Organización de Computadoras

Segundo Cuatrimestre 2017

Trabajo Práctico 0

Integrante	Padrón	Correo electrónico
Rodrigo De Rosa	97799	rodrigoderosa@outlook.com
Marcos Schapira	97934	schapiramarcos@gmail.com
Facundo Guerrero	97981	facundoiguerrero@gmail.com

Índice

1. Diseño e Implementación **1**
1.1. Estructura del problema . . . 1
1.2. Entorno . . . 1
1.3. Complicaciones . . . 1
1.4. Desarrollo . . . 1
1.5. Manejo de errores . . . 1
1.5.1. Valores devueltos por la función `main` . . . 2
1.6. Documentación . . . 2

2. Ejecución **4**
2.1. Instrucciones para la compilación . . . 4
2.2. Instrucciones para la ejecución . . . 4
2.3. Pruebas . . . 4
2.3.1. Prueba 1 . . . 5
2.3.2. Prueba 2 . . . 5
2.3.3. Prueba 3 . . . 5
2.3.4. Prueba 4 . . . 5
2.3.5. Prueba 5 . . . 6
2.3.6. Prueba 6 . . . 6
2.3.7. Prueba 7 . . . 6
2.3.8. Prueba 8 . . . 6
2.3.9. Prueba 9 . . . 7
2.3.10. Prueba 10 . . . 7
2.3.11. Prueba 11 . . . 7
2.3.12. Prueba 12 . . . 8

3. Conclusiones **8**

1. Diseño e Implementación

En este trabajo práctico inicial, cuyo objetivo es el de familiarizarnos con el entorno de desarrollo que utilizaremos en el cuatrimestre, se implementa un programa que recibe una entrada de texto e identifica los palíndromos que se encuentran en ella.

1.1. Estructura del problema

La entrada de texto previamente mencionada es una cadena de caracteres *ASCII* sin ninguna restricción. Dentro de esta cadena son consideradas *palabras* aquellas que están compuestas por los caracteres:

- $a - z$
- $0 - 9$
- _ y $-$

Cualquier otro caracter *ASCII* es considerado un *espacio*. Es decir, indica el fin de una *palabra* y el comienzo de otra. Cabe destacar que una cadena con un sólo caracter es considerada *palabra*.

1.2. Entorno

El trabajo se realizó en una máquina virtual *NetBSD* (que simula tener un procesador *MIPS*) montada por el emulador *GXemul* en *Ubuntu 17.04*.

1.3. Complicaciones

La principal complicación que surgió en el desarrollo del trabajo fue el hecho de que algunas librerías que existen en *Ubuntu* no existen en *NetBSD* (particularmente *argp*), por lo que hubo que adaptarse a esta limitación y utilizar librerías que funcionaran en ambos.

Por último, un problema a resolver fue el de tener que enviar por `scp` todos los archivos que fueran modificados en *Ubuntu* hacia *NetBSD*. De todos modos este problema fue resuelto utilizando `sshfs` que permite utilizar la interfaz gráfica de *Ubuntu* para modificar un directorio en la máquina virtual.

1.4. Desarrollo

El programa fue implementado en lenguaje C con sus librerías estándar y se utilizó la librería `getopt` para facilitar el parseo de los flags.

En cuanto al problema en sí, la solución implementada consiste en buscar las previamente llamadas *palabras* dentro de una cadena de caracteres y verificar si invertidas son iguales a su versión original (esto indica que son palíndromos). Para hacer esto se utilizó una implementación de la función `strrev` que, si bien en las versiones más actuales de C viene en las librerías estándar, en *NetBSD* no está dentro de estas. A continuación se detalla la documentación explicita de las funciones implementadas.

1.5. Manejo de errores

A lo largo del desarrollo del programa se definen ciertos errores para manejar posibles fallas del programa y así lograr un funcionamiento controlado y acorde. Estas son:

- `ALLOC_ERROR`
  El error se puede dar al llamar a la función `malloc`. Junto a su mensaje específico se imprime a la vez el código generado por strerror en la anterior función.

  Mensaje:

  ```
  An error ocurred while allocating memory!
  ```

- `REALLOC_ERROR`
  El error se puede dar al llamar a la función `realloc`. Junto a su mensaje específico se imprime a la vez el código generado por strerror en la anterior función.

