UBA FACULTAD DE INGENIERÍA

66.20 Organización de Computadoras

Trabajo Práctico 1

2^{do} Cuatrimestre 2017

Integrantes:

Rodriquez Longhi, Federico	93336
federico.rlonghi@gmail.com	
Deciancio, Nicolás Andrés	92150
nicodec_89@hotmail.com	
Marshall, Juan Patricio	95471
juan.pmarshall@gmail.com	

Índice

1. Introducción

2. Documentación

El uso del programa se compone de las siguientes opciones que le son pasadas por parámetro:

- `-h` o `--help`: muestra la ayuda.
- `-V` o `--version`: muestra la versión.
- `-i` o `--input`: recibe como parámetro un archivo de texto como entrada. En caso de que no usar esta opción, se toma como entrada la entrada estándar.
- `-o` o `--output`: recibe como parámetro un archivo de texto como salida. En caso de que no usar esta opción, se toma como salida la salida estándar.
- `-I` o `--ibuf-bytes`: recibe como parámetro el tamaño en bytes del buffer de entrada.
- `-O` o `--iobuf-bytes`: recibe como parámetro el tamaño en bytes del buffer de salida.

3. Compilación

Dentro del directorio raiz se encuentra un Makefile. Ejecuntando `make` se compilara el programa y se generará el ejecutable `tp1`.

4. Pruebas

Para las pruebas se proveen de dos scripts que las ejecutan. El primer script `test.sh` ejecuta los ejemplos del enunciado.

El segundo script `test_p.sh` ejecuta las pruebas propias. Este archivo esta diseñado para poder agregar pruebas de forma sencilla, simplemente se debe agregar una linea en el sector de pruebas de la siguiente manera:

```
make_test <nombre><entrada de texto><salida esperada>
```

Este script crea los archivos correspondientes en la carpeta tests (dentro del directorio sobre el cual se ejecuta). Los archivos creados son de la forma:

- `test-<nombre del test>_in`: archivo de entrada
- `test-<nombre del test>_out`: archivo de salida generado por el programa
- `test-<nombre del test>_expected`: archivo de salida esperado

4.1. Corridas de Prueba

A continuación se muestran las corridas de prueba generadas por el script:

```
1
2  Compiling Source
3  Compilation Success
4  Starting Tests
5
6  Test: one_letter_a
7  Test passed
8
9  Test: empty_file
10 Test passed
11
12 Test: no_palindroms
13 Test passed
14
15 Test: todos_palindromos
16 Test passed
17
18 Test: varias_lineas
19 Test passed
20
21 Test: all_letters
22 Test passed
23
24 Test: case_sensitive
25 Test passed
26
27 Test: numbers_and_letters
28 Test passed
29
30 Test: text_with_dash
31 Test passed
32
33 --------------------
34 All 9 tests passed!!!
35 --------------------
```

5. Conclusión

6. Código en C

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <ctype.h>
4 #include <unistd.h>
5 #include <getopt.h>
6 #include <errno.h>
```

```
7  #include <string.h>
8
9  extern size_t mystrlen(const char*);
10 extern int palindrome(int, size_t, int, size_t);
11
12 /*int N = 100;
13
14 typedef struct {
15   char *array;
16   size_t used;
17   size_t size;
18   size_t initial_size;
19 } WordArray;
20
21 void init_array(WordArray *a, size_t initial_size){
22     a->array = (char*)malloc(sizeof(char*)*initial_size);
23     a->used = 0;
24     a->size = initial_size;
25     a->initial_size = initial_size;
26     a->array[0] = '\0';
27 }
28
29 void clear_array(WordArray *a){
30     free(a->array);
31     a->array = (char*)malloc(sizeof(char*)*a->initial_size);
32     a->used = 0;
33     a->size = a->initial_size;
34     a->array[0] = '\0';
35 }
36
37 void insert_char(WordArray *a, char c){
38     if (a->used == a->size){
39         size_t new_size = a->size*2;
40         a->array = (char*)realloc(a->array, sizeof(char*)*
               new_size);
41         a->size = new_size;
42     }
43     a->array[a->used]=c;
44     a->array[a->used+1]='\0';
45     a->used++;
46 }
47
48 void free_array(WordArray *a){
49     free(a->array);
50     a->array = NULL;
51     a->size = a->used = 0;
52 }*/
53
54 /* imprimir el uso de tp0 */
55 void print_usage() {
56     printf("Usage: tp0 -i [input_file] -o [output_file]\n");
57 }
58
59 /* imprimir la pagina de ayuda */
```

