"Conjunto de instrucciones MIPS"

Federico Colangelo, Padrón Nro. 89.869
federico.colangelo@semperti.com

Facundo Rossi, Padrón Nro. 86.707
frossi85@gmail.com

Federico Martín Rossi, Padrón Nro. 92.086
federicomrossi@gmail.com

1er. Cuatrimestre 2013
66.20 Organización de Computadoras
Facultad de Ingeniería, Universidad de Buenos Aires

Índice

1. Introducción **1**

2. Compilación **1**

3. Utilización **1**
3.1. Implementación en C . . . 1
3.2. Implementación en Assembly . . . 1

4. Implementaciones **1**
4.1. Implementación en C . . . 2
4.1.1. Algoritmo *Bubblesort* . . . 2
4.1.2. Algoritmo *Shellsort* . . . 3
4.2. Implementación en Assembly . . . 5
4.2.1. Algoritmo *Shellsort* . . . 5

5. Debugging **6**

6. Pruebas **7**

7. Conclusiones **7**

Appendices **8**

A. Implementación completa en lenguaje C **8**
A.1. *tp0.c.* Implementación del main del programa . . . 8
A.2. *bubblesort.h.* Declaración de la librería Bubblesort . . . 12
A.3. *bubblesort.c.* Definición de la librería Bubblesort . . . 13
A.4. *shellsort.h.* Declaración de la librería Shellsort . . . 14
A.5. *shellsort.c.* Definición de la librería Shellsort . . . 14
A.6. *fileloader.h.* Declaración de la librería File Loader . . . 15
A.7. *fileloader.c.* Definición de la librería File Loader . . . 16

B. Implementación completa de Shellsort en assembly MIPS **18**
B.1. *tp0.c.* Implementación del main del programa . . . 18
B.2. *shellsort.S.* Implementación del algoritmo Shellsort . . . 19
B.3. *compare.S.* Implementación del comparador de palabras . . . 20
B.4. *swap.S.* Implementación del comparador de palabras . . . 21

1. Introducción

En este trabajo presentamos la comparación entre dos algoritmos de ordenamiento: el *Bubblesort* y el *Shellsort*.

Para realizar la comparación entre ambos se realizó la implementación en lenguaje C de cada uno. A su vez, para comparar la performance entre código de alto nivel y código nativo, se hizo la implementación de shellsort en assembler MIPS, con el fin de poder comparar los tiempos de ejecución de ambos programas.

Todo el trabajo se realizo en una plataforma *NetBSD/MIPS-32* mediante el *GXEmul* [1]

Todos los archivos y códigos fuente aquí mencionados, así como también el presente informe, pueden ser descargados de la sección Downloads del repositorio del grupo[1].

2. Compilación

La herramienta para compilar tanto el código asembly como C será el *GCC* [2]. Para tratar de equiparar al máximo el *S* generado por ambas implementaciones, se utilizará el flag de gcc "-O0" para que no realice optimizaciones sobre el código en lenguaje C.

Para automatizar las tareas de compilación se hace uso de la herramienta *GNU Make* . Los Makefiles utilizados para la compilación se incluyen junto al resto de los archivos fuentes del presente trabajo [2].

3. Utilización

En los siguientes apartados se especifica la forma en la que deben ser ejecutados los programas implementados tanto en C como en assembly.

3.1. Implementación en C

El resultado de compilación con "make" será un programa ejecutable, de nombre *tp0*, que podrá ser invocado con los siguientes parámetros:

- *-h*: Imprime ayuda para la utilización del programa;
- *-V*: Imprimer la versión actual del programa;
- *-b [ARGS]*: El programa recibe nombres de archivos de texto o strings ingresados por *stdin*, ordenandolos utilizando el algoritmo bubblersort. Para utilizar *stdin* deberá omitirse [ARGS] y luego introducir las palabras;
- *-s [ARGS]*: El programa recibe nombres de archivos de texto o strings ingresados por *stdin*, ordenandolos utilizando el algoritmo shellsort. Para utilizar *stdin* deberá omitirse [ARGS] y luego introducir las palabras.

3.2. Implementación en Assembly

El resultado de compilación con "make" será un programa ejecutable, de nombre *tp0*, el cual aceptará un archivo de texto como argumento y lo ordenará con el algoritmo Shellsort.

4. Implementaciones

En lo que sigue de la sección, se presentarán los códigos fuente de las implementaciones de los algoritmos. Aquellos lectores interesados en la implementación completa de los dos programas, pueden dirigirse a los apéndices ubicados al final del presente informe. Recordamos que se han separado las implementaciones en dos de manera de poder mantener un orden entre ambos.

[1] URI del Repositorio: `https://code.google.com/p/tp-orga-computadoras/`

[2] Los archivos se encuentran separados según la implemetación a la que pertenecen, por lo que habrán dos Makefiles distintos, uno para la implementación en lenguaje C y otro para la implementación en assembly

4.1. Implementación en C

La implementación del programa fue divida en los siguientes módulos:

- **tp0**: Programa principal responsable de interpretar los comandos pasados por la terminal de modo que realice las tareas solicitadas por el usuario. Su principal función es encadenar el funcionamiento de los otros módulos y mostrar por pantalla el resultado obtenido;
- **fileloader**: Módulo encargado de levantar un archivo de texto desde el filesystem y convertirlo en un array donde cada posición es una palabra. Permite además dimensionar el tamaño en memoria necesario para cargar todo el archivo como un arreglo de palabras;
- **bubblesort**: Módulo encargado de levantar un archivo de texto desde el filesystem y convertirlo en un array donde cada posición es una palabra. Permite además dimensionar el tamaño en memoria necesario para cargar todo el archivo como un arreglo de palabras;
- **shellsort**: Módulo encargado de implementar el algoritmo de shellsort. Recibe como paramentros un arreglo de palabras desordenado y el tamaño del mismo. Como resultado devuelve dicho arreglo ordenado.

4.1.1. Algoritmo *Bubblesort*

En el *Código 1* se muestra el header de la librería, donde se declara la función Bubblesort, mientras que en el *Código 2* se muestra la definición de la librería.