  Mensaje:

  ```
  An error ocurred while reallocating memory!
  ```

- INPUT_OPEN_ERROR
  El error se puede dar al llamar la función ***fopen****. Junto a su mensaje específico se imprime a la vez el código generado por strerror en la anterior función.*

  Mensaje:

  ```
  An error ocurred while opening input file!
  ```

- OUTPUT_OPEN_ERROR
  El error se puede dar al llamar la función ***fopen****. Junto a su mensaje específico se imprime a la vez el código generado por strerror en la anterior función.*

  Mensaje:

  ```
  An error ocurred while opening output file!
  ```

- RESULT_WRITING_ERROR
  El error se puede dar al llamar la función ***fprintf*** *si no se logró escribir todo el mensaje o si algo falló. Junto a su mensaje específico se imprime a la vez el código generado por strerror en la anterior función.*

  Mensaje:

  ```
  An error ocurred while writing the result!
  ```

- PALINDROME_ERROR_MESSAGE
  El error se puede dar al llamar a la función interna ***get_palindromes****. Esta devuelve* ***NULL*** *en caso de fallar (junto con su adecuado mensaje, explicado a continuación en el informe).*

  Mensaje:

  ```
  An error ocurred while checking for palindromes!
  ```

1.5.1. Valores devueltos por la función `main`

Los siguientes códigos son mensajes devueltos por la función `main` al utilizar las funciones internas del programa(documentadas en la próxima sección del informe). Algunos de estos valores, en especial `FAIL` y `SUCCESS` son utilizados en otras funciones como valores booleanos False y True respectivamente.

- `SUCCESS` valor 0
  valor booleano de éxito.

- `FAIL` valor 1
  valor booleano de falla.

- `WRITING_ERROR` valor 2
  ocurre cuando la función ***write_result*** *devuelve* ***FAIL****.*

- `PALINDROME_ERROR` valor 3
  ocurre cuando la función ***get_palindromes*** *falla. Previamente se imprime el error* ***PALINDROME_ERROR_MESSAGE***

- `BAD_ARGUMENTS` valor 4
  ocurre cuando la función ***process_params*** *devuelve este mismo codigo al no poder procesar los parámetros correctamente.*

- `BAD_INPUT_PATH` valor 5
  ocurre cuando la función ***open_input*** *devuelve* ***FAIL****.*

- `BAD_OUTPUT_PATH` valor 6
  ocurre cuando la función ***open_output*** *devuelve* ***FAIL****.*

- `READING_ERROR` valor 7
  ocurre cuando la función ***read_input*** *devuelve* ***FAIL o NULL****.*

1.6. Documentación

Las siguientes funciones fueron implementadas con el objetivo de encontrar una solución al problema en cuestión:

- `char* read_input(FILE* fp, size_t size)`
  Lee el string entrante y lo devuelve.

  Parámetros:

  `fp`: File Pointer de input

`size`: Tamaño inicial del arreglo

Return:

El string leído o NULL en caso de fallar (imprimiendo el el error ocurrido)

Errores Posibles:

`REALLOC_ERROR`

`ALLOC_ERROR`

- `FILE* open_input(char* path)`
  Abre el input_file y se devuelve su fp. Si el path es `NULL`, se utiliza `DEFAULT_INPUT` siendo en este caso stdin.

  Parámetros:

  `path`: Dirección del archivo a abrir

  Return:

  File Pointer de input o `DEFAULT_INPUT` en caso de no especificar un path.

  Errores Posibles:

  `INPUT_OPEN_ERROR`

- `FILE* open_output(char* path)`
  Abre el output_file y se devuelve su fp. Si el path es `NULL` se utiliza `DEFAULT_OUTPUT` siendo en este caso stdout.

  Parámetros:

  `path`: Dirección del archivo a abrir

  Return:

  File Pointer de output o `DEFAULT_OUTPUT` en caso de no especificar un path.

  Errores Posibles:

  `OUTPUT_OPEN_ERROR`

- `int write_result(FILE* output_fp, char* result)`
  Escribe los palíndromos en el archivo indicado.

  Parámetros:

  `output_fp`: File Pointer del archivo a escribir

  `result`: String a escribir en el archivo

  Return:

  `SUCCESS o FAIL`

  Errores Posibles:

  `RESULT_WRITING_ERROR`

- `void close_files(FILE* fp1, FILE* fp2)`
  Cierra los dos archivos recibidos.

  Parámetros:

  `fp2`: File Pointer de archivo a cerrar

  `fp1`: File Pointer de archivo a cerrar

- `char* strrev(char* str)`
  Invierte la cadena recibida.

  Parámetros:

  `str`: Cadena de caracteres a invertir

  Return:

  Cadena recibida, en orden inverso

- `char* get_palindromes(char* string)`
  Devuelve un arreglo listo para escribir en un archivo que contiene en cada linea un palíndromo del string recibido.

  Parámetros:

`string`: Cadena a analizar en busca de palíndromos

Return:

Cadena que contiene solