la
                direccion de memoria 40($fp), que seria la
                variable character.
882         sb              v0,0(v1)
                        # Guardo en la posicion de memoria
                guardada almacenada en v1 (0 de desplazamiento),
                el contenido de v0
883                                                                 #
                                                                    (
                                                                    character
                                                                    )
```

```
.

0

sea
:

lexico
[
quantityCharacte
]

=

character
;
```

```
884         lw              v0,quantityCharacterInLexico
                        # Cargo en v0 quantityCharacterInLexico
885         addu            v0,v0,1
                        # Incremento en 1 la cantidad de
                    caracteres guardados en lexico (
                    quantityCharacterInLexico ++;). Este resultado se
                     guarda en v0.
886         sw              v0,quantityCharacterInLexico
                        # Guardo el resultado del incremento (v0)
                     en quantityCharacterInLexico.
887         b               $VerifyLoadIBuffer
                        # Salto incondicional a VerifyLoadIBuffer
                    .
888 $VerifyQuantityCharacterInLexico:
889         # quantityCharacterInLexico > 0 ?
890         lw              v0,quantityCharacterInLexico
                        # Cargo en v0 quantityCharacterInLexico.
891         blez            v0,$VerifyLoadIBuffer
                        # Si quantityCharacterInLexico (v0) es
                    menor que 0, salto a VerifyLoadIBuffer.
892
893         # quantityCharacterInLexico > 0 => Verifico si
                    lexico es palindromo.
894         lw              a0,lexico
                        # Cargo en a0 lexico (parametro para la
                    funcion verifyPalindromic).
895         lw              a1,quantityCharacterInLexico
                        # Cargo en a1 quantityCharacterInLexico (
                    parametro para la funcion verifyPalindromic).
896         la              t9,verifyPalindromic
                        # Cargo en t9 la direccion de la
                    verifyPalindromic.
897         jal             ra,t9
                        # Ejecuto la funcion verifyPalindromic.
898         sw              v0,44($fp)
                        # Guardo en la direccion de memoria 44(
                    $fp) el resultado de la funcion verifyPalindromic
```

```
            , que esta
899                                                                 #
                                                                        almacenado
                                                                        en
                                                                        v0
                                                                        y
                                                                        que
                                                                        representaria
                                                                        a
                                                                        la
                                                                        variable
                                                                        itsPalindromic
                                                                        .
900         lw              v1,44($fp)
                        # Cargo en v1 lo que se encuentra en la
                    direccion de memoria 44($fp) que seria el
                    resultado de la funcion verifyPalindromic.
901         li              v0,TRUE
                        # Cargo en v0 TRUE (es el literal 1) para
                     hacer luego la comparacion.
902         bne             v1,v0,$myfreeLexico
                        # If (itsPalindromic != TRUE) goto
                    myfreeLexico.
903
904         # itsPalindromic is TRUE
905
906         # int amountToSaved = (*amountSavedInOBuffer) +
                    quantityCharacterInLexico;
907         lw              v0,84($fp)
                        # Cargo en v0 lo almacenado en la
                    direccion de memoria 84($fp), que seria la
                    variable *amountSavedInOBuffer.
908         lw              v1,0(v0)
                        # Cargo lo almacenado en la direccion de
                    memoria apuntada por *amountSavedInOBuffer en v1.
909         lw              v0,quantityCharacterInLexico
                        # Cargo en v0 quantityCharacterInLexico.
910         addu            v0,v1,v0
                        # Hago (*amountSavedInOBuffer) +
                    quantityCharacterInLexico, y guardo el resultado
                    en v0.
911         sw              v0,48($fp)
                        # Guardo el resultado de la suma (
```

```
            almacenado el v0) en la direccion de memoria 48(
            $fp), que representaria a la variable
            amountToSaved.
912
913         # (*amountSavedInOBuffer) > 0 ?
914         lw              v0,84($fp)
                        # Cargo en v0 lo almacenado en la
            direccion de memoria 84($fp), que seria la
            variable *amountSavedInOBuffer.
915         lw              v0,0(v0)
                        # Cargo lo almacenado en la direccion de
            memoria apuntada por *amountSavedInOBuffer en v0.
916         bgtz            v0,$IncrementAmountToSaved
                        # Si el contenido de lo almacenado en la
            direccion apuntada por amountSavedInOBuffer (que
            esta en v0)
917                                                             #
                                                                es
                                                                mas
                                                                grande
                                                                que
                                                                0,
                                                                salto
                                                                a
                                                                IncrementAmountT
                                                                .
918
919         # (*amountSavedInOBuffer) <= 0
920         # savedInOFile == TRUE ?
921         lw              v1,savedInOFile
                        # Cargo en v1 savedInOFile.
922         li              v0,TRUE
                        # Cargo en v0 TRUE (literal igual a 1).
923         beq             v1,v0,$IncrementAmountToSaved
                        # If (savedInOFile == TRUE) goto
            IncrementAmountToSaved.
924         b
            $ContinueVerificationAboutAmountToSaved # Salto
            incondicional a
            ContinueVerificationAboutAmountToSaved.
925 $IncrementAmountToSaved:
926         # amountToSaved ++;  Es para el separador entre
            lexicos
927         lw              v0,48($fp)
                        # Cargo en v0 lo guardado en la direccion
```

```
                de memoria 48($fp), que representaria a la
            variable amountToSaved.
928         addu            v0,v0,1
                        # Incremento en uno a amountToSaved.
929         sw              v0,48($fp)
                        # Guardo el nuevo valor de amountToSaved
            (almacenado en v0) en la direccion de memoria 48(
            $fp).
930 $ContinueVerificationAboutAmountToSaved:
931         # amountToSaved > osize ?
932         lw              v0,48($fp)
                        # Cargo en v0 lo guardado en la direccion
                 de memoria 48($fp), que representaria a la
            variable amountToSaved.
933         lw              v1,osize
                        # Cargo en v1 osize.
934         sltu            v0,v1,v0
                        # Si v1 (osize) es mas chico que v0 (
            amountToSaved), guardo TRUE en v0, sino guardo
            FALSE.
935         beq             v0,FALSE,$LoadLexicoInOBuffer
                        # Si el resultado de la comparacion es
            FALSE (amountToSaved <= osize), salto a
            LoadLexicoInOBuffer.
936
937         # amountToSaved > osize
938         # Tomo la decision de pedir mas memoria para bajar
            el lexico completo
939         # y luego rearmo el buffer de salida y reinicio la
            cantidad guardada en 0.
940
941         # obuffer = myRealloc(obuffer, amountToSaved*sizeof(
            char), (*amountSavedInOBuffer));
942         lw              v0,84($fp)
                        # Cargo en v0 lo guardado en la direccion
                 de memoria 84($fp), que representaria a la
            variable *amountSavedInOBuffer.
943         lw              a0,obuffer
                        # Cargo en a0 obuffer (parametro para la
            funcion myRealloc).
944         lw              a1,48($fp)
                        # Cargo en a1 lo guardado en la direccion
                 de memoria 48($fp), que representaria a la
            variable *amountToSaved. Parametro
945                                                             #
                                                                para
                                                                la
                                                                funcion
                                                                myRealloc
                                                                .
```

```
946         lw              a2,0(v0)
                    # Cargo en a2 lo almacenado en la
                direccion de memoria guardada en v0 (parametro
                para la funcion myRealloc).
947         la              t9,myRealloc
                    # Cargo en t9 la direccion de la
                myRealloc.
948         jal             ra,t9
                    # Ejecuto myRealloc con los parametros:
                myRealloc(obuffer, amountToSaved*sizeof(char), (*
                amountSavedInOBuffer));
949         sw              v0,obuffer
                    # Asigno a obuffer el resultado de
                myRealloc almacenado en v0.
950
951         # (*amountSavedInOBuffer) > 0 ?
952         lw              v0,84($fp)
                    # Cargo en v0 lo guardado en la direccion
                 de memoria 84($fp), que representaria a la
                variable *amountSavedInOBuffer.
953         lw              v0,0(v0)
                    # Cargo en v0 lo almacenado en la
                direccion de memoria guardada en v0 (*
                amountSavedInOBuffer).
954         bgtz            v0,$LoadEnterInOBuffer
                    # Si el contenido de v0 (*
                amountSavedInOBuffer) es mayor a 0, salta
                LoadEnterInOBuffer.
955
956         # (*amountSavedInOBuffer) <= 0 => continuo
                verificando si debo de guardar un enter ('\n').
957         # savedInOFile == TRUE ?
958         lw              v1,savedInOFile
                    # Cargo en v1 savedInOFile.
959         li              v0,TRUE
                    # Cargo en v0 TRUE (literal igual a 1)
                para luego hacer comparacion.
960         beq             v1,v0,$LoadEnterInOBuffer
                    # If (savedInOFile == TRUE), goto
                LoadEnterInOBuffer.
961         b               $LoadLexicoInOBufferToWriteFile
                    # Salto incondicional a
                LoadLexicoInOBufferToWriteFile.
962 $LoadEnterInOBuffer:
963         # (*amountSavedInOBuffer) > 0 || savedInOFile ==
                TRUE
964
965         # obuffer[*amountSavedInOBuffer] = '\n';
966         lw              v0,84($fp)
                    # Cargo en v0 lo guardado en la direccion
                 de memoria 84($fp), que representaria a la
                variable *amountSavedInOBuffer.
```

```
967         lw              v1,obuffer
                        # Cargo en v1 obuffer.
968         lw              v0,0(v0)
                        # Cargo en v0 lo almacenado en la
                    direccion de memoria guardada en v0 (*
                    amountSavedInOBuffer).
969         addu            v1,v1,v0
                        # Me muevo en la memoria: obuffer + (*
                    amountSavedInOBuffer). La nueva direccion la
                    guardo en v1.
970         li              v0,LINE_BREAK
                        # Cargo en v0 el salto de linea (literal
                    10).
971         sb              v0,0(v1)
                        # Cargo el salto de linea en la direccion
                     apuntada por v1: obuffer[*amountSavedInOBuffer]
                    = '\n';
972
973         # *amountSavedInOBuffer = (*amountSavedInOBuffer) +
                    1;
974         lw              v1,84($fp)
                        # Cargo en v1 lo guardado en la direccion
                     de memoria 84($fp), que representaria a la
                    variable *amountSavedInOBuffer.
975         lw              v0,84($fp)
                        # Cargo en v0 lo guardado en la direccion
                     de memoria 84($fp), que representaria a la
                    variable *amountSavedInOBuffer.
976         lw              v0,0(v0)
                        # Cargo en v0 lo almacenado en la
                    direccion de memoria guardada en v0 (*
                    amountSavedInOBuffer).
977         addu            v0,v0,1
                        # Incremento en 1 el contenido apuntado
                    por amountSavedInOBuffer. Guardo resultado en v0.
978         sw              v0,0(v1)
                        # Guardo el incremento en la direccion
                    apuntada por amountSavedInOBuffer.
979 $LoadLexicoInOBufferToWriteFile:
980         sw              zero,52($fp)
                        # Creo un indice (i) en la direccion de
                    memoria 52($fp), inicializado en 0.
981 $ForLexicoInOBuffer:
982         lw              v0,52($fp)
                        # Cargo el indice i en v0.
983         lw              v1,quantityCharacterInLexico
                        # Cargo en v1 quantityCharacterInLexico.
984         slt             v0,v0,v1
                        # Guardo TRUE en v0 si (i <
                    quantityCharacterInLexico), sino guardo FALSE.
985         bne             v0,FALSE,$InForLexicoInOBuffer
                        # Si el resultado de la comparacion no es
                     FALSE, o sea, (i < quantityCharacterInLexico),
                    entro al for (goto InForLexicoInOBuffer).
```