Código 1: *"bubblesort.h"*

```
/* ******************************************************************************
 * ******************************************************************************
 *
 * LIBRERIA BUBBLESORT
 *
 * ******************************************************************************
 * ******************************************************************************/



#ifndef BUBBLESORT_H
#define BUBBLESORT_H



// Funcion que aplica el algoritmo de ordenamiento Bubblesort para ordenar un
// arreglo de palabras.
// PRE: 'words' es un puntero a un arreglo de punteros a caracter; 'arraysize'
// es el tamanio de dicho arreglo.
// POST: el arreglo 'words' queda ordenado.
void bubblesort(char* words[], int arraysize);



#endif
```

Código 2: *"bubblesort.c"*

```
/* ******************************************************************************
 * ******************************************************************************
 *
 * LIBRERIA BUBBLESORT
 *
 * ******************************************************************************
 * ******************************************************************************/


#include "bubblesort.h"
#include <stdbool.h>
#include <string.h>

```

```
15
16 // Funcion que aplica el algoritmo de ordenamiento Bubblesort para ordenar un
17 // arreglo de palabras.
18 // PRE: 'words' es un puntero a un arreglo de punteros a caracter; 'arraysize'
19 // es el tamanio de dicho arreglo.
20 // POST: el arreglo 'words' queda ordenado.
21 void bubblesort(char* words[], int arraysize)
22 {
23   // Variables de procesamiento
24   bool huboIntercambio;
25   int i;
26   int n = arraysize;
27   char* sAux;
28
29   // Recorremos el arreglo haciendo intercambios hasta que ya no se registre
30   // ningun cambio realizado.
31   do
32   {
33     huboIntercambio = false;
34
35     for(i = 1; i < n; i++)
36     {
37       // Si el de indice menor es mayor que el de indice superior, los
38       // intercambiamos
39       if(strcasecmp(words[i-1], words[i]) > 0)
40       {
41         sAux = words[i-1];
42         words[i-1] = words[i];
43         words[i] = sAux;
44
45         // Cambiamos el flag para registrar que hubo un cambio
46         huboIntercambio = true;
47       }
48     }
49
50     // Como el elemento del indice superior se encuentra ya ordenado una
51     // vez finalizada la pasada, se reduce en uno la cantidad de indices
52     // a iterar en la proxima pasada.
53     n -= 1;
54
55   } while(huboIntercambio);
56 }
```

4.1.2. Algoritmo *Shellsort*

En el *Código 3* se muestra el header de la librería, donde se declara la función Shellsort, mientras que en el *Código 4* se muestra la definición de la librería.

Código 3: *"shellsort.h"*

```
1  /* ****************************************************************************
2   * ****************************************************************************
3   *
4   * LIBRERIA SHELLSORT
5   *
6   * ****************************************************************************
7   * ****************************************************************************/
8
9
10
11 #ifndef SHELLSORT_H
12 #define SHELLSORT_H
13
14
15
16 // Funcion que aplica el algoritmo de ordenamiento Shellsort para ordenar un
17 // arreglo de palabras.
18 // PRE: 'words' es un puntero a un arreglo de punteros a caracter; 'arraysize'
19 // es el tamanio de dicho arreglo.
20 // POST: el arreglo 'words' queda ordenado.
21 void shellsort(char* words[], int arraysize);
```

```
#endif
```

Código 4: *"shellsort.c"*

```
/* ****************************************************************************
 * ****************************************************************************
 *
 * LIBRERIA SHELLSORT
 *
 * ****************************************************************************
 * ****************************************************************************/


#include "shellsort.h"
#include <string.h>



// Funcion que aplica el algoritmo de ordenamiento Shellsort para ordenar un
// arreglo de palabras.
// PRE: 'words' es un puntero a un arreglo de punteros a caracter; 'arraysize'
// es el tamanio de dicho arreglo.
// POST: el arreglo 'words' queda ordenado.
void shellsort(char* words[], int arraysize)
{
  int intervalo, k, j, i;
  char* sAux;

  // Tomamos como intervalo el modulo de la mitad del arreglo
  intervalo = arraysize / 2;

  // Iteramos hasta que el intervalo sea nulo
  while(intervalo > 0)
  {
    for(i = intervalo-1; i < arraysize; i++)
    {
      j = i - intervalo;

      // Procesamos hasta caer en un indice fuera de rango del arreglo
      while(j >= 0)
      {
        k = j + intervalo;

        // Si el de indice menor es mayor que el de indice superior, los
        // intercambiamos
        if(strcasecmp(words[j], words[k]) > 0)
        {
          sAux = words[j];
          words[j] = words[k];
          words[k] = sAux;
        }
        else
          j = 0;

        j -= intervalo;
      }
    }

    // Tomamos como nuevo intervalo el modulo de la mitad del intervalo
    // anterior
    intervalo = intervalo / 2;
  }
}
```

4.2. Implementación en Assembly

La implementación del programa fue divida en los siguientes módulos:

- **tp0**: Solamente recibe un texto como argumento por linea de comandos y lo imprime ordenandolo mediante shellsort. Esta implementado en C;
- **fileloader**: Se utiliza el mismo de la otra implementación;
- **swap**: Función implementada en assembler para intercambiar el valor de dos registros recibidos como parametros;
- **compare**: Función similar a strcmp de C. Recibe dos argumentos de tipo texto y decide cual es el que precede alfabeticamente.
- **shellsort**: Implementación en assembler del algoritmo de ordenamiento.

4.2.1. Algoritmo *Shellsort*

En el *Código 5* se muestra la implementación en assembly del algoritmo Shellsort.