```
60
61  void print_help() {
62      printf("\tUsage:\n"
63        "\t\ttp0 -h\n"
64        "\t\ttp0 -V\n"
65        "\t\ttp0 [options]\n"
66        "\tOptions:\n"
67        "\t\t-V, --version\tPrint version and quit.\n"
68        "\t\t-h, --help\tPrint this information.\n"
69        "\t\t-i, --input\tLocation of the input file.\n"
70        "\t\t-o, --output\tLocation of the output file.\n"
71          "\t\t-I, --ibuf-bytes\tByte count of the input buffer.\
              n"
72          "\t\t-O, --obuf-bytes\tByte count of the output buffer
              .\n"
73        "\tExamples:\n"
74        "\t\ttp0 -i ~/input -o ~/output\n");
75  }
76
77  /* imprimir la version del programa */
78  void print_version(){
79      printf("tp1 1.0\n");
80  }
81
82  /* funcion para determinar si una palabra es capicua o no */
83  /*int es_capicua(WordArray *word){
84
85      size_t len = word->used;
86      if (len == 0) return 0;
87
88      int capicua = 1;
89      int i=0;
90      while (capicua && i < (len / 2)){
91          if (tolower(word->array[i]) != tolower(word->array[len
                - i - 1])){
92              return 0;
93          }
94          i++;
95      }
96      return 1;
97  }*/
98
99  /* Lee la palabra de un archivo y la devuelve en word */
100 /*int read_word (FILE *f, WordArray *word) {
101     int c = fgetc(f);
102     if (c == EOF) return 0;
103     while (1){
104         if ( (65 <= c && c <= 90) || //letras mayusculas
105              (97 <= c && c <= 122) || //letras minusculas
106              (48 <= c && c <= 57) || //numeros
107              c == 95 || c == 45){ //barras
108             insert_char(word,c);
109         }else{
110             return 1;
```

```
111         }
112         c = fgetc(f);
113     }
114     return 0;
115 }*/
116 
117 int main(int argc, char *argv[]) {
118 
119     int opt= 0;
120 
121     int help = -1;
122     int version = -1;
123     int input = -1;
124     int output = -1;
125     int ibuf = 1;
126     int obuf = 1;
127 
128     char *input_filename = NULL;
129     char *output_filename = NULL;
130 
131     // especificacion de las opciones
132     static struct option long_options[] = {
133         {"help",         no_argument,       0,  'h' },
134         {"version",      no_argument,       0,  'V' },
135         {"input",        required_argument, 0,  'i' },
136         {"output",       required_argument, 0,  'o' },
137         {"ibuf-bytes",   required_argument, 0,  'I' },
138         {"obuf-bytes",   required_argument, 0,  'O' },
139         {0,              0,                 0,  0   }
140     };
141 
142     int long_index = 0;
143 
144     // evaluacion de los parametros enviados al programa
145     while ((opt = getopt_long(argc, argv,"hVui:o:I:O:",
146                    long_options, &long_index )) != -1) {
147         switch (opt) {
148             case 'h' :
149                 help = 0;
150                 break;
151             case 'V' :
152                 version = 0;
153                 break;
154             case 'i' :
155                 input = 0;
156                 input_filename = optarg;
157                 break;
158             case 'o' :
159                 output = 0;
160                 output_filename = optarg;
161                 break;
162             case 'I':
163                 ibuf = atoi(optarg);
164                 break;
```

```
165             case 'O':
166                 obuf = atoi(optarg);
167                 break;
168             case '?':
169               exit(1);
170             default:
171                 print_usage();
172                 exit(EXIT_FAILURE);
173         }
174     }
175
176     // procesamiento de los parametros
177     if (help == 0) {
178         print_help();
179         exit(0);
180     }
181     else if (version == 0) {
182         print_version();
183         exit(0);
184     }
185
186     /* Si no se recibe parametro de ayuda o version se ejecuta
            el programa */
187
188     // estableciendo los archivos de entrada y salida
189     FILE *input_file = stdin;
190     FILE *output_file = stdout;
191
192     if (input == 0){
193         input_file = fopen(input_filename,"r");
194         if (input_file == NULL) {
195             printf ("can't open input file, errno = %d\n",
                    errno);
196             return 1;
197         }
198     }
199     if (output == 0){
200         output_file = fopen(output_filename,"w");
201         if (output_file == NULL) {
202             printf ("Can't open output file, errno = %d\n",
                    errno);
203             return 1;
204         }
205     }
206
207     /* ejecucion del programa */
208     char *msg = "Estoy probando mips!!!\n";
209   write(1, msg, mystrlen(msg));
210
211     int file_in = fileno(input_file);
212     int file_out = fileno(output_file);
213
214     int a = palindrome(file_in,ibuf,file_out,obuf);
215     printf("file_in: %i\nibuf: %zu\nfile_out: %i\nobuf: %zu\n",
```