```
986         b               $WriteLexicoInOFile
                        # Salto incondicional a
                WriteLexicoInOFile.
987 $InForLexicoInOBuffer:
988         # obuffer[*amountSavedInOBuffer] = lexico[i];
989         lw              v0,84($fp)
                                    # Cargo en v0 lo guardado
                 en la direccion de memoria 84($fp), que
                representaria a la variable *amountSavedInOBuffer
                .
990         lw              v1,obuffer
                                    # Cargo en v1 obuffer.
991         lw              v0,0(v0)
                                    # Cargo en v0 lo
                almacenado en la direccion de memoria guardada en
                 v0 (*amountSavedInOBuffer).
992         addu            a0,v1,v0
                                    # Guardo en a0 la nueva
                direccion de memoria sobre obuffer: obuffer + *
                amountSavedInOBuffer = obuffer[*
                amountSavedInOBuffer]
993         lw                      v1,lexico
                                            # Cargo en v1
                lexico.
994         lw                      v0,52($fp)
                                            # Cargo en v0 el
                indice i guardado en la direccion de memoria 52(
                $fp).
995         addu            v0,v1,v0
                                    # Guardo en v0 la nueva
                direccion de memoria sobre lexico: lexico + i =
                lexico[i]
996         lbu                     v0,0(v0)
                                            # Cargo en v0 lo
                guardado en la direccion de memoria almacenada en
                 v0 (es sobre lexico).
997         sb                      v0,0(a0)
                                            # Guardo en la
                direccion de memoria almacenada en a0 (es sobre
                obuffer) lo almacenado en v0. O sea: obuffer[*
                amountSavedInOBuffer] = lexico[i];
998
999         # *amountSavedInOBuffer = (*amountSavedInOBuffer) +
                1;
1000        lw                      v1,84($fp)
                                            # Cargo en v1 lo
                guardado en la direccion de memoria 84($fp), que
                representaria a la variable *amountSavedInOBuffer
                .
1001        lw                      v0,84($fp)
                                            # Cargo en v0 lo
                guardado en la direccion de memoria 84($fp), que
                representaria a la variable *amountSavedInOBuffer
                .
```

```
1002        lw                      v0,0(v0)
                                            # Cargo en v0 lo
            almacenado en la direccion de memoria guardada en
             v0 (*amountSavedInOBuffer).
1003        addu            v0,v0,1
                                # Incremento en 1 el contenido
            apuntado por amountSavedInOBuffer.
1004        sw                      v0,0(v1)
                                            # Guardo el
            incremento.
1005
1006        # ++ i
1007        lw                      v0,52($fp)
                                            # Cargo en v0 el
            indice i guardado en la direccion de memoria 52(
            $fp).
1008        addu            v0,v0,1
                                # Incremento en 1 el indice i.
1009        sw                      v0,52($fp)
                                            # Guardo el
            incremento.
1010        b                       $ForLexicoInOBuffer
                                        # Salto incondicional.
            Vuelvo al comienzo del loop for.
1011    $WriteLexicoInOFile:
1012        # int rdoWrite = writeOBufferInOFile(
            amountSavedInOBuffer);
1013        lw                      a0,84($fp)
                                            # Cargo en a0 lo
            guardado en la direccion de memoria 84($fp), que
            representaria a la variable *amountSavedInOBuffer
            . Parametro de la funcion writeOBufferInOFile.
1014        la                      t9,writeOBufferInOFile
                                # Cargo en t9 la direccion de la
            writeOBufferInOFile.
1015        jal                     ra,t9
                                            # Ejecuto la
            funcion writeOBufferInOFile.
1016        sw                      v0,56($fp)
                                            # Guardo en la
            direccion de memoria 56($fp) el resultado de
            ejecutar la funcion writeOBufferInOFile
            almacenado en v0.
1017
1018        # rdoWrite != OKEY ?
1019        lw                      v0,56($fp)
                                            # Cargo en v0 el
            resultado de la ejecucion de la funcion
            writeOBufferInOFile, que seria la variable
            rdoWrite.
1020        beq                     v0,OKEY,$WriteInNewOBuffer
                              # If (rdoWrite == OKEY) goto
            WriteInNewOBuffer.
```

```
1021        lw                      v0,56($fp)
                                            # Cargo en v0 el
            resultado de la ejecucion de la funcion
            writeOBufferInOFile, que seria la variable
            rdoWrite.
1022        sw                      v0,60($fp)
                                            # Guardo el
            codigo de error (en v0) en la direccion de
            memoria 60($fp).
1023        b
                $ReturnExecutePalindromeWrite   # Salto
            incondicional al return de la funcion (goto
            ReturnExecutePalindromeWrite).
1024    $WriteInNewOBuffer:
1025        # *amountSavedInOBuffer = 0;
1026        lw                      v0,84($fp)
                                            # Cargo en v0 lo
            guardado en la direccion de memoria 84($fp), que
            representaria a la variable *amountSavedInOBuffer
            .
1027        sw                      zero,0(v0)
                                        # Guardo 0 en la
            direccion apuntada por amountSavedInOBuffer.
1028
1029        # savedInOFile = TRUE;
1030        li                      v0,TRUE
1031        sw                      v0,savedInOFile
                                # Guardo en savedInOFile el
            contenido de v0 (TRUE).
1032
1033        # obuffer != NULL ?
1034        lw                      v0,obuffer
                                            # Cargo en v0
            obuffer.
1035        beq                     v0,DIR_NULL,
            $mymallocNewOBuffer # If (obuffer == NULL) goto
            mymallocNewOBuffer.
1036
1037        # obuffer != NULL => myfree(obuffer) and obuffer =
            NULL.
1038        lw                      a0,obuffer
                                            # Cargo en a0
            obuffer.
1039        la                      t9,myfree
                                            # Cargo en t9 la
            direccion de memoria de myfree.
1040        jal                     ra,t9
                                            # Ejecuto myfree
            sobre obuffer.
1041        sw                      zero,obuffer
                                # Asigno NULL a obuffer.
1042    $mymallocNewOBuffer:
1043        # obuffer = (char *) mymalloc(osize*sizeof(char));
```

```
1044        lw                      a0,osize
                                    # Cargo en a0
            osize.
1045        la                      t9,mymalloc
                                    # Cargo en t9 la
            direccion de memoria de mymalloc.
1046        jal                     ra,t9
                                    # Ejecuto
            mymalloc.
1047        sw                      v0,obuffer
                                    # Asigno la nueva
             direccion de memoria, que se encuentra
            almacenada en v0, a obuffer.
1048
1049        # obuffer == NULL ?
1050        lw                      v0,obuffer
                                    # Cargo en v0
            obuffer.
1051        bne                     v0,DIR_NULL,
            $InitializeNewOBuffer # If (obuffer != NULL) goto
             InitializeNewOBuffer.
1052
1053        # obuffer is NULL => Mensaje de error
1054        li                      a0,FILE_DESCRIPTOR_STDERR
                            # Cargo en a0
            FILE_DESCRIPTOR_STDERR.
1055        la                      a1,
            MENSAJE_ERROR_MEMORIA_OBUFFER # Cargo en a1 la
            direccion de memoria donde se encuentra el
            mensaje a cargar.
1056        li                      a2,
            BYTES_MENSAJE_ERROR_MEMORIA_OBUFFER  # Cargo en
            a2 la cantidad de bytes a escribir.
1057        li                      v0, SYS_write
1058        syscall                                         # No
             controlo error porque sale de por si de la
            funcion por error.
1059
1060        li                      v0,ERROR_MEMORY
1061        sw                      v0,60($fp)
                                    # Guardo el
            codigo de error (en v0) en la direccion de
            memoria 60($fp).
1062        b
            $ReturnExecutePalindromeWrite   # Salto
            incondicional al return de la funcion (goto
            ReturnExecutePalindromeWrite).
1063    $InitializeNewOBuffer:
1064        lw                      a0,osize
                                    # Cargo en a0
            osize. Parametro de la funcion initializeBuffer.
1065        lw                      a1,obuffer
                                    # Cargo en a1
            obuffer. Parametro de la funcion initializeBuffer
```

```
            .
1066        la                      t9,initializeBuffer
                                # Cargo en t9 la direccion de
            memoria de la funcion initializeBuffer.
1067        jal                     ra,t9
                                                # Ejecuto la
            funcion initializeBuffer.
1068
1069        b                       $myfreeLexico
                                                # Salto
            incondicional a myfreeLexico.
1070    $LoadLexicoInOBuffer:
1071        # (*amountSavedInOBuffer) > 0 ?
1072        lw                      v0,84($fp)
                                                # Cargo en v0 lo
            guardado en la direccion de memoria 84($fp), que
            representaria a la variable *amountSavedInOBuffer
            .
1073        lw                      v0,0(v0)
                                                # Cargo en v0 lo
            almacenado en la direccion de memoria guardada en
             v0 (*amountSavedInOBuffer).
1074        bgtz            v0,$LoadLineBeakInOBuffer
                        # Si (*amountSavedInOBuffer > 0), voy a
            guardar el salto de linea para separar lexicos
            que son palindromos (LoadLineBeakInOBuffer).
1075
1076        # (*amountSavedInOBuffer) <= 0
1077        # savedInOFile == TRUE ?
1078        lw                      v1,savedInOFile
                                # Cargo en v1 savedInOFile.
1079        li                      v0,TRUE
1080        beq                     v1,v0,$LoadLineBeakInOBuffer
                    # Si (savedInOFile == TRUE), voy a guardar el
             salto de linea para separar lexicos que son
            palindromos (LoadLineBeakInOBuffer).
1081
1082        b
            $LoadLexicoInOBufferNotWriteFile # Salto
            incondicional a LoadLexicoInOBufferNotWriteFile.
1083    $LoadLineBeakInOBuffer:
1084        # obuffer[*amountSavedInOBuffer] = '\n';
1085        lw                      v0,84($fp)
                                                # Cargo en v0 lo
            guardado en la direccion de memoria 84($fp), que
            representaria a la variable *amountSavedInOBuffer
            .
1086        lw                      v1,obuffer
                                                # Cargo en v1
            obuffer.
1087        lw                      v0,0(v0)
                                                # Cargo en v0 lo
            almacenado en la direccion de memoria guardada en
             v0 (*amountSavedInOBuffer).
```