Código 5: *"shellsort.S"*

```
#ifndef USE_MIPS_ASSEMBLY
#define USE_MIPS_ASSEMBLY
#include <mips/regdef.h>
#include <sys/syscall.h>

.text
.align 2
.globl shellsort
.extern swap
.extern compare


shellsort:  subu sp,sp,24   # creo stack frame, es de 24 porque necesito guardar ra
    sw ra,16(sp)
    sw $fp,12(sp)
    sw gp,8(sp)
    move $fp,sp

    #Guardo los parametros recibidos en fp
    sw a0,24($fp)   # words = char **
    sw a1,28($fp)     # size

    #Guardo los parametros cargados en fp a variables temporales
    lw s6, 24($fp)    #en 24 esta el words
    lw s7, 28($fp)    #en 28 esta el size

    #Iniciaizo variables locales
    srl s0, s7, 1   # intervalo = size/2
    li s1, 0    # k
    li s2, 0    # j
    li s3, 0    # i
    li s4, 0    # sAux


    #add v0, s0, zero
    #j fin

    #while(intervalo > 0)
loop1:    bgt s0, zero, initloop2 #branch grater than, salta si s0 es mayor a 0
    j fin

    #for(i = intervalo-1; i < arraysize; i++)
initloop2:  addi s3, s0, -1   # i = interlvalo - 1
loop2:    bge s3, s7, endloop2  # si i >= size fin de for, bge=branch greater than or equal
    sub s2, s3, s0    #j = i - intervalo;


    #while(j >= 0)
loop3:    bge s2, zero, loop3task
    addi s3, s3, 1
    j loop2
```

```
52
53
54
55
56 loop3task:  add s1, s2, s0    #k = j + intervalo;
57
58     #Para comparar cadenas, la forma mas sencilla es iterar sobre cada caracter de cada cadena, y restarlos. ↩
           Si el resultado es 0, son
59 #iguales. Si no, a continuacion, si el resultado es> 0, entonces la primera cadena es mayor de que la otra ↩
       cadena, de lo contrario la segunda cadena es inferior y #habria que intercambiarlas. Si te quedas sin ↩
       alguna de las cadenas antes que el otro, y son iguales hasta llegar a ese punto, la cadena mas corta es ↩
       menor que el que #mas.
60     #CARGO LAS CADENAS A COMPARAR
61     #words[j]
62     ##Guardo en variables auxiliares las direcciones de memoria a intercambiar
63     #words = s6
64     #Guardo i y j por q las voy a modificar y no las quiero perder
65     sw s1, 0($fp)
66     sw s2, 4($fp)
67
68
69
70     sll s1, s1, 2
71     sll s2, s2, 2
72     addu s1, s1, s6
73     addu s2, s2, s6
74
75     #Cargo los strings a comparar en ax que luego compare utiliza para trabajar
76     lw a0, 0(s1)    #OPTIMIZAR CODIGO PARA ISAR MENOS ti
77     lw a1, 0(s2)
78     jal compare   #llamo a compare
79
80     lw s1, 0($fp)   #Cargo el valor de k en stack
81     lw s2, 4($fp)   #Cargo el valor de j en stack
82
83     bltz v0, swapping #swap si compare > 0
84     add s2, zero, zero  #j=0
85     j endif
86
87     # poner el valor a utilizar en los registros correctos y llamar a swap
88 swapping: add a0, s6,0
89     add a1, s1, 0
90     add a2, s2, 0
91     jal swap
92
93 endif:    sub s2, s2, s0    #j = j - intervalo;
94     j loop3
95
96
97 endloop2: srl s0, s0, 1   #intervalo = intervalo / 2;
98     j loop1
99
100
101
102     ##Comienzo retorno
103 fin:    move sp, $fp
104     lw gp, 8(sp)
105     lw $fp, 12(sp)
106     lw ra, 16(sp)
107     addiu sp, sp, 24
108     jr ra     #ra = return address es la direccion a la q quiero volver luego de llamar la funcion
109
110 #endif
```

5. Debugging

Para analizar el correcto funcionamiento de los programas se crearon programas adhoc que fueran corroborando el correcto funcionamiento de cada uno de los modulos de manera individual. La idea era simular el concepto de test unitario de lenguajes de alto nivel.

Una vez que todos los módulos funcionaban de la manera esperada, se utilizó el programa principal como test de integración. Finalmente se utilizó la herramienta *Valgrind* para realizar un análisis minucioso del uso de memoria, corrigiendo así las pérdidas de memoria que se presentaban en cada módulo.

6. Pruebas

Para realizar las pruebas de cada algoritmo se procesaron cuatro textos provistos por la cátedra. Estos son "*Alicia en el País de las Maravillas*" (173kB), "*Beowulf*" (220kB), una "*Enciclopedia*" (643kB) y "*Don Quijote*" (2147kB).

A cada ejecución se le midió el tiempo de procesamiento mediante el comando *GNU "time"* [5]. A continuación un ejemplo de su utilización:

```
$ time ./tp0 -b alice.txt
```

Esto imprimirá por pantalla el tiempo de ejecución que fue tabulado para cada caso. En el *Cuadro 1* se muestran los valores de tiempo obtenidos.

Archivo de texto	**bubblesort.c [s]**	**shellsort.c [s]**	**shellsort.S [s]**
alice.txt	571,715	2,207	1,035
beowulf.txt	958,527	3,418	1,500
cyclopedia.txt	>3600	12,184	6,914
elquijote.txt	>3600	48,957	33,098

Cuadro 1: *Tiempos en segundos obtenidos en la ejecución de los distintos algoritmos.*

7. Conclusiones

Al realizar la comparación entre los algoritmos bubblesort y shellsort en C, vemos que la diferencia entre ambos es notable. Estamos hablando de varios ordenes de maginitud.

Esto se debe a que por su codigo bubblesort tiene un coste de $O(n^2)$ mientras que si bien no podemos determinarlo fácilmente para shellsort, ya que depende mucho del gap elegido, estamos hablando de algo cercano a $O(nlog(n))$. Esto hace que para textos grandes no sea una opción viable la utilización de bubblesort, mientras que shellsort prueba ser muy eficiente y el esfuerzo de codificación no es demsiado grande.

En cuanto a la comparación de assembler y C, notamos una ligera ventaja del primero. Creemos que esto se debe a que muchas de las cosas que el algoritmo en C utiliza como variables en memoria principal, assembler utiliza registros los cuales presentan un tiempo de lecto/escritura notablemente menor.