```
216                 file_in,ibuf,file_out,obuf);
217     printf("a: %i\n",a);
218     //write(1,a,sizeof(int));
219
220     /*WordArray word;
221     init_array(&word,N);
222     int i = read_word(input_file, &word);
223     while (i == 1){
224         if (es_capicua(&word)){
225           fprintf(output_file,"%s\n", word.array);
226         }
227         clear_array(&word);
228         i = read_word(input_file, &word);
229     }
230     free_array(&word);*/
231
232     // cierro los archivos
233
234     if (input == 0){
235         fclose(input_file);
236     }
237     if (output == 0){
238         fclose(output_file);
239     }
240
241     printf("sali bien\n");
242
243     return 0;
244 }
```

7. Función Palindromo en MIPS

```
1  # palindrome.S - ver tp1.c.
2  #
3  # $Date: 2017/09/24 17:12:06 $
4
5  #include <sys/syscall.h>
6  #include <mips/regdef.h>
7
8    .text
9    .align  2
10
11   .globl  palindrome
12   .ent  palindrome
13 palindrome:
14   .frame $fp, 48, ra
15   .set  noreorder
16   .cpload t9
17   .set  reorder
18
19     subu  sp, sp, 48
```

```
20
21    .cprestore 32
22    sw  ra, 44(sp)
23    sw  $fp, 40(sp)
24    sw  gp, 36(sp)
25
26      sw  a0, 0(sp)       #file_in
27      sw  a1, 4(sp)       #ibuf
28      sw  a2, 8(sp)       #file_out
29      sw  a3, 12(sp)      #obuf
30
31      move  $fp, sp
32
33      lw  a0, 4(sp)   # cargo el parametro para mymalloc
34      jal mymalloc  # llamo a la funcion
35      sw  v0, 24(sp)    # guardo la direccion de memoria
            reservada para el buffer de ENTRADA
36
37      lw  a0, 12(sp)    # cargo el parametro para mymalloc
38      jal mymalloc  # llamo a la funcion
39      sw  v0, 28(sp)    # guardo la direccion de memoria
            reservada para el buffer de SALIDA
40
41  prog:
42      li  v0, SYS_read    # ver dentro de <sys/syscall.h>.
43    lw  a0, 0(sp)       # a0: file descriptor number.
44    la  a1, 24(sp)      # a1: data pointer.
45    lw  a2, 4(sp)          # a2: available space.
46    syscall
47
48    add   t0, zero, zero    # indice para recorrer el buffer (i)
49    la  t1, 24(sp)    # cargo la direccion del buffer de etrada
          en t1
50    lb  t2, 0(t1)     # cargo el primer valor del buffer de
          entrada en t2
51    lw  t7, 4(sp)
52    beqz  t2, buffer_write
53
54  buffer_word_read:
55    beq   t0, t7, prog
56    move  t8, t0    # guardo el indice del inicio de la palabra
          en t8
57    slti  t3, t2, 48
58    bnez  t3, palindrome_check
59    slti  t3, t2, 58
60    slti  t4, t2, 65
61    slt   t3, t3, t4
62    bnez  t3, palindrome_check
63    slti  t3, t2, 91
64    slti  t4, t2, 97
65    slt   t3, t3, t4
66    bnez  t3, palindrome_check
67    slti  t3, t2, 123
68    beqz  t3, palindrome_check
```

```
69
70    move  t9, t0    # guardo el indice del final de la palabra en
          t9
71    add   t0, t0, 1 # i = i + 1
72    add   t1, t1, 1
73    lb    t2, 0(t1)
74    j     buffer_word_read  # sigo leyendo letras hasta econtrar
          un caracter que no sea alfanumerico
75
76
77  palindrome_check:
78    add   t0, t0, 1 # i = i + 1
79
80
81  buffer_write:
82      li  v0, SYS_write   # ver dentro de <sys/syscall.h>.
83    lw  a0, 8(sp)       # a0: file descriptor number.
84    la  a1, 28(sp)      # a1: output data pointer.
85    lw  a2, 12(sp)          # a2: output byte size.
86    syscall
87
88      li  t0, 10          #guardo salto de linea en t0
89      sw  t0, 52(sp)      #guardo el salto de linea en el stack
90
91      li  v0, SYS_write   # ver dentro de <sys/syscall.h>.
92    lw  a0, 8(sp)       # a0: file descriptor number.
93    la  a1, 52(sp)      # a1: output data pointer.
94    li  a2, 1           # a2: output byte size.
95    syscall
96
97
98  free_buffer:
99    la  a0, 24(sp)
100   jal myfree    # libero la memoria reservada para el buffer de
          entrada
101
102   la  a0, 28(sp)
103   jal myfree    # libero la memoria reservada para el buffer de
          salida
104
105 palindrome_return:
106     move    v0, t0
107   move  sp, $fp
108   lw      ra, 44(sp)
109   lw      $fp, 40(sp)
110   addu  sp, sp, 48
111   j     ra
112   .end  palindrome
```

8. Enunciado

El enunciado se encuentra anexado al final de este documento.