```
1088        addu            v1,v1,v0
                                # Guardo en v1 la nueva
            posicion dentro de obuffer: obuffer[*
            amountSavedInOBuffer] = obuffer + *
            amountSavedInOBuffer
1089        li                      v0,LINE_BREAK
                                # Cargo en v0 el literal
            LINE_BREAK (es 10 que representa a '\n').
1090        sb                      v0,0(v1)
                                # Guardo en
            obuffer, en la posicion indicada en v1,
            LINE_BREAK. O sea: obuffer[*amountSavedInOBuffer]
             = '\n';
1091
1092        # *amountSavedInOBuffer = (*amountSavedInOBuffer) +
            1;
1093        lw                      v1,84($fp)
                                # Cargo en v1 lo
            guardado en la direccion de memoria 84($fp), que
            representaria a la variable *amountSavedInOBuffer
            .
1094        lw                      v0,84($fp)
                                # Cargo en v0 lo
            guardado en la direccion de memoria 84($fp), que
            representaria a la variable *amountSavedInOBuffer
            .
1095        lw                      v0,0(v0)
                                # Cargo en v0 lo
            almacenado en la direccion de memoria guardada en
             v0 (*amountSavedInOBuffer).
1096        addu            v0,v0,1
                            # Incremento en 1
1097        sw                      v0,0(v1)
                                # Guardo nuevo
            valor de *amountSavedInOBuffer.
1098    $LoadLexicoInOBufferNotWriteFile:
1099        sw                      zero,56($fp)
                                # Guardo 0 en la
            direccion de memoria 56($fp). Inicializo indice
            en 0 para un nuevo loop.
1100    $ForLexicoInOBufferNotWriteFile:
1101        lw                      v0,56($fp)
                                # Cargo en v0 el
            indice (i) en 0.
1102        lw                      v1,quantityCharacterInLexico
                # Cargo en v1 quantityCharacterInLexico.
1103        slt                     v0,v0,v1
                                # Guado en v0
            TRUE si el indice i es menor que
            quantityCharacterInLexico. Caso contrario guardo
            FALSE
1104        bne                     v0,FALSE,
            $GoInForLexicoInOBufferNotWriteFile # Si (i <
            quantityCharacterInLexico) goto
```

```
            GoInForLexicoInOBufferNotWriteFile.
1105        b                       $myfreeLexico
1106    $GoInForLexicoInOBufferNotWriteFile:
1107        # obuffer[*amountSavedInOBuffer] = lexico[i];
1108        lw                      v0,84($fp)
                                            # Cargo en v0 lo
            guardado en la direccion de memoria 84($fp), que
            representaria a la variable *amountSavedInOBuffer
            .
1109        lw                      v1,obuffer
                                            # Cargo en v1
            obuffer.
1110        lw                      v0,0(v0)
                                            # Cargo en v0 lo
            almacenado en la direccion de memoria guardada en
             v0 (*amountSavedInOBuffer).
1111        addu            a0,v1,v0
                                    # Guardo en v1 la nueva
            posicion dentro de obuffer: obuffer[*
            amountSavedInOBuffer] = obuffer + *
            amountSavedInOBuffer
1112        lw                      v1,lexico
                                            # Cargo en v1
            lexico.
1113        lw                      v0,56($fp)
                                            # Cargo en v0 el
            indice i.
1114        addu            v0,v1,v0
                                    # Guardo en v0 la nueva
            posicion dentro de lexico: lexico[i] = lexico + i
1115        lbu                     v0,0(v0)
                                            # Cargo en v0 el
            contenido de la posicion dentro de lexico.
1116        sb                      v0,0(a0)
                                            # Guardo en la
            direccion apuntada por a0 (posicion dentro de
            obuffer0 el contenido almacenado en v0: obuffer[*
            amountSavedInOBuffer] = lexico[i];
1117
1118        # *amountSavedInOBuffer = (*amountSavedInOBuffer) +
            1;
1119        lw                      v1,84($fp)
1120        lw                      v0,84($fp)
1121        lw                      v0,0(v0)
1122        addu            v0,v0,1
                               # Incremento en 1 a *
            amountSavedInOBuffer.
1123        sw                      v0,0(v1)
                                            # Guardo el nuevo
             valor de *amountSavedInOBuffer.
1124
1125        # ++i
1126        lw                      v0,56($fp)
                                            # Cargo en v0 el
```

```
            indice i.
1127        addu            v0,v0,1
                                # Incremento el indice.
1128        sw                      v0,56($fp)
                                            # Guardo nuevo
            valor del indice.
1129        b
            $ForLexicoInOBufferNotWriteFile # Salto al
            principio del for para intentar entrar nuevamente
             al loop.
1130    $myfreeLexico:
1131        lw                      a0,lexico
                                            # Cargo en a0
            lexico.
1132        la                      t9,myfree
                                            # Cargo en t9 la
            direccion de la funcion myfree.
1133        jal                     ra,t9
                                            # Ejecuto la
            funcion myfree
1134        sw                      zero,lexico
                                        # Asigno NULL a lexico.
1135        sw                      zero,
            quantityCharacterInLexico # Dejo
            quantityCharacterInLexico en 0.
1136    $VerifyLoadIBuffer:
1137        # (idx + 1) == isize ?
1138        lw                      v0,32($fp)
                                            # Cargo en v0 lo
            guardado en la direccion 32($fp), que seria la
            variable idx.
1139        addu            v1,v0,1
                                # Incremento en 1 a idx y lo
            guardo en v1.
1140        lw                      v0,isize
                                            # Cargo en v0
            isize para luego hacer comparacion.
1141        bne                     v1,v0,$IncrementIdx
                                # If ((idx+1) != isize) goto
            IncrementIdx
1142
1143        # ((idx + 1) == isize) is TRUE
1144        li                      v0,TRUE
1145        sw                      v0,28($fp)
                                            # Guardo en la
            direccion 28($fp), que estaba la variable
            loadIBuffer, TRUE.
1146        li                      v0,LOAD_I_BUFFER
1147        sw                      v0,36($fp)
                                            # Guardo en la
            direccion 36($fp), que estaba la variable rdo -
            resultado de la operacion-, LOAD_I_BUFFER.
1148
```

```
1149        b                       $WhileExecPalindromeWrite
                                # Salto incondicional al comienzo
            del while para verificar entrada al mismo.
1150    $IncrementIdx:
1151        # idx ++
1152        lw                      v0,32($fp)
                                            # Cargo en v0 idx
            , guardado en la direccion 32($fp).
1153        addu            v0,v0,1
                                # Incremento en 1 a idx.
1154        sw                      v0,32($fp)
                                            # Guardo el nuevo
             valor de idx.
1155
1156        b                       $WhileExecPalindromeWrite
                                # Salto incondicional al comienzo
            del while para verificar entrada al mismo.
1157    $LeaveWhileExecPalindromeWrite:
1158        lw                      v0,36($fp)
                                            # Cargo en v0 el
            resultado del while: variable rdo guardada en la
            direccion 36($fp).
1159        sw                      v0,60($fp)
                                            # Guardo en 60(
            $fp) el resultado de la funcion.
1160    $ReturnExecutePalindromeWrite:
1161        lw                      v0,60($fp)
1162        move            sp,$fp
1163        lw                      ra,72(sp)
1164        lw                      $fp,68(sp)
1165        addu            sp,sp,80
1166        j                       ra
                                                    # Jump
            and return
1167        .end            executePalindromeWrite
1168
1169
1170    ##----- palindrome -----##
1171
1172        .align          2
1173        .globl          palindrome
1174        .ent            palindrome
1175    palindrome:
1176        .frame          $fp,80,ra
1177        .set            noreorder
1178        .cpload         t9
1179        .set            reorder
1180
1181        #Stack frame creation
1182        subu            sp,sp,80
1183
1184        .cprestore 16
1185        sw                      ra,72(sp)
1186        sw                      $fp,68(sp)
```

```
1187        sw                      gp,64(sp)
1188        move            $fp,sp
1189
1190        # Parameters
1191        sw                      a0,80($fp)
                                            # Guardo en la
                direccion de memoria 80($fp) la variable ifd (int
                 ifd).
1192        sw                      a1,84($fp)
                                            # Guardo en la
                direccion de memoria 84($fp) la variable ibytes (
                size_t ibytes).
1193        sw                      a2,88($fp)
                                            # Guardo en la
                direccion de memoria 88($fp) la variable ofd (int
                 ofd).
1194        sw                      a3,92($fp)
                                            # Guardo en la
                direccion de memoria 92($fp) la variable obytes (
                size_t obytes).
1195
1196        # isize = ibytes;
1197        lw                      v0,84($fp)
                                            # Cargo en v0
                ibytes, guardado en 84($fp).
1198        sw                      v0,isize
                                            # Guardo en isize
                 ibytes.
1199
1200        # osize = obytes;
1201        lw                      v0,92($fp)
                                            # Cargo en v0
                obytes, guardado en 92($fp).
1202        sw                      v0,osize
                                            # Guardo en osize
                 obytes.
1203
1204        # oFileDescriptor = ofd;
1205        lw                      v0,88($fp)
                                            # Cargo en v0 ofd
                , guardado en 88($fp).
1206        sw                      v0,oFileDescriptor
                                               # Guardo en
                oFileDescriptor ofd.
1207
1208        # char * ibuffer = (char *) mymalloc(ibytes*sizeof(
                char));
1209        lw                      a0,84($fp)
                                            # Cargo en a0
                ibytes, guardado en 84($fp). Parametro de la
                funcion mymalloc.
1210        la                      t9,mymalloc
                                            # Cargo en t9 la
                direccion de la funcion mymalloc.
```

```
1211        jal                     ra,t9
                                            # Ejecuto la
            funcion mymalloc.
1212        sw                      v0,24($fp)
                                            # En v0 esta el
            resultado de mymalloc. Guardo esto en la
            direccion 24($fp) que representaria la variable *
             ibuffer.
1213
1214        # ibuffer == NULL ?
1215        lw                      v0,24($fp)
                                            # Cargo en v0 *
            ibuffer, guardado en la direccion 24($fp).
1216        bne                     v0,DIR_NULL,$OBuffermymalloc
                        # If (ibuffer != NULL) goto
            OBuffermymalloc
1217
1218        # ibuffer is NULL => Mensaje de error
1219        li                      a0,FILE_DESCRIPTOR_STDERR
                            # Cargo en a0
            FILE_DESCRIPTOR_STDERR.
1220        la                      a1,
            MENSAJE_ERROR_MEMORIA_IBUFFER # Cargo en a1 la
            direccion de memoria donde se encuentra el
            mensaje a cargar.
1221        li                      a2,
            BYTES_MENSAJE_ERROR_MEMORIA_IBUFFER  # Cargo en
            a2 la cantidad de bytes a escribir.
1222        li                      v0, SYS_write
1223        syscall                                         # No
             controlo error porque sale de por si de la
            funcion por error.
1224
1225        li                      v0,ERROR_MEMORY
                           # Cargo en v0 el codigo de error.
1226        sw                      v0,60($fp)
                                            # Guardo el
            codigo de error (en v0) en la direccion de
            memoria 60($fp).
1227        b                       $ReturnPalindrome
                               # Salto incondicional al
            return de la funcion (goto ReturnPalindrome).
1228    $OBuffermymalloc:
1229        # obuffer = (char *) mymalloc(obytes*sizeof(char));
1230        lw                      a0,92($fp)
                                            # Cargo en a0
            obytes. Parametro de la funcion mymalloc.
1231        la                      t9,mymalloc
                                            # Cargo en t9 la
            direccion de la funcion mymalloc.
1232        jal                     ra,t9
                                            # Ejecuto la
            funcion mymalloc.
```