Referencias

[1] The NetBSD project, `http://www.netbsd.org/`

[2] GCC, the GNU Compiler Collection, `http://gcc.gnu.org/`

[3] Bubblesort, `http://en.wikipedia.org/wiki/Bubble_sort`

[4] Shellsort, `http://en.wikipedia.org/wiki/Shell_sort`

[5] time man page, `http://unixhelp.ed.ac.uk/CGI/man-cgi?time`

[6] MIPS ABI, `http://www.sco.com/developers/devspecs/mipsabi.pdf`

[7] J. L. Hennessy and D. A. Patterson, "Computer Architecture. A Quantitative Approach," 4th Edition, Morgan Kaufmann Publishers, 2000.

Apéndices

A. Implementación completa en lenguaje C

A.1. *tp0.c*. Implementación del main del programa

Código 6: *"tp0.c"*

```
/* ******************************************************************************
 * PROGRAMA DE ORDENAMIENTO DE PALABRAS
 * ******************************************************************************
 *
 * Facultad de Ingenieria - UBA
 * 66.20 Organizacion de computadoras
 * Trabajo Practico: Conjunto de instrucciones MIPS
 *
 * ALUMNOS:
 *
 * Facundo Rossi (86707) - frossi85@gmail.com
 * Federico Colangelo (89869) - federico.colangelo@semperti.com
 * Federico Martin Rossi (92086) - federicomrossi@gmail.com
 *
 * ******************************************************************************
 *
 * Programa que permite aplicar metodos de ordenamiento a un texto o textos
 * especificados por el usuario. Estos textos pueden ser ingresados a traves de
 * la entrada estandar o bien a traves de la especificacion de los nombres de
 * archivo que se desean procesar. Cabe resaltar que al especificar varios
 * nombres de archivos, los metodos seran aplicados al conjunto total de los
 * textos contenidos en estos. Es decir, el resultado del ordenamiento sera
 * unico ya que contendra la mezcla de palabras de todos los archivos.
 * Los metodos de ordenamiento soportados por la presente version del programa
 * son:
 *
 *    - Bubblesort
 *    - Shellsort
 *
 *
 *
 * FORMA DE USO
 * ============
 *
 * El programa debera ser ejecutado por consola mediante el siguiente comando:
 *
 *    # ./tp0 [OPCION] [archivos...]
 *
 * donde, la opcion puede ser:
 *
 *  -h,   Muestra la ayuda
 *  -V,   Muestra la informacion de la version
 *  -b,   Indica la utilizacion del algoritmo Bubblesort
 *  -s,   Indica la utilizacion del algoritmo Shellsort
 *
 * y en archivos se debe especificar los nombres de archivo (incluyendo su
 * extension) separados uno del otro por un espacio.
 * En caso de desear ingresar el texto a traves de la entrada estandar, debe
 * utilizarse la siguiente variacion del comando anterior:
 *
 *    # ./tp0 [OPCION]
 *
 *
 */



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

```c
#include "bubblesort.h"
#include "shellsort.h"
#include "fileloader.h"




/* ************************************************************************
 * CONSTANTES
 * ************************************************************************/


// Maximo de caracteres permitidos por texto de entrada estandar
#define MAX_CHARS 1000



/* ************************************************************************
 * FUNCIONES AUXILIARES
 * ************************************************************************/


// Funcion que imprime por pantalla el mensaje de ayuda del programa.
void ayudaImprimirSalidaEstandar()
{
  printf("tp0 [OPTIONS] [files...]\n");
  printf("-h, \t\t\t display this help and exit.\n");
  printf("-V, \t\t\t display version information and exit.\n");
  printf("-b, \t\t\t use the bubblesort algorithm.\n");
  printf("-s, \t\t\t use the shellsort algorithm.\n\n");
  printf("$echo -n echo 'El tractorcito rojo que silbo y bufo' > entrada.txt\n");
  printf("$tp0 -b entrada.txt\n");
  printf("bufo El que rojo silbo tractorcito y\n\n");
  printf("cat letters.txt\n");
  printf("aAbBcCdDeEfFgGhHiIjJkKlLmMnNoOpPqQrRsStTuUvVwWxXyYzZ$\n\n");
  printf("$tp0 letters.txt\n");
  printf("aAbBcCdDeEfFgGhHiIjJkKlLmMnNoOpPqQrRsStTuUvVwWxXyYzZ\n$\n\n");
  printf("$tp0 letters.txt entrada.txt\n");
  printf("aAbBcCdDeEfFgGhHiIjJkKlLmMnNoOpPqQrRsStTuUvVwWxXyYzZ bufo El que rojo silbo tractorcito y\n$\n");
}


// Funcion que imprime por pantalla la version del programa.
void versionImprimirSalidaEstandar()
{
  printf("Programa de ordenamiento de palabras\nVersion: v0.1\n");
}


// Funcion que imprime los resultados del ordenamiento a traves de la salida
// estandar.
// PRE: 'palabras' es un arreglo de palabras; 'arraysize' es el tamanio del
// arreglo
void ordenamientoImprimirSalidaEstandar(char* palabras[], int arraysize)
{
  int i;
  for(i = 0; i < arraysize; printf("%s ", palabras[i++]));
  printf("\n");
}


// Funcion que se encarga leer archivos de texto devolviendo una cadena con
// el contenido de estos.
// PRE: 'iniArchivos' es el indice inicial desde donde 'argv' posee nombres
// de archivo; 'argc' y 'argv' son los parametros de entrada de la funcion
// main.
// POST: se devuelve una unica cadena con el contenido completo de archivos.
char* cargarTextosDeArchivos(int iniArchivos, int argc, char **argv)
{
  int i;
  char * texto = "";

  // Sensamos cada archivo ingresado y concatenamos su contenido
  for(i = iniArchivos; i < argc; i++)
  {
```

```
    // Cargamos contenido de archivo
    char * texto_tmp = file_loader(argv[i]);

    // Recalculamos el tamanio del arreglo
    int len = strlen(texto) + strlen(texto_tmp) + 2;

    // Solicitamos espacio de memoria nuevo
    char * texto_alloc = (char *) malloc(len * sizeof(char));

    // Copiamos contenido antigui y concatenamos el nuevo
    strcpy(texto_alloc, texto);
    if(i > iniArchivos) strcat(texto_alloc, " ");
    strcat(texto_alloc, texto_tmp);
    free(texto_tmp);

    // Liberamos espacio de memoria antiguio
    if(i > iniArchivos) free(texto);

    // Asociamos al nuevo arreglo
    texto = texto_alloc;
  }

  return texto;
}


// Funcion que se encarga de leer el texto ingresado por entrada estandar.
// POST: se devuelve una unica cadena con el texto ingresado.
char* cargarTextosDeEntradaEstandar()
{
  char * texto = (char *) malloc(MAX_CHARS * sizeof(char) + 1);
  if(fgets(texto, MAX_CHARS + 1, stdin) == NULL) return "";

  return texto;
}


void testFileLoader(const char * url)
{
  char * text;
  char ** words = NULL;
  int wordsSize = 0;

  text = file_loader(url);

  if (text != NULL)
  {
    //fputs(text, stdout);
  }

  wordsSize = to_words(text, &words);

  for(int i = 0; i < wordsSize; i++)
  {
    fputs("\n", stdout);
    fputs(words[i], stdout);
    fputs("\n", stdout);
  }
}




/* ****************************************************************************
 * PROGRAMA PRINCIPAL
 * ****************************************************************************/


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

  //File loader variables
  char * text;
  char ** words = NULL;
  int wordsSize = 0;
```

```
215
216  opterr = 0;
217
218  while ((c = getopt(argc, argv, "hVb:s:")) != -1)
219  {
220    switch (c)
221    {
222      // Ayuda
223      case 'h':
224        ayudaImprimirSalidaEstandar();
225        break;
226
227      // Version
228      case 'V':
229        versionImprimirSalidaEstandar();
230        break;
231
232      // Ejecucion de bubblesort
233      case 'b':
234
235        // Cargamos contenidos de archivos
236        text = cargarTextosDeArchivos(2, argc, argv);
237
238        // Si el/los archivos esta/n vacio/s, salimos
239        if (text == NULL) break;
240
241        wordsSize = to_words(text, &words);
242
243        if (wordsSize == -1) break;
244
245        // Ejecutamos bubblesort
246        bubblesort(words, wordsSize);
247
248        // Enviamos a salida estandar
249        ordenamientoImprimirSalidaEstandar(words, wordsSize);
250
251        free(words);
252        free(text);
253        break;
254
255      // Ejecucion de shellsort
256      case 's':
257
258        // Cargamos contenidos de archivos
259        text = cargarTextosDeArchivos(2, argc, argv);
260
261        // Si el/los archivos esta/n vacio/s, salimos
262        if (text == NULL) break;
263
264        wordsSize = to_words(text, &words);
265
266        if (wordsSize == -1) break;
267
268        // Ejecutamos shellsort
269        shellsort(words, wordsSize);
270
271        // Enviamos a salida estandar
272        ordenamientoImprimirSalidaEstandar(words, wordsSize);
273
274        free(words);
275        free(text);
276        break;
277
278      // No se especifica nombre de archivo
279      case '?':
280
281        // Ejecutar bubblesort con texto desde entrada estandar
282        if(optopt == 'b')
283        {
284          // Cargamos texto
285          text = cargarTextosDeEntradaEstandar();
286
287          // Si no se ingreso texto, salimos
288          if (text == NULL) break;
289          wordsSize = to_words(text, &words);
290
```

```
          if (wordsSize == -1) break;

          // Ejecutamos bubblesort
          bubblesort(words, wordsSize);

          // Enviamos a salida estandar
          ordenamientoImprimirSalidaEstandar(words, wordsSize);

          free(words);
          free(text);
          break;
        }

        // Ejecutar shellsort con texto desde entrada
        // estandar
        else if(optopt == 's')
        {
          // Cargamos texto
          text = cargarTextosDeEntradaEstandar();

          // Si no se ingreso texto, salimos
          if (text == NULL) break;
          wordsSize = to_words(text, &words);

          if (wordsSize == -1) break;

          // Ejecutamos shellsort
          shellsort(words, wordsSize);

          // Enviamos a salida estandar
          ordenamientoImprimirSalidaEstandar(words, wordsSize);

          free(words);
          free(text);
          break;
        }

      default:
        opterr = 1;
        break;
    }
  }

  // Manejo de casos en que no se ingresan argumentos validos
  if(opterr || argc <= 1)
  {
    fprintf (stderr, "Los argumentos no son validos.\n");
    printf("Tipee 'tp0 -h' para ver el modo de uso.\n");
  }


  return 0;
}
```

A.2. *bubblesort.h*. Declaración de la librería Bubblesort

Código 7: *"bubblesort.h"*

```
/* ********************************************************************************
 * ********************************************************************************
 *
 * LIBRERIA BUBBLESORT
 *
 * ********************************************************************************
 * ********************************************************************************/



#ifndef BUBBLESORT_H
#define BUBBLESORT_H

```

```
14
15
16 // Funcion que aplica el algoritmo de ordenamiento Bubblesort para ordenar un
17 // arreglo de palabras.
18 // PRE: 'words' es un puntero a un arreglo de punteros a caracter; 'arraysize'
19 // es el tamanio de dicho arreglo.
20 // POST: el arreglo 'words' queda ordenado.
21 void bubblesort(char* words[], int arraysize);
22
23
24
25 #endif
```

A.3. *bubblesort.c*. Definición de la librería Bubblesort

Código 8: *"bubblesort.c"*