```
1233        sw                      v0,obuffer
                                            # En v0 esta el
                resultado de mymalloc. Guardo esto en obuffer.
1234
1235        # obuffer == NULL ?
1236        lw                      v0,obuffer
                                            # Cargo en v0
                obuffer.
1237        bne                     v0,DIR_NULL,
                $InitializeBuffers # If (obuffer != NULL) goto
                InitializeBuffers
1238
1239        # obuffer is NULL => Mensaje de error
1240        li                      a0,FILE_DESCRIPTOR_STDERR
                                # Cargo en a0
                FILE_DESCRIPTOR_STDERR.
1241        la                      a1,
                MENSAJE_ERROR_MEMORIA_OBUFFER # Cargo en a1 la
                direccion de memoria donde se encuentra el
                mensaje a cargar.
1242        li                      a2,
                BYTES_MENSAJE_ERROR_MEMORIA_OBUFFER  # Cargo en
                a2 la cantidad de bytes a escribir.
1243        li                      v0, SYS_write
1244        syscall                                         # No
                 controlo error porque sale de por si de la
                funcion por error.
1245
1246        # myfree(ibuffer)
1247        lw                      a0,24($fp)
                                            # Cargo en a0
                ibuffer. Parametro de la funcion myfree.
1248        la                      t9,myfree
                                            # Cargo en t9
                direccion de la funcion myfree.
1249        jal                     ra,t9
                                            # Ejecuto la
                funcion myfree.
1250        sw                      zero,24($fp)
                                # Asigno NULL a ibuffer.
1251
1252        li                      v0,ERROR_MEMORY
                                # Cargo en v0 el codigo de error.
1253        sw                      v0,60($fp)
                                            # Guardo el
                codigo de error (en v0) en la direccion de
                memoria 60($fp).
1254        b                       $ReturnPalindrome
                                        # Salto incondicional al
                return de la funcion (goto ReturnPalindrome).
1255    $InitializeBuffers:
1256        # initialize the ibuffer: initializeBuffer(ibytes,
                ibuffer);
```

```
1257        lw                      a0,84($fp)
                                            # Cargo en a0
            ibytes. Parametro de la funcion initializeBuffer.
1258        lw                      a1,24($fp)
                                            # Cargo en a1
            ibuffer. Parametro de la funcion initializeBuffer
            .
1259        la                      t9,initializeBuffer
                                # Cargo en t9 la direccion de la
            funcion initializeBuffer.
1260        jal                     ra,t9
                                            # Ejecuto la
            funcion initializeBuffer.
1261
1262        # initialize the obuffer: initializeBuffer(obytes,
            obuffer);
1263        lw                      a0,92($fp)
                                            # Cargo en a0 0
            bytes. Parametro de la funcion initializeBuffer.
1264        lw                      a1,obuffer
                                            # Cargo en a1 0
            buffer. Parametro de la funcion initializeBuffer.
1265        la                      t9,initializeBuffer
                                # Cargo en t9 la direccion de la
            funcion initializeBuffer.
1266        jal                     ra,t9
                                            # Ejecuto la
            funcion initializeBuffer.
1267
1268        # int * amountSavedInOBuffer = (int *) mymalloc(
            sizeof(int));
1269        li                      a0,4
                                            # Cargo en a0 la
            cantidad de bytes a asignar (por ser un int, son
            4 bytes).
1270        la                      t9,mymalloc
                                            # Cargo en t9 la
            direccion de la funcion mymalloc.
1271        jal                     ra,t9
                                            # Ejecuto la
            funcion mymalloc.
1272        sw                      v0,28($fp)
                                            # En v0 esta el
            resultado de mymalloc. Asigno este resultado a la
             direccion 28($fp), que representaria a la
            variable * amountSavedInOBuffer.
1273
1274        # amountSavedInOBuffer == NULL ?
1275        lw                      v0,28($fp)
                                            # Cargo en v0
            amountSavedInOBuffer
1276        bne                     v0,DIR_NULL,
            $ContinueProcessToLoadIBuffer # If (
            amountSavedInOBuffer != NULL) goto
```

```
            ContinueProcessToLoadIBuffer
1277
1278        # amountSavedInOBuffer is NULL => Mensaje de error
1279        li                      a0,FILE_DESCRIPTOR_STDERR
                                # Cargo en a0
            FILE_DESCRIPTOR_STDERR.
1280        la                      a1,
            MENSAJE_ERROR_MEMORIA_AMOUNT_SAVED # Cargo en a1
            la direccion de memoria donde se encuentra el
            mensaje a cargar.
1281        li                      a2,
            BYTES_MENSAJE_ERROR_MEMORIA_AMOUNT_SAVED # Cargo
            en a2 la cantidad de bytes a escribir.
1282        li                      v0, SYS_write
1283        syscall                                         # No
             controlo error porque sale de por si de la
            funcion por error.
1284
1285        # myfree(ibuffer)
1286        lw                      a0,24($fp)
                                            # Cargo en a0
            ibuffer. Parametro de la funcion myfree.
1287        la                      t9,myfree
                                            # Cargo en t9 la
            direccion de la funcion myfree.
1288        jal                     ra,t9
                                            # Ejecuto la
            funcion myfree.
1289        sw                      zero,24($fp)
                                # Asigno NULL a ibuffer.
1290
1291        # myfree(obuffer)
1292        lw                      a0,obuffer
                                            # Cargo en a0
            obuffer. Parametro de la funcion myfree.
1293        la                      t9,myfree
                                            # Cargo en t9 la
            direccion de la funcion myfree.
1294        jal                     ra,t9
                                            # Ejecuto la
            funcion myfree.
1295        sw                      zero,obuffer
                                # Asigno NULL a obuffer.
1296
1297        li                      v0,ERROR_MEMORY
                                # Cargo en v0 el codigo de error.
1298        sw                      v0,60($fp)
                                            # Guardo el
            codigo de error (en v0) en la direccion de
            memoria 60($fp).
1299        b                       $ReturnPalindrome
                                        # Salto incondicional al
            return de la funcion (goto ReturnPalindrome).
1300    $ContinueProcessToLoadIBuffer:
```

```
1301        # amountSavedInOBuffer[0] = 0;
1302        lw                      v0,28($fp)
                                            # Cargo en v0
            amountSavedInOBuffer.
1303        sw                      zero,0(v0)
                                        # Asigno a
            amountSavedInOBuffer el valor 0.
1304
1305        # int rdoProcess = OKEY;
1306        sw                      zero,32($fp)
                                        # Asigno a rdoProcess,
            que esta en la direccion 32($fp), el valor OKEY.
1307
1308        # int end = FALSE;
1309        sw                      zero,36($fp)
                                        # Asigno a end, que esta
            en la direccion 36($fp), el valor FALSE.
1310
1311        # int error = FALSE;
1312        sw                      zero,40($fp)
                                        # Asigno a error, que
            esta en la direccion 40($fp), el valor FALSE.
1313    $WhilePalindrome:
1314        # end == FALSE ?
1315        lw                      v0,36($fp)
                                            # Cargo en v0 end
            .
1316        bne                     v0,FALSE,$myfreeBuffers
                                # If (end != FALSE) goto
            myfreeBuffers.
1317
1318    #       # end is FALSE
1319
1320    #       # error == FALSE ?
1321        lw                      v0,40($fp)
                                            # Cargo en v0 end
            .
1322        bne                     v0,FALSE,$myfreeBuffers
                                # If (error != FALSE) goto
            myfreeBuffers.
1323
1324    #       #  error is FALSE
1325
1326        # Within the while
1327
1328        # int completeDelivery = FALSE;
1329        sw                      zero,44($fp)
                                        # Guardo FALSE en la
            direccion 44($fp), que representaria la variable
            completeDelivery.
1330
1331        # int bytesReadAcum = 0;
1332        sw                      zero,48($fp)
                                        # Guardo 0 en la
```

```
            direccion 48($fp), que representaria la variable
            bytesReadAcum.
1333
1334    #       #  size_t bytesToRead = ibytes;
1335            lw                      v0,84($fp)
                                                # Cargo en v0
            ibytes, que esta en la direccion 84($fp).
1336            sw                      v0,52($fp)
                                                # Guardo ibytes (
            que esta en v0) en la direccion 52($fp), que
            representaria la variable bytesToRead.
1337    $WhileLoadIBuffer:
1338            # completeDelivery == FALSE ?
1339            lw                      v0,44($fp)
                                                # Cargo en v0
            completeDelivery.
1340            bne                     v0,FALSE,$VerifyIfWriteOFile
                    # If (completeDelivery != FALSE) goto
            VerifyIfWriteOFile.
1341
1342            # end == FALSE
1343            lw                      v0,36($fp)
                                                # Cargo en v0 end
            .
1344            bne                     v0,FALSE,$VerifyIfWriteOFile
                    # If (end != FALSE) goto VerifyIfWriteOFile.
1345
1346            # Read iterative
1347
1348            # int bytesRead = read(ifd, ibuffer + bytesReadAcum,
                 bytesToRead);
1349            lw                      v1,24($fp)
                                                # Cargo en v1
            ibuffer.
1350            lw                      v0,48($fp)
                                                # Cargo en v0
            bytesReadAcum.
1351            addu            v0,v1,v0
                                        # Guardo en v0 el
            resultado de ibuffer + bytesReadAcum.
1352            lw                      a0,80($fp)
                                                # Cargo en a0 ifd
            . Parametro de la funcion read.
1353            move            a1,v0
                                        # Cargo en a1 la
            direccion del buffer a donde se van a guardar los
             bytes leidos (ibuffer + bytesReadAcum).
            Parametro de la funcion read.
1354            lw                      a2,52($fp)
                                                # Cargo en a2
            bytesToRead. Parametro de la funcion read.
1355            li                      v0, SYS_read
1356            syscall                                         #
            Seria read: int bytesRead = read(ifd, ibuffer +
```

```
            bytesReadAcum, bytesToRead);
1357
1358        # Controlo errores y cantidad de bytes leidos. v0
                contiene el numero de caracteres leidos (es
                negativo si hubo error y es 0 si llego a fin del
                archivo).
1359        sw                      v0,56($fp)
                                            # Guardo en la
                direccion de memoria 56($fd) el resultado de la
                funcion read, que estaria representado por la
                variable bytesRead.
1360
1361        # bytesRead == -1 ?
1362        lw                      v1,56($fp)
                                            # Cargo en v1
                bytesRead.
1363        li                      v0,-1
                                            # Cargo en v0 -1
                para la comparacion.
1364        bne                     v1,v0,
                $ContinueValidationResultRead # If (bytesRead !=
                -1) goto ContinueValidationResultRead.
1365
1366        # bytesRead is -1 => Mensaje de error.
1367        li                      a0,FILE_DESCRIPTOR_STDERR
                                # Cargo en a0
                FILE_DESCRIPTOR_STDERR.
1368        la                      a1,
                MENSAJE_ERROR_LECTURA_ARCHIVO # Cargo en a1 la
                direccion de memoria donde se encuentra el
                mensaje a cargar.
1369        li                      a2,
                BYTES_MENSAJE_ERROR_LECTURA_ARCHIVO  # Cargo en
                a2 la cantidad de bytes a escribir.
1370        li                      v0, SYS_write
1371        syscall                                         # No
                 controlo error porque sale de por si de la
                funcion por error.
1372
1373        # myfree(ibuffer)
1374        lw                      a0,24($fp)
                                            # Cargo en a0
                ibuffer. Parametro de la funcion myfree.
1375        la                      t9,myfree
                                            # Cargo la
                direccion de la funcion myfree.
1376        jal                     ra,t9
                                            # Ejecuto la
                funcion myfree.
1377        sw                      zero,24($fp)
                                # Asigno NULL a ibuffer.
1378
1379        # myfree(obuffer)
```