```
1  /* ****************************************************************************
2   * ****************************************************************************
3   *
4   * LIBRERIA BUBBLESORT
5   *
6   * ****************************************************************************
7   * ****************************************************************************/
8
9
10 #include "bubblesort.h"
11 #include <stdbool.h>
12 #include <string.h>
13
14
15
16 // Funcion que aplica el algoritmo de ordenamiento Bubblesort para ordenar un
17 // arreglo de palabras.
18 // PRE: 'words' es un puntero a un arreglo de punteros a caracter; 'arraysize'
19 // es el tamanio de dicho arreglo.
20 // POST: el arreglo 'words' queda ordenado.
21 void bubblesort(char* words[], int arraysize)
22 {
23   // Variables de procesamiento
24   bool huboIntercambio;
25   int i;
26   int n = arraysize;
27   char* sAux;
28
29   // Recorremos el arreglo haciendo intercambios hasta que ya no se registre
30   // ningun cambio realizado.
31   do
32   {
33     huboIntercambio = false;
34
35     for(i = 1; i < n; i++)
36     {
37       // Si el de indice menor es mayor que el de indice superior, los
38       // intercambiamos
39       if(strcasecmp(words[i-1], words[i]) > 0)
40       {
41         sAux = words[i-1];
42         words[i-1] = words[i];
43         words[i] = sAux;
44
45         // Cambiamos el flag para registrar que hubo un cambio
46         huboIntercambio = true;
47       }
48     }
49
50     // Como el elemento del indice superior se encuentra ya ordenado una
51     // vez finalizada la pasada, se reduce en uno la cantidad de indices
52     // a iterar en la proxima pasada.
53     n -= 1;
54
```

```
  } while(huboIntercambio);
}
```

A.4. *shellsort.h.* Declaración de la librería Shellsort

Código 9: *"shellsort.h"*

```
/* ****************************************************************************
 * ****************************************************************************
 *
 * LIBRERIA SHELLSORT
 *
 * ****************************************************************************
 * ****************************************************************************/



#ifndef SHELLSORT_H
#define SHELLSORT_H



// Funcion que aplica el algoritmo de ordenamiento Shellsort para ordenar un
// arreglo de palabras.
// PRE: 'words' es un puntero a un arreglo de punteros a caracter; 'arraysize'
// es el tamanio de dicho arreglo.
// POST: el arreglo 'words' queda ordenado.
void shellsort(char* words[], int arraysize);



#endif
```

A.5. *shellsort.c.* Definición de la librería Shellsort

Código 10: *"shellsort.c"*

```
/* ****************************************************************************
 * ****************************************************************************
 *
 * LIBRERIA SHELLSORT
 *
 * ****************************************************************************
 * ****************************************************************************/


#include "shellsort.h"
#include <string.h>



// Funcion que aplica el algoritmo de ordenamiento Shellsort para ordenar un
// arreglo de palabras.
// PRE: 'words' es un puntero a un arreglo de punteros a caracter; 'arraysize'
// es el tamanio de dicho arreglo.
// POST: el arreglo 'words' queda ordenado.
void shellsort(char* words[], int arraysize)
{
  int intervalo, k, j, i;
  char* sAux;

  // Tomamos como intervalo el modulo de la mitad del arreglo
  intervalo = arraysize / 2;

```

```
  // Iteramos hasta que el intervalo sea nulo
  while(intervalo > 0)
  {
    for(i = intervalo-1; i < arraysize; i++)
    {
      j = i - intervalo;

      // Procesamos hasta caer en un indice fuera de rango del arreglo
      while(j >= 0)
      {
        k = j + intervalo;

        // Si el de indice menor es mayor que el de indice superior, los
        // intercambiamos
        if(strcasecmp(words[j], words[k]) > 0)
        {
          sAux = words[j];
          words[j] = words[k];
          words[k] = sAux;
        }
        else
          j = 0;

        j -= intervalo;
      }
    }

    // Tomamos como nuevo intervalo el modulo de la mitad del intervalo
    // anterior
    intervalo = intervalo / 2;
  }
}
```

A.6. *fileloader.h*. Declaración de la librería File Loader

Código 11: *"fileloader.h"*

```
/* ****************************************************************************
 * ****************************************************************************
 *
 * LIBRERIA DE CARGA DE ARCHIVO DE PALABRAS
 *
 * ****************************************************************************
 * ****************************************************************************/



#ifndef FILELOADER_H
#define FILELOADER_H

#define INITIAL_BUFFER_SIZE 100

#include <stdbool.h>


// Funcion que carga un archivo de texto a memoria
// PRE: 'fileUrl' es la url completa al archivo a ordenar.
// POST: devuelve la lista de palabras cargadas en memoria
char* file_loader(const char * fileUrl);

// Funcion que separa las palabras de un texto en unidades.
// PRE: 'text' es una cadena que contiene el texto; 'result' es un puntero en donde se
// insertan las palabras del texto.
// POST: en 'result' se almacenaron las palabras.
int to_words(char * text, char *** result);

// Funcion que verifica si un caracter es un fin de palabra o un separador.
// PRE: 'letter' es el caracter a procesar.
// POST: devuelve true si el caracter es un fin de palabra o separador, o
// false en su defecto.
```

```
bool isEndOfWord(char letter);

#endif
```

A.7. *fileloader.c*. Definición de la librería File Loader

Código 12: *"fileloader.c"*

```
/* ****************************************************************************
 * ****************************************************************************
 *
 * LIBRERIA DE CARGA DE ARCHIVO DE PALABRAS
 *
 * ****************************************************************************
 * ****************************************************************************/


#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "fileloader.h"




// Funcion que carga un archivo de texto a memoria
// PRE: 'fileUrl' es la url completa al archivo a ordenar.
// POST: devuelve la lista de palabras cargadas en memoria
char * file_loader(const char * fileUrl)
{
   FILE * file;
   long fileSize;
   char * buffer;
   size_t result;

   file = fopen (fileUrl, "r");

   if (file==NULL)
   {
    //Fail to load file
    fputs ("The file can not be open or not exist \n", stderr);
    return NULL;
   }

   //Get file size
   fseek (file , 0 , SEEK_END);
   fileSize = ftell (file);
   rewind (file);

   //Allocate memory for the whole file
   buffer = (char *) malloc ((sizeof(char) * fileSize) + 1);

   if (buffer == NULL)
   {
    fputs ("There is no enoght memory to allocate the file in memory \n", stderr);
    return NULL;
  }

   //Copy the file to the buffer
   result = fread (buffer, 1, fileSize, file);
         buffer[(sizeof(char) * fileSize)] = '\0';

   if (result != fileSize)
   {
    fputs ("Fail to read the whole file \n", stderr);
    return NULL;
   }

   /* the whole file is now loaded in the memory buffer. */

```

```
   fclose (file);

   return buffer;
}


// Funcion que separa las palabras de un texto en unidades.
// PRE: 'text' es una cadena que contiene el texto; 'result' es un puntero en donde se
// insertan las palabras del texto.
// POST: en 'result' se almacenaron las palabras.
int to_words(char * text, char *** result)
{
  char ** words;
  char ** reallocatedWords = NULL;
  bool isNewWord = true;
  size_t textSize = strlen(text);
  int wordsIndex = 0;
  size_t bufferSize = INITIAL_BUFFER_SIZE * sizeof(char*);
  size_t spaceForWords = INITIAL_BUFFER_SIZE;

  //initialy allocate memory for 100 words
  words = (char **) malloc ( INITIAL_BUFFER_SIZE * sizeof(char*) );

  if(words == NULL)
  {
    fputs ("Not enough memory to allocate the words \n", stderr);
    return -1;
  }

  for (int i = 0; i < textSize; i++)
  {
    if ( !isEndOfWord (text[i]) )
    {
      if(isNewWord)
      {
        words[wordsIndex] = &text[i];
        isNewWord = false;

        if(wordsIndex >= spaceForWords-1)
        {
          //Reallocate for more space and copy the old array to the new one
          bufferSize *= 2;
          spaceForWords *= 2;
          reallocatedWords = realloc(words, bufferSize);

          if(reallocatedWords != NULL)
          {
            words = reallocatedWords;
            reallocatedWords = NULL;
          }
          else
          {
            fputs ("Not enough memory to allocate the words \n", stderr);
            return -1;
          }
        }

        wordsIndex++;
      }
    }
    else
    {
      text[i] = '\0';
      isNewWord = true;
    }
  }

  *result = words;

  return wordsIndex;
}


// Funcion que verifica si un caracter es un fin de palabra o un separador.
// PRE: 'letter' es el caracter a procesar.
// POST: devuelve true si el caracter es un fin de palabra o separador, o
```

```
// false en su defecto.
bool isEndOfWord(char letter)
{
  //A-Z es de 65 a 90
  //a-z es de 97 a 122
  //0-9 es de 48 a 57
  return !( (letter > 64 && letter < 91) || (letter > 96 && letter < 123) || (letter > 47 && letter < 58));
}
```

B. Implementación completa de Shellsort en assembly MIPS

B.1. *tp0.c*. Implementación del main del programa

Código 13: *"tp0.c"*

```
/* ******************************************************************************
 * PROGRAMA DE ORDENAMIENTO DE PALABRAS
 * ******************************************************************************
 *
 * Facultad de Ingenieria - UBA
 * 66.20 Organizacion de computadoras
 * Trabajo Practico: Conjunto de instrucciones MIPS
 *
 * ALUMNOS:
 *
 * Facundo Rossi (86707) - frossi85@gmail.com
 * Federico Colangelo (89869) - federico.colangelo@semperti.com
 * Federico Martin Rossi (92086) - federicomrossi@gmail.com
 *
 * ******************************************************************************
 *
 * Programa que permite aplicar metodos de ordenamiento a un texto o textos
 * especificados por el usuario. Estos textos pueden ser ingresados a traves de
 * la entrada estandar o bien a traves de la especificacion de los nombres de
 * archivo que se desean procesar. Cabe resaltar que al especificar varios
 * nombres de archivos, los metodos seran aplicados al conjunto total de los
 * textos contenidos en estos. Es decir, el resultado del ordenamiento sera
 * unico ya que contendra la mezcla de palabras de todos los archivos.
 * Los metodos de ordenamiento soportados por la presente version del programa
 * son:
 *
 *    - Shellsort
 *
 */

include <stdio.h>
#include <ctype.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>

#include "fileloader.h"

extern int shellsort(char **, size_t);

int main (int argc, char** argv)
{
  char* texto;
  char** words;
  size_t size;
  int i;

  texto=file_loader((charargv[1]);
  size=to_words(texto,&words);

  shellsort(words, size);
  for(i=0;i<size;i++)
  {
    puts(words[i]);
```

```
  }

  return 0;
}
```

B.2. *shellsort.S*. Implementación del algoritmo Shellsort

Código 14: *"shellsort.S"*

```
#ifndef USE_MIPS_ASSEMBLY
#define USE_MIPS_ASSEMBLY
#include <mips/regdef.h>
#include <sys/syscall.h>

.text
.align 2
.globl shellsort
.extern swap
.extern compare


shellsort:  subu sp,sp,24   # creo stack frame, es de 24 porque necesito guardar ra
    sw ra,16(sp)
    sw $fp,12(sp)
    sw gp,8(sp)
    move $fp,sp

    #Guardo los parametros recibidos en fp
    sw a0,24($fp)   # words = char **
    sw a1,28($fp)     # size

    #Guardo los parametros cargados en fp a variables temporales
    lw s6, 24($fp)    #en 24 esta el words
    lw s7, 28($fp)    #en 28 esta el size

    #Iniciaizo variables locales
    srl s0, s7, 1   # intervalo = size/2
    li s1, 0    # k
    li s2, 0    # j
    li s3, 0    # i
    li s4, 0    # sAux


    #add v0, s0, zero
    #j fin

    #while(intervalo > 0)
loop1:    bgt s0, zero, initloop2 #branch grater than, salta si s0 es mayor a 0
    j fin

    #for(i = intervalo-1; i < arraysize; i++)
initloop2:  addi s3, s0, -1   # i = interlvalo - 1
loop2:    bge s3, s7, endloop2  # si i >= size fin de for, bge=branch greater than or equal
    sub s2, s3, s0    #j = i - intervalo;


    #while(j >= 0)
loop3:    bge s2, zero, loop3task
    addi s3, s3, 1
    j loop2




loop3task:  add s1, s2, s0    #k = j + intervalo;