```
1380        lw                      a0,obuffer
                                            # Cargo en a0
            obuffer. Parametro de la funcion myfree.
1381        la                      t9,myfree
                                            # Cargo la
            direccion de la funcion myfree.
1382        jal                     ra,t9
                                            # Ejecuto la
            funcion myfree.
1383        sw                      zero,obuffer
                                # Asigno NULL a obuffer.
1384
1385        # myfree(amountSavedInOBuffer)
1386        lw                      a0,28($fp)
                                            # Cargo en a0
            amountSavedInOBuffer. Parametro de la funcion
            myfree.
1387        la                      t9,myfree
                                            # Cargo la
            direccion de la funcion myfree.
1388        jal                     ra,t9
                                            # Ejecuto la
            funcion myfree.
1389        sw                      zero,28($fp)
                                # Asigno NULL a
            amountSavedInOBuffer.
1390
1391        # lexico != NULL ?
1392        lw                      v0,lexico
                                            # Cargo en v0
            lexico.
1393        beq                     v0,DIR_NULL,$ReturnErrorRead
                    # If (lexico == NULL) goto ReturnErrorRead.
1394
1395        # lexico is not NULL
1396        lw                      a0,lexico
                                            # Cargo en a0
            lexico. Parametro de la funcion myfree.
1397        la                      t9,myfree
                                            # Cargo la
            direccion de la funcion myfree.
1398        jal                     ra,t9
                                            # Ejecuto la
            funcion myfree.
1399        sw                      zero,lexico
                                        # Asigno NULL a lexico.
1400    $ReturnErrorRead:
1401        li                      v0,ERROR_READ
                                        # Cargo en v0 el codigo
            de ERROR_READ.
1402        sw                      v0,60($fp)
                                            # Guardo en la
            direccion 60($fp) el resultado de la funcion
            palindrome, que en este caso es un error.
```

```
1403        b                       $ReturnPalindrome
                                        # Salto incondicional al
            return de la funcion palindrome.
1404    $ContinueValidationResultRead:
1405        lw                      v0,56($fp)
                                            # Cargo en v1
            bytesRead.
1406        bne                     v0,zero,
            $ContinueAccumulatingBytesRead # If (bytesRead !=
             0) goto ContinueAccumulatingBytesRead
1407        li                      v0,TRUE
1408        sw                      v0,36($fp)
                                            # Asigno a la
            variable end, guardada en 36($fp), TRUE.
1409    $ContinueAccumulatingBytesRead:
1410        # bytesReadAcum += bytesRead;
1411        lw                      v1,48($fp)
                                            # Cargo en v1
            bytesReadAcum.
1412        lw                      v0,56($fp)
                                            # Cargo en v0
            bytesRead.
1413        addu            v0,v1,v0
                                        # Sumo bytesReadAcum con
            bytesRead y guardo resultado en v0.
1414        sw                      v0,48($fp)
                                            # Guardo el
            resultado de la suma en bytesReadAcum.
1415
1416        # bytesToRead = ibytes - bytesReadAcum;
1417        lw                      v1,84($fp)
                                            # Cargo en v1
            ibytes.
1418        lw                      v0,48($fp)
                                            # Cargo en v0
            bytesReadAcum.
1419        subu            v0,v1,v0
                                        # Resto ibytes con
            bytesReadAcum y guardo resultado en v0, para
            saber cuandos bytes restan por leer del archivo.
1420        sw                      v0,52($fp)
                                            # Asigno a
            bytesToRead el resultado de la resta.
1421
1422        # bytesToRead == 0 ?
1423        lw                      v0,52($fp)
                                            # Cargo en v0
            bytesToRead.
1424        bne                     v0,zero,$WhileLoadIBuffer
                              # If (bytesToRead != 0) goto
            WhileLoadIBuffer
1425
1426        # bytesToRead is 0.
1427        li                      v0,TRUE
```

```
1428        sw                      v0,44($fp)
                                            # Asigno a
            completeDelivery TRUE.
1429        b                       $WhileLoadIBuffer
                                        # Salto incondicional al
            comienzo del while para cargar el buffer con los
            datos del archivo (goto WhileLoadIBuffer).
1430    $VerifyIfWriteOFile:
1431        # Verifico si tengo datos en el buffer de entrada
            para verificar palindromos,
1432        # guardar en el buffer de salida y en el archivo de
            salida si corresponde.
1433
1434        # ibuffer != NULL && ibuffer[0] != '\0' ?
1435
1436        # ibuffer != NULL ?
1437        lw                      v0,24($fp)
                                            # Cargo en v0
            ibuffer.
1438        beq                     v0,DIR_NULL,$WhilePalindrome
                # If (ibuffer == NULL) goto WhilePalindrome.
1439
1440        # ibuffer is not NULL
1441
1442        # ibuffer[0] != '\0' ?
1443        lw                      v0,24($fp)
                                            # Cargo en v0
            ibuffer.
1444        lb                      v0,0(v0)
                                            # Cargo en
            contenido de la primer posicion de ibuffer en v0.
1445        beq                     v0,zero,$WhilePalindrome
                              # If (ibuffer[0] == '\0') goto
            WhilePalindrome
1446
1447        # ibuffer[0] is not equal '\0'
1448
1449        # int resultProcessWrite = executePalindromeWrite(
            ibuffer, amountSavedInOBuffer);
1450        lw                      a0,24($fp)
                                            # Cargo en a0
            ibuffer. Parametro de la funcion
            executePalindromeWrite.
1451        lw                      a1,28($fp)
                                            # Cargo en a1
            amountSavedInOBuffer. Parametro de la funcion
            executePalindromeWrite.
1452        la                      t9,executePalindromeWrite
                             # Cargo en t9 la direccion de
            memoria en donde se encuentra la funcion
            executePalindromeWrite.
1453        jal                     ra,t9
                                            # Ejecuto la
            funcion executePalindromeWrite.
```

```
1454        sw                      v0,56($fp)
                                            # El resultado de
                la ejecucion de la funcion
            executePalindromeWrite esta en v0. Guardo este
            resultado en la direccion de memoria 56($fp), que
                representaria a la variable resultProcessWrite.
1455
1456        # resultProcessWrite == LOAD_I_BUFFER ?
1457        lw                      v1,56($fp)
                                            # Cargo en v1
            resultProcessWrite.
1458        li                      v0,LOAD_I_BUFFER
1459        bne                     v1,v0,
            $ContinueValidationResultExecutePalinWrite # If (
            resultProcessWrite != LOAD_I_BUFFER) goto
            ContinueValidationResultExecutePalinWrite.
1460
1461        # resultProcessWrite is equal LOAD_I_BUFFER
1462
1463        # initializeBuffer(ibytes, ibuffer);
1464        lw                      a0,84($fp)
                                            # Cargo en a0
            ibytes. Parametro de la funcion initializeBuffer.
1465        lw                      a1,24($fp)
                                            # Cargo en a1
            ibuffer. Parametro de la funcion initializeBuffer
            .
1466        la                      t9,initializeBuffer
                                # Cargo en t9 la direccion de la
            funcion initializeBuffer.
1467        jal                     ra,t9
                                            # Ejecuto la
            funcion initializeBuffer.
1468    $ContinueValidationResultExecutePalinWrite:
1469        # (resultProcessWrite == ERROR_MEMORY ||
            resultProcessWrite == ERROR_WRITE) ?
1470
1471        # resultProcessWrite == ERROR_MEMORY ?
1472        lw                      v1,56($fp)
                                            # Cargo en v1
            resultProcessWrite.
1473        li                      v0,ERROR_MEMORY
1474        beq                     v1,v0,
            $LoadErrorOfExecutePalinWrite # If (
            resultProcessWrite == ERROR_MEMORY) goto
            LoadErrorOfExecutePalinWrite.
1475
1476        # resultProcessWrite is not equal ERROR_MEMORY
1477
1478        # resultProcessWrite == ERROR_WRITE ?
1479        lw                      v1,56($fp)
                                            # Cargo en v1
            resultProcessWrite.
1480        li                      v0,ERROR_WRITE
```

```
1481        beq                     v1,v0,
                $LoadErrorOfExecutePalinWrite # If (
                resultProcessWrite == ERROR_WRITE) goto
                LoadErrorOfExecutePalinWrite.
1482
1483        # No hay errores
1484        b                       $WhilePalindrome
                                        # Vuelvo a intentar
                entrar al loop.
1485    $LoadErrorOfExecutePalinWrite:
1486    #       # error = TRUE;
1487        li                      v0,TRUE
1488        sw                      v0,40($fp)
                                                # Asigno a la
                variable error TRUE.
1489
1490        # rdoProcess = resultProcessWrite;
1491        lw                      v0,56($fp)
                                                # Cargo en v0
                resultProcessWrite.
1492        sw                      v0,32($fp)
                                                # Asigno a la
                variable rdoProcess resultProcessWrite.
1493
1494        b                       $WhilePalindrome
                                        # Vuelvo a intentar
                entrar al loop.
1495    $myfreeBuffers:
1496        # (ibuffer != NULL) ?
1497        lw                      v0,24($fp)
                                                # Cargo en v0
                ibuffer.
1498        beq                     v0,DIR_NULL,
                $myfreeOBufferInPalindrome # If (ibuffer == NULL)
                 goto myfreeOBufferInPalindrome
1499
1500        # ibuffer es not NULL
1501        lw                      a0,24($fp)
                                                # Cargo en a0
                ibuffer. Parametro de la funcion myfree.
1502        la                      t9,myfree
                                                # Cargo en t9 la
                direccion de la funcion myfree.
1503        jal                     ra,t9
                                                # Ejecuto la
                funcion myfree.
1504        sw                      zero,24($fp)
                                  # Asigno a ibuffer NULL.
1505    $myfreeOBufferInPalindrome:
1506        # (obuffer != NULL) ?
1507        lw                      v0,obuffer
                                                # Cargo en v0
                obuffer.
```

```
1508        beq                     v0,DIR_NULL,
                $myfreeLexicoInPalindrome # If (obuffer == NULL)
                goto myfreeLexicoInPalindrome.
1509
1510        # obuffer is not NULL
1511
1512        # (amountSavedInOBuffer != NULL && (*
                amountSavedInOBuffer) > 0) ?
1513
1514        # (amountSavedInOBuffer != NULL) ?
1515        lw                      v0,28($fp)
                                            # Cargo en v0
                amountSavedInOBuffer.
1516        beq                     v0,DIR_NULL,
                $myfreeLexicoPalin  # If (amountSavedInOBuffer ==
                 NULL) goto myfreeLexicoPalin.
1517
1518        # amountSavedInOBuffer is not NULL
1519
1520        # ((*amountSavedInOBuffer) > 0) ?
1521        lw                      v0,28($fp)
                                            # Cargo en v0
                amountSavedInOBuffer.
1522        lw                      v0,0(v0)
                                            # Cargo el
                contenido de lo apuntado por amountSavedInOBuffer
                 en v0.
1523        blez            v0,$myfreeLexicoPalin
                        # If ((* amountSavedInOBuffer) <= 0) goto
                 myfreeLexicoPalin.
1524
1525        # (*amountSavedInOBuffer) is greater then 0
1526
1527        # int rdoWrite = writeOBufferInOFile(
                amountSavedInOBuffer);
1528        lw                      a0,28($fp)
                                            # Cargo en a0
                amountSavedInOBuffer. Parametro de la funcion
                writeOBufferInOFile.
1529        la                      t9,writeOBufferInOFile
                                # Cargo en t9 la direccion de la
                funcion writeOBufferInOFile.
1530        jal                     ra,t9
                                            # Ejecuto la
                funcion writeOBufferInOFile.
1531        sw                      v0,56($fp)
                                            # En v0 esta el
                resultado de writeOBufferInOFile (que seria la
                variable rdoWrite). Guado esto en la direccion
                56($fp).
1532
1533        # (rdoWrite != OKEY) ?
1534        lw                      v0,56($fp)
                                            # Cargo en v0
```

```
            rdoWrite.
1535        beq                     v0,OKEY,$myfreeLexicoPalin
                                # If (rdoWrite == OKEY) goto
            myfreeLexicoPalin.
1536
1537        # rdoWrite is OKEY.
1538
1539        # rdoProcess = rdoWrite;
1540        lw                      v0,56($fp)
                                            # Cargo en v0
            rdoWrite.
1541        sw                      v0,32($fp)
                                            # Asigno a la
            variable rdoProcess rdoWrite.
1542    $myfreeLexicoPalin:
1543        # myfree(obuffer);
1544        lw                      a0,obuffer
                                            # Cargo en a0
            obuffer. Parametro de la funcion myfree.
1545        la                      t9,myfree
                                            # Cargo en t9 la
            direccion de la funcion myfree.
1546        jal                     ra,t9
                                            # Ejecuto la
            funcion myfree.
1547        sw                      zero,obuffer
                                # Asigno NULL a obuffer.
1548    $myfreeLexicoInPalindrome:
1549        # (lexico != NULL) ?
1550        lw                      v0,lexico
                                            # Cargo en v0
            lexico.
1551        beq                     v0,DIR_NULL,
            $myfreeAmountSavedInOBuffer # If (lexico == NULL)
             goto myfreeAmountSavedInOBuffer.
1552
1553        # lexico is not NULL
1554        lw                      a0,lexico
                                            # Cargo en a0
            lexico. Parametro de la funcion myfree.
1555        la                      t9,myfree
                                            # Cargo en t9 la
            direccion de la funcion myfree.
1556        jal                     ra,t9
                                            # Ejecuto la
            funcion myfree.
1557        sw                      zero,lexico
                                        # Asigno NULL a lexico.
1558    $myfreeAmountSavedInOBuffer:
1559        # (amountSavedInOBuffer != NULL) ?
1560        lw                      v0,28($fp)
                                            # Cargo en v0
            amountSavedInOBuffer.
```

```
1561        beq                     v0,DIR_NULL,
                $LoadReturnPalindrome # If (amountSavedInOBuffer
                == NULL) goto LoadReturnPalindrome.
1562
1563        # amountSavedInOBuffer is not NULL
1564
1565        # myfree(amountSavedInOBuffer);
1566        lw                      a0,28($fp)
                                            # Cargo en a0
                amountSavedInOBuffer. Parametro de la funcion
                myfree.
1567        la                      t9,myfree
                                            # Cargo en t9 la
                direccion de la funcion myfree.
1568        jal                     ra,t9
                                            # Ejecuto la
                funcion myfree.
1569        sw                      zero,28($fp)
                                        # Asigno NULL a
                amountSavedInOBuffer.
1570    $LoadReturnPalindrome:
1571        lw                      v0,32($fp)
                                            # Cargo en v0
                rdoProcess.
1572        sw                      v0,60($fp)
                                            # Cargo en la
                direccion 60($fp) el resultado de la funcion
                palindrome, que en este caso seria rdoProcess.
1573    $ReturnPalindrome:
1574        lw                      v0,60($fp)
1575        move            sp,$fp
1576        lw                      ra,72(sp)
1577        lw                      $fp,68(sp)
1578        addu            sp,sp,80
1579        j                       ra
                                                    # Jump
                and return
1580        .end    palindrome
1581
1582
1583    ## Variables auxiliares
1584
1585        .data
1586        .align 2
1587        isize:          .space 4
1588
1589        .align 2
1590        osize:          .space 4
1591
1592        .align 2
1593        oFileDescriptor:        .space 4
1594
1595
1596        .rdata
```

```
1597            .align  3
1598    doubleWord:
1599            .word   0
1600            .word   1073741824
1601
1602            .globl          lexico
1603            .section        .bss
1604            .align  2
1605            .type   lexico, @object
1606            .size   lexico, 4
1607    lexico:
1608            .space  4
1609
1610            .globl  quantityCharacterInLexico
1611            .globl  quantityCharacterInLexico
1612            .align  2
1613            .type   quantityCharacterInLexico, @object
1614            .size   quantityCharacterInLexico, 4
1615    quantityCharacterInLexico:
1616            .space  4
1617
1618            .globl  savedInOFile
1619            .globl  savedInOFile
1620            .align  2
1621            .type   savedInOFile, @object
1622            .size   savedInOFile, 4
1623    savedInOFile:
1624            .space  4
1625
1626            .globl  obuffer
1627            .globl  obuffer
1628            .align  2
1629            .type   obuffer, @object
1630            .size   obuffer, 4
1631    obuffer:
1632            .space  4
1633
1634            .globl  bytesLexico
1635            .globl  bytesLexico
1636            .align  2
1637            .type   bytesLexico, @object
1638            .size   bytesLexico, 4
1639    bytesLexico:
1640            .space  4
1641
1642
1643    ## Mensajes de error
1644
1645            .rdata
1646            .align  2
1647    MENSAJE_ERROR_MEMORIA_LEXICO:
1648            .ascii  "[Error] Hubo un error en memoria (lexico).
                       \n\000"
1649
```

```
1650            .align  2
1651    MENSAJE_ERROR_MEMORIA_OBUFFER:
1652            .ascii  "[Error] Hubo un error de asignacion de
                    memoria (obuffer)"
1653            .ascii  ". \n\000"
1654
1655            .align  2
1656    MENSAJE_ERROR_MEMORIA_IBUFFER:
1657            .ascii  "[Error] Hubo un error de asignacion de
                    memoria (ibuffer)"
1658            .ascii  ". \n\000"
1659
1660            .align  2
1661    MENSAJE_ERROR_MEMORIA_AMOUNT_SAVED:
1662            .ascii  "[Error] Hubo un error de asignacion de
                    memoria (amountSa"
1663            .ascii  "ved). \n\000"
1664
1665            .align  2
1666    MENSAJE_ERROR_LECTURA_ARCHIVO:
1667            .ascii  "[Error] Hubo un error en la lectura de
                    datos del archivo"
1668            .ascii  ". \n\000"
```

4. Ejecución

A continuación algunos de los comandos válidos para la ejecución del programa:
Comandos usando un archivo de entrada y otro de salida

```
$ tp1 -i input.txt -o output.txt
```

```
$ tp1 --input input.txt --output output.txt
```

Comando para la salida standard

```
$ tp1 -i input.txt
```

Comando para el ingreso standard

```
$ tp1 -o output.txt
```

Por defecto los tamaños del buffer in y buffer out son 1 byte. puede específicar el tamaño a usar los mismos en la llamada.

```
$ tp1 -i input.txt -o output.txt -I 10 -O 10
```

-I: indica el tamaño (bytes) a usar por el buffer in
-O: indica el tamaño (bytes) a usar por el buffer out

4.1. Comandos para ejecución

Desde el netBSD ejecutar:
Para compilar el código

```
$ gcc -Wall -o tp1 tp1.c process.S
```

-Wall: activa los mensajes de warnning
-o: indica el archivo de salida.

Para obtener el código MIPS32 del proyecto c:

```
$ gcc -Wall -O0 -S -mrnames tp1.c
```

-S: detiene el compilador luego de generar el código assembly
-mrnames: indica al compilador que genere la salida con nombre de registros
-O0: indica al compilador que no aplique optimizaciones.

4.2. Análisis sobre tiempo de ejecución

Comando para la medición del tiempo (time):

```
$ time ./tp1 -i ../input-large.txt -I 10 -O 10
```

Se midieron y se tuvieron en cuenta los tiempo transcurridos entre distintas ejecuciones cambiando los parámetros de entrada de buffer in y buffer out. Para medir se usó la instrucción "time"la cual arroja los tiempos efectivamente consumidos por el CPU en la ejecución del programa. A continuación una tabla con los valores medidos:

Tamaño de archivo usado apróximadamente 1,70 MB.
Tamaño de línea en archivo apróximadamente: 1 byte * 450 char = 450 byte(caracteres/línea).

Cómo puede verse en la figura las ejecuciones iniciales con valores bajos de lectura y escritura(buffer 1 byte) tienen tiempos de respuesta del programa elevados; mientras que a medida que se aumenta el tamaño del buffer los tiempos van creciendo hasta un limite asintótico alrededor de 7 segundos.
Es de notar que un pequeño aumento en el tamaño del buffer(in/out) aumenta la performance considerablemente.

4.3. Comandos para ejecución de tests

Comando para ejecutar el test automático

```
$ bash test-automatic.sh
```

La salida debería ser la siguiente(todos los test OK):

id	stream input	stream output	real time[s]	user time[s]	sys time[s]
1	1	1	82,53	9,74	36,68
2	2	2	50,55	8,16	21,11
3	3	2	30,70	7,78	8,72
4	5	5	19,34	6,94	4,67
5	10	10	11,42	7,10	1,10
6	50	50	8,90	6,85	0,66
7	100	100	7,97	6,97	0,34
8	300	300	7,45	6,76	0,22
9	600	600	7,53	6,90	0,18
10	1000	1000	7,58	6,89	0,25
11	1500	1500	7,33	6,84	0,24
12	2000	2000	7,40	6,86	0,23
13	2500	2500	7,43	6,94	0,11
14	3500	3500	7,18	6,70	0,16

Cuadro 1: Valores de exe medidos(time).

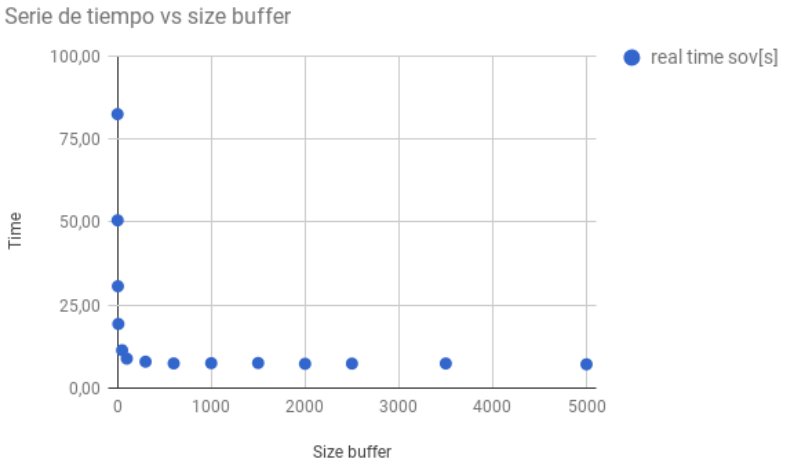

Figura 1: Gráfico de incidencia del buffer

```
################################################################################

########################## Tests automaticos
    ################################
################################################################################

###----------###    COMIENZA test ejercicio 1 del informe.
    ###----------###
###----------###        STDIN ::: FILE OUTPUT
    ###----------###
OK
###----------###    FIN test ejercicio 1 del informe.
    ###----------###
###------------------------------------------------------------------------###