    #Para comparar cadenas, la forma mas sencilla es iterar sobre cada caracter de cada cadena, y restarlos. ↩
        Si el resultado es 0, son
#iguales. Si no, a continuacion, si el resultado es> 0, entonces la primera cadena es mayor de que la otra ↩
     cadena, de lo contrario la segunda cadena es inferior y #habria que intercambiarlas. Si te quedas sin ↩
```

```
    alguna de las cadenas antes que el otro, y son iguales hasta llegar a ese punto, la cadena mas corta es ↩
    menor que el que #mas.
    #CARGO LAS CADENAS A COMPARAR
    #words[j]
    ##Guardo en variables auxiliares las direcciones de memoria a intercambiar
    #words = s6
    #Guardo i y j por q las voy a modificar y no las quiero perder
    sw s1, 0($fp)
    sw s2, 4($fp)



    sll s1, s1, 2
    sll s2, s2, 2
    addu s1, s1, s6
    addu s2, s2, s6

    #Cargo los strings a comparar en ax que luego compare utiliza para trabajar
    lw a0, 0(s1)    #OPTIMIZAR CODIGO PARA ISAR MENOS ti
    lw a1, 0(s2)
    jal compare   #llamo a compare

    lw s1, 0($fp)   #Cargo el valor de k en stack
    lw s2, 4($fp)   #Cargo el valor de j en stack

    bltz v0, swapping #swap si compare > 0
    add s2, zero, zero  #j=0
    j endif

    # poner el valor a utilizar en los registros correctos y llamar a swap
swapping: add a0, s6,0
    add a1, s1, 0
    add a2, s2, 0
    jal swap

endif:    sub s2, s2, s0    #j = j - intervalo;
    j loop3


endloop2: srl s0, s0, 1   #intervalo = intervalo / 2;
    j loop1



    ##Comienzo retorno
fin:    move sp, $fp
    lw gp, 8(sp)
    lw $fp, 12(sp)
    lw ra, 16(sp)
    addiu sp, sp, 24
    jr ra     #ra = return address es la direccion a la q quiero volver luego de llamar la funcion

#endif
```

B.3. *compare.S*. Implementación del comparador de palabras

Código 15: *"compare.S"*

```

#ifndef USE_MIPS_ASSEMBLY
#define USE_MIPS_ASSEMBLY
#include <mips/regdef.h>
#include <sys/syscall.h>

.text
.align 2
.globl compare

# t1 = posicion caracter actual en cadena 1
# t2 = posicion caracter actual en cadena 2
```

```
13
14
15  compare:
16      subu sp,sp,16   # creo stack frame
17      sw $fp,12(sp)
18      sw gp,8(sp)
19      move $fp,sp
20
21      add t0, zero, zero  # result =0
22      add t1, zero, a0
23      add t2, zero, a1
24
25  nextchar:
26      #Cargo los elementos a comparar en t4 y t5
27      lb t3, 0(t1)
28      lb t4, 0(t2)
29
30      #Voy a pasar todas las mayusculas a minusculas para que compare sea case insenstive como en C
31
32      addi t5, t3, -91
33      bgtz t5, aMinusculas2
34      addi t5, t3, -64
35      bltz t5, aMinusculas2
36      addi t3, t3, 32
37
38  aMinusculas2: addi t5, t4, -91
39      bgtz t5, comparacion
40      addi t5, t4, -64
41      bltz t5, comparacion
42      addi t4, t4, 32
43
44
45      #Comparo caracter para ver si es final del string
46  comparacion:  beqz t3, fincadena1
47      beqz t4, fincadena2
48
49  continue: sub t0, t3, t4    # cadena1[i] - cadena2[i]
50      add t1, t1, 1   #NO VA sll x,y,2: por q char es un byte
51      add t2, t2, 1
52
53      beqz t0, nextchar
54      j fin
55
56
57  fincadena1: beqz t4, fin
58      add t0, zero, -1
59      j fin
60
61  fincadena2: add t0, zero, 1
62      j fin
63
64  fin:    ##Comienzo retorno
65      addi v0, t0, 0
66
67      move sp, $fp
68      lw gp, 8(sp)
69      lw $fp, 12(sp)
70      addiu sp, sp, 16
71      jr ra     #ra = return address es la direccion a la q quiero volver luego de llamar la funcion
72
73  #endif
```

B.4. *swap.S*. Implementación del comparador de palabras

Código 16: *"swap.S"*

```
1  #ifndef USE_MIPS_ASSEMBLY
2  #define USE_MIPS_ASSEMBLY
3  #include <mips/regdef.h>
4  #include <sys/syscall.h>
```

```
5
6  .text
7  .align 2
8  .globl swap
9
10
11
12 swap:
13     subu sp,sp,16   # creo stack frame
14     sw $fp,12(sp)
15     sw gp,8(sp)
16     move $fp,sp
17
18     #Guardo los parametros recibidos en fp
19     sw a0,16($fp)   # guardo los primeros 4 parametros con la llamo a checksum
20     sw a1,20($fp)     # 20-16 = 4Bytes=32bits
21     sw a2,24($fp)
22
23     ##Guardo en variables auxiliares las direcciones de memoria a intercambiar
24     lw t0, 16($fp)    #en 16 esta el words
25     lw t1, 20($fp)    #en 20 esta el i
26     lw t2, 24($fp)    #en 20 esta el j
27
28     ##Realizo el intercambio
29     #t3 = direccion de fp+(i*4)
30     #t4 = direccion de fp+(j+4)
31     sll t1, t1, 2
32     sll t2, t2, 2
33     addu t1, t1, t0
34     addu t2, t2, t0
35
36
37     #Cargo los elementos a intercambiar en t3 y t4
38     lw t3, 0(t1)
39     lw t4, 0(t2)
40
41     #Guardo los elementos de manera intercambiada
42     sw t4, 0(t1)
43     sw t3, 0(t2)
44
45
46     lw v0, 0(t2)
47
48     ##Comienzo retorno
49     move sp, $fp
50     lw gp, 8(sp)
51     lw $fp, 12(sp)
52     addiu sp, sp, 16
53     jr ra     #ra = return address es la direccion a la q quiero volver luego de llamar la funcion
54
55 #endif
```