###------------------------------------------------------------------------###

###------------------------------------------------------------------------###

###----------###    COMIENZA test ejercicio 2 del informe.
    ###----------###
###----------###        FILE INPUT ::: STDOUT
    ###----------###
OK
###----------###    FIN test ejercicio 2 del informe.
    ###----------###
###------------------------------------------------------------------------###

###------------------------------------------------------------------------###

###------------------------------------------------------------------------###

###----------###    COMIENZA test con -i - -o -
    ###----------###
###----------###        STDIN ::: STDOUT
    ###----------###
OK
###----------###    FIN test con -i - -o -
    ###----------###
###------------------------------------------------------------------------###

###------------------------------------------------------------------------###

###------------------------------------------------------------------------###

###----------###    COMIENZA test palabras con acentos
    ###----------###
OK
###----------###    FIN test palabras con acentos
    ###----------###
###------------------------------------------------------------------------###

###------------------------------------------------------------------------###

###------------------------------------------------------------------------###

###----------###    COMIENZA test con caritas
    ###----------###
OK
###----------###    FIN test con caritas
    ###----------###
###------------------------------------------------------------------------###

###------------------------------------------------------------------------###

###------------------------------------------------------------------------###

###----------###    COMIENZA test con entrada estandar
    ###----------###
OK
###----------###    FIN test con entrada estandar
    ###----------###
```

```
###------------------------------------------------------------------------###

###------------------------------------------------------------------------###

###------------------------------------------------------------------------###

###----------###    COMIENZA test con salida estandar
     ###----------###
OK
###----------###    FIN test con salida estandar
     ###----------###
###------------------------------------------------------------------------###

###------------------------------------------------------------------------###

###------------------------------------------------------------------------###

###----------###  COMIENZA test con entrada y salida estanda
     ###----------###
OK
###----------###  FIN test con entrada y salida estanda
     ###----------###
###------------------------------------------------------------------------###

###------------------------------------------------------------------------###

###------------------------------------------------------------------------###

###----------###  COMIENZA test menu version (-V)
     ###----------###
OK
###----------###  FIN test menu version (-V)
     ###----------###
###------------------------------------------------------------------------###

###------------------------------------------------------------------------###

###------------------------------------------------------------------------###

###----------###  COMIENZA test menu version (--version)
     ###----------###
OK
###----------###  FIN test menu version (--version)
     ###----------###
###------------------------------------------------------------------------###

###------------------------------------------------------------------------###

###------------------------------------------------------------------------###

###----------###  COMIENZA test menu help (-h)
     ###----------###
OK
###----------###  FIN test test menu help (-h)
     ###----------###
###------------------------------------------------------------------------###

###------------------------------------------------------------------------###

###------------------------------------------------------------------------###

###----------###  COMIENZA test menu help (--help)
     ###----------###
OK
###----------###  FIN test menu help (--help)
     ###----------###
###------------------------------------------------------------------------###

###------------------------------------------------------------------------###

###------------------------------------------------------------------------###

##############################################################################
```

```
########################## Tests automaticos
    ################################
################################################################################

#----------# COMIENZA test con /-o -i - #----------#
OK
```

5. Stack frame de funciones

A continuación los stack frame de las diferentes funciones:

int isKeywords(char character)	
Posición	**Contenido**
24	
20	fp
16	gp
12	Guardo el res de la fun TRUE (v0) en la dir de mem 12
8	v0=char
4	
0	

Stack frame 4

void initializeBuffer(size_t bytes, char * buffer)	
Posición	**Contenido**
28	charBuffer
24	bytes
20	fp
16	gp
12	
8	0
4	
0	

Stack frame 5

int toLowerCasse(char word)	
Posición	**Contenido**
24	ra
20	fp
16	gp
12	Resultado de la función: TRUE que es igual a 1 o FALSE que es igual a 0
8	character

Figura 2: Stack frame 1

char toLowerCasse(char word)	
Posición	**Contenido**
24	ra
20	fp
16	gp
12	
8	word
4	
0	

Figura 3: Stack frame 2

int verifyPalindromic(char * word, int quantityCharacterInWord)	
Posición	**Contenido**
76	quantityCharacterInWord
72	* word
68	
64	ra
60	fp
56	gp
52	Resultado de la función: TRUE que es igual a 1 o FALSE que es igual a 0
48	last
44	validPalindromic
40	idx
36	
32	middle
28	
25	Cuando quantityCharacterInWord es igual a 2 contiene a lastCharacter. Cuando quantityCharacterInWord es mayor a 2 contiene a firstCharacter.
24	Cuando quantityCharacterInWord es igual a 2 contiene a firstCharacter. Cuando quantityCharacterInWord es mayor a 2 contiene a lastCharacter.
20	
16	
12	79
8	
4	
0	

Figura 4: Stack frame 3

int palindrome(int ifd, size_t ibytes, int ofd, size_t obytes)	
Posición	**Contenido**
92	obytes
88	ofd
84	ibytes
80	ifd
76	
72	ra
68	fp
64	gp
60	Resultado de la función palindrome.
56	Se reutiliza para guardar varios valores: 1) bytesRead: resultado del read al archivo ifd. 2) resultProcessWrite. 3) rdoWrite
52	bytesToRead
48	bytesReadAcum
44	completeDelivery
40	error
36	end
32	rdoProcess
28	* amountSavedInOBuffer
24	* ibuffer
20	
16	
12	
8	
4	

Stack frame 6

int executePalindromeWrite(char * ibuffer, int * amountSavedInOBuffer)	
Posición	**Contenido**
84	* amountSavedInOBuffer
80	* ibuffer
76	
72	ra
68	fp
64	gp
60	Resultado de la función executePalindromeWrite.
56	Se reutiliza para guardar varios valores: 1) rdoWrite. 2) i.
52	i
48	amountToSaved
44	Se reutiliza para guardar varios valores: 1) bytesLexicoPreview. 2) resultado de la funcion verifyPalindromic.
40	character
36	rdo
32	idx
28	loadIBuffer
24	findEnd
20	
16	
12	
8	
4	
0	

Stack frame 7

void * myRealloc(void * ptr, size_t tamanyoNew, int tamanyoOld)	
Posición	**Contenido**
72	tamanyoOld
68	tamanyoNew
64	* ptr
60	
56	ra
52	fp
48	gp
44	
40	Resultado de la función myRealloc.
36	* src
32	* tmp
28	end
24	Dirección de memoria asignada con mymalloc a ptrNew
20	
16	
12	
8	
4	
0	

Stack frame 8

int writeOBufferInOFile(int * amountSavedInOBuffer)	
Posición	**Contenido**
64	* amountSavedInOBuffer
60	
56	ra
52	fp
48	gp
44	
40	Resultado de la función writeOBufferInIFile.
36	bytesWrite
32	bytesToWrite
28	bytesWriteAcum
24	completeDelivery
20	
16	
12	
8	
4	
0	

Stack frame 9

6. Conclusiones

A través del presente trabajo se logro realizar una implementación pequeña de un programa c y assembly MIPS32. La invocación desde un programa assembly a un programa c; la implementación de una función malloc, free y realloc en código assembly, sin hacer uso de la implementación c. La forma de llamar a funciones de

Por otro lado se logró familiarizarse con la implementación de assembly MIPS y con la ABI.

La implementación de la función palindroma con un buffer permitió ver que en función de la cantidad de caracteres leídos cada vez, el tiempo de ejecución del programa disminuia considerablemente. Al mismo tiempo la mejora en el tiempo de ejecución tiene un límite a partir del cual un aumento en el tamaño del buffer no garantiza ganancia en la ejecución del programa.

Referencias

[1] Intel Technology & Research, "Hyper-Threading Technology," 2006, http://www.intel.com/technology/hyperthread/.

[2] J. L. Hennessy and D. A. Patterson, "Computer Architecture. A Quantitative Approach," 3ra Edición, Morgan Kaufmann Publishers, 2000.

[3] J. Larus and T. Ball, "Rewriting Executable Files to Mesure Program Behavior," Tech. Report 1083, Univ. of Wisconsin, 1992.

https://es.wikipedia.org/wiki/Pal

Apéndice: código C(process.c)

```
1
2  #include "process.h"
3
4
5  #define FALSE 0
6  #define TRUE 1
7  #define LEXICO_BUFFER_SIZE 10
8
9  enum ParameterState {
10          OKEY = 0, INCORRECT_QUANTITY_PARAMS = 1,
                ERROR_MEMORY = 2, ERROR_READ = 3, ERROR_WRITE =
                4, LOAD_I_BUFFER = 5
11 };
12
13 size_t isize;
14 size_t osize;
15 int oFileDescriptor;
16 char * lexico = NULL;
17 int quantityCharacterInLexico = 0;
18 int savedInOFile = FALSE;
19 char * obuffer = NULL;
```

```
20 int bytesLexico = 0;
21
22 char toLowerCase(char word) {
23         /* ASCII:
24          *              A - Z = [65 - 90]
25          *              a - z = [97 - 122]
26          *              0 - 9 = [48 - 57]
27          *              - =  45
28          *              _ = 95
29          */
30         if (word >= 65 && word <= 90) {
31                 word += 32;
32         }
33
34         return word;
35 }
36
37 int verifyPalindromic(char * word, int
       quantityCharacterInWord) {
38         if (word == NULL || quantityCharacterInWord <= 0) {
39                 return FALSE;
40         }
41
42         if (quantityCharacterInWord == 1) {
43                 // The word has one character
44                 return TRUE;
45         }
46
47         if (quantityCharacterInWord == 2) {
48                 char firstCharacter = toLowerCase(word[0]);
49                 char lastCharacter = toLowerCase(word[1]);
50                 if (firstCharacter != lastCharacter) {
51                         return FALSE;
52                 }
53
54                 return TRUE;
55         }
56
57         double middle = (double)quantityCharacterInWord / 2;
58         int idx = 0;
59         int validPalindromic = TRUE;
60         int last = quantityCharacterInWord - 1;
61         while(idx < middle && last > middle &&
               validPalindromic == TRUE) {
62                 char firstCharacter = toLowerCase(word[idx])
                       ;
63                 char lastCharacter = toLowerCase(word[last])
                       ;
64                 if (firstCharacter != lastCharacter) {
65                         validPalindromic = FALSE;
66                 }
67
68                 idx ++;
69                 last --;
```

```
70          }
71
72          return validPalindromic;
73  }
74
75  int isKeywords(char character) {
76          /* ASCII:
77           *              A - Z = [65 - 90]
78           *              a - z = [97 - 122]
79           *              0 - 9 = [48 - 57]
80           *              - =  45
81           *              _ = 95
82           */
83          if ((character >= 65 && character <= 90) || (
                character >= 97 && character <= 122)
84                          || (character >= 48 && character <=
                                57)
85                          || character == 45 || character ==
                                95) {
86                  return TRUE;
87          }
88
89          return FALSE;
90  }
91
92  void * myRealloc(void * ptr, size_t tamanyoNew, int
        tamanyoOld) {
93          if (tamanyoNew <= 0) {
94                  free(ptr);
95                  ptr = NULL;
96
97                  return NULL;
98          }
99
100         void * ptrNew = (void *) malloc(tamanyoNew);
101         if (ptrNew == NULL) {
102                 free(ptr);
103                 ptr = NULL;
104
105                 return NULL;
106         }
107
108         if (ptr == NULL) {
109                 return ptrNew;
110         }
111
112         int end = tamanyoNew;
113         if (tamanyoOld < tamanyoNew) {
114                 end = tamanyoOld;
115         }
116
117         char *tmp = ptrNew;
118         const char *src   = ptr;
119
```

```
120         while (end--) {
121                 *tmp = *src;
122                 tmp++;
123                 src++;
124         }
125 
126         free(ptr);
127         ptr = NULL;
128 
129         return ptrNew;
130 }
131 
132 void initializeBuffer(size_t bytes, char * buffer) {
133     // initialize the buffer
134     int i;
135     for(i = 0; i < bytes; ++i){
136         buffer[i] = '\0';
137     }
138 }
139 
140 int writeOBufferInOFile(int * amountSavedInOBuffer) {
141         int completeDelivery = FALSE;
142         int bytesWriteAcum = 0;
143         int bytesToWrite = (*amountSavedInOBuffer);
144         while (completeDelivery == FALSE) {
145                 int bytesWrite = write(oFileDescriptor,
                        obuffer + bytesWriteAcum, bytesToWrite);
146                 if (bytesWrite < 0) {
147                         return ERROR_WRITE;
148                 }
149 
150                 bytesWriteAcum += bytesWrite;
151                 bytesToWrite = (*amountSavedInOBuffer) -
                        bytesWriteAcum;
152 
153                 if (bytesToWrite <= 0) {
154                         completeDelivery = TRUE;
155                 }
156         }
157 
158         return OKEY;
159 }
160 
161 int executePalindromeWrite(char * ibuffer, int *
        amountSavedInOBuffer) {
162         int findEnd = FALSE;
163         int loadIBuffer = FALSE;
164         int idx = 0;
165         int rdo = OKEY;
166         while (findEnd == FALSE && loadIBuffer == FALSE) {
167                 char character = ibuffer[idx];
168                 if (character == '\0') {
169                         findEnd = TRUE;
170                 }
```

```
171
172                 if (findEnd != TRUE && isKeywords(character)
                        == TRUE) {
173                     if (lexico == NULL) {
174                             lexico = malloc(
                                    LEXICO_BUFFER_SIZE *
                                    sizeof(char));
175                             bytesLexico =
                                    LEXICO_BUFFER_SIZE;
176                     } else if (quantityCharacterInLexico
                            >= bytesLexico) {
177                             int bytesLexicoPreview =
                                    bytesLexico;
178                             bytesLexico +=
                                    LEXICO_BUFFER_SIZE;
179                             lexico = myRealloc(lexico,
                                    bytesLexico*sizeof(char),
                                     bytesLexicoPreview);
180                     }
181
182                     if (lexico == NULL) {
183                             fprintf(stderr, "[Error]
                                    Hubo un error en memoria
                                    (lexico). \n");
184                             return ERROR_MEMORY;
185                     }
186
187                     lexico[quantityCharacterInLexico] =
                            character;
188                     quantityCharacterInLexico ++;
189                 } else if (quantityCharacterInLexico > 0) {
190                     int itsPalindromic =
                            verifyPalindromic(lexico,
                            quantityCharacterInLexico);
191                     if (itsPalindromic == TRUE) {
192                             int amountToSaved = (*
                                    amountSavedInOBuffer) +
                                    quantityCharacterInLexico
                                    ;
193                             if ((*amountSavedInOBuffer)
                                    > 0 || savedInOFile ==
                                    TRUE) {
194                                     amountToSaved ++; //
                                            Es para el
                                           separador
195                             }
196                             if (amountToSaved > osize) {
197                                     /*
198                                      * Tomo la decision
                                            de pedir mas
                                            memoria para
                                            bajar el lexico
                                            completo
```

```
199                                      * y luego rearmo el
                                             buffer de
                                         salida y
                                         reinicio la
                                         cantidad
                                         guardada en 0.
200                                      */
201                                     obuffer = myRealloc(
                                            obuffer,
                                            amountToSaved*
                                            sizeof(char), (*
                                            amountSavedInOBuffer
                                            ));
202                                     if ((*
                                            amountSavedInOBuffer
                                            ) > 0 ||
                                            savedInOFile ==
                                            TRUE) {
203                                              obuffer[*
                                                    amountSavedInOBuffer
                                                    ] = '\n';
204                                              *
                                                    amountSavedInOBuffer
                                                     = (*
                                                    amountSavedInOBuffer
                                                    ) + 1;
205                                     }
206
207                                     int i;
208                                     for (i = 0; i <
                                            quantityCharacterInLexico
                                            ; ++i) {
209                                              obuffer[*
                                                    amountSavedInOBuffer
                                                    ] =
                                                    lexico[i
                                                    ];
210                                              *
                                                    amountSavedInOBuffer
                                                     = (*
                                                    amountSavedInOBuffer
                                                    ) + 1;
211                                     }
212
213                                     int rdoWrite =
                                            writeOBufferInOFile
                                            (
                                            amountSavedInOBuffer
                                            );
214                                     if (rdoWrite != OKEY
                                            ) {
215                                              return
                                                    rdoWrite;
216                                     }
```

```
217
218                                     *
                                            amountSavedInOBuffer
                                             = 0;
219                                     savedInOFile = TRUE;
220                                     if (obuffer != NULL)
                                             {
221                                         free(obuffer
                                                );
222                                         obuffer =
                                                NULL;
223                                     }
224
225                                     obuffer = (char *)
                                            malloc(osize*
                                            sizeof(char));
226                                     if (obuffer == NULL)
                                             {
227                                         fprintf(
                                                stderr, "
                                                [Error]
                                                Hubo un
                                                error de
                                                asignacion
                                                 de
                                                memoria (
                                                obuffer).
                                                 \n");
228                                         return
                                                ERROR_MEMORY
                                                ;
229                                     }
230
231                                     // initialize the
                                            obuffer
232                                     initializeBuffer(
                                            osize, obuffer);
233                                 } else {
234                                     if ((*
                                            amountSavedInOBuffer
                                            ) > 0 ||
                                            savedInOFile ==
                                            TRUE) {
235                                         obuffer[*
                                                amountSavedInOBuffer
                                                ] = '\n';
236                                         *
                                                amountSavedInOBuffer
                                                 = (*
                                                amountSavedInOBuffer
                                                ) + 1;
237                                     }
238
239                                     int i;
```

```
240                                         for (i = 0; i <
                                                quantityCharacterInLexico
                                                ; ++i) {
241                                                 obuffer[*
                                                        amountSavedInOBuffer
                                                        ] =
                                                        lexico[i
                                                        ];
242                                                 *
                                                        amountSavedInOBuffer
                                                         = (*
                                                        amountSavedInOBuffer
                                                        ) + 1;
243                                         }
244                                 }
245                         }
246
247                         free(lexico);
248                         lexico = NULL;
249                         quantityCharacterInLexico = 0;
250                 }
251
252                 if ((idx + 1) == isize) {
253                         loadIBuffer = TRUE;
254                         rdo = LOAD_I_BUFFER;
255                 } else {
256                         idx ++;
257                 }
258         }
259
260         return rdo;
261 }
262
263 int palindrome(int ifd, size_t ibytes, int ofd, size_t
        obytes) {
264         isize = ibytes;
265         osize = obytes;
266         oFileDescriptor = ofd;
267         char * ibuffer = (char *) malloc(ibytes*sizeof(char)
                );
268         if (ibuffer == NULL) {
269                 fprintf(stderr, "[Error] Hubo un error de
                        asignacion de memoria (ibuffer). \n");
270                 return ERROR_MEMORY;
271         }
272
273         obuffer = (char *) malloc(obytes*sizeof(char));
274         if (obuffer == NULL) {
275                 fprintf(stderr, "[Error] Hubo un error de
                        asignacion de memoria (obuffer). \n");
276                 free(ibuffer);
277                 ibuffer = NULL;
278                 return ERROR_MEMORY;
279         }
```

```
280
281         // initialize the ibuffer
282         initializeBuffer(ibytes, ibuffer);
283     // initialize the obuffer
284         initializeBuffer(obytes, obuffer);
285
286         int * amountSavedInOBuffer = (int *) malloc(sizeof(
                int));
287         if (amountSavedInOBuffer == NULL) {
288                 fprintf(stderr, "[Error] Hubo un error de
                        asignacion de memoria (amountSaved). \n")
                        ;
289                 free(ibuffer);
290                 ibuffer = NULL;
291                 free(obuffer);
292                 obuffer = NULL;
293                 return ERROR_MEMORY;
294         }
295         amountSavedInOBuffer[0] = 0;
296
297         int rdoProcess = OKEY;
298         int end = FALSE;
299         int error = FALSE;
300         while (end == FALSE && error == FALSE) {
301                 int completeDelivery = FALSE;
302                 int bytesReadAcum = 0;
303                 size_t bytesToRead = ibytes;
304                 // Lleno el buffer de entrada
305                 while (completeDelivery == FALSE && end ==
                        FALSE) {
306                         int bytesRead = read(ifd, ibuffer +
                                bytesReadAcum, bytesToRead);
307                         if (bytesRead == -1) {
308                                 fprintf(stderr, "[Error]
                                        Hubo un error en la
                                        lectura de datos del
                                        archivo. \n");
309                                 free(ibuffer);
310                                 ibuffer = NULL;
311                                 free(obuffer);
312                                 obuffer = NULL;
313                                 free(amountSavedInOBuffer);
314                                 amountSavedInOBuffer = NULL;
315                                 if (lexico != NULL) {
316                                         free(lexico);
317                                         lexico = NULL;
318                                 }
319                                 return ERROR_READ;
320                         }
321
322                         if (bytesRead == 0) {
323                                 end = TRUE;
324                         }
325
```

```
326                         bytesReadAcum += bytesRead;
327                         bytesToRead = ibytes - bytesReadAcum
                                ;
328 
329                         if (bytesToRead <= 0) {
330                                 completeDelivery = TRUE;
331                         }
332                 }
333 
334                 if (ibuffer != NULL && ibuffer[0] != '\0') {
335                         int resultProcessWrite =
                                executePalindromeWrite(ibuffer,
                                amountSavedInOBuffer);
336                         if (resultProcessWrite ==
                                LOAD_I_BUFFER) {
337                                 // initialize the ibuffer
338                                 initializeBuffer(ibytes,
                                        ibuffer);
339                         }
340                         if (resultProcessWrite ==
                                ERROR_MEMORY ||
                                resultProcessWrite == ERROR_WRITE
                                ) {
341                                 error = TRUE;
342                                 rdoProcess =
                                        resultProcessWrite;
343                         }
344                 }
345         }
346 
347         if (ibuffer != NULL) {
348                 free(ibuffer);
349                 ibuffer = NULL;
350         }
351 
352         if (obuffer != NULL) {
353                 if (amountSavedInOBuffer != NULL && (*
                        amountSavedInOBuffer) > 0) {
354                         int rdoWrite = writeOBufferInOFile(
                                amountSavedInOBuffer);
355                         if (rdoWrite != OKEY) {
356                                 rdoProcess = rdoWrite;
357                         }
358                 }
359 
360                 free(obuffer);
361                 obuffer = NULL;
362         }
363 
364         if (lexico != NULL) {
365                 free(lexico);
366                 lexico = NULL;
367         }
368 
```

```
369         if (amountSavedInOBuffer != NULL) {
370                 free(amountSavedInOBuffer);
371                 amountSavedInOBuffer = NULL;
372         }
373 
374         return rdoProcess;
375 }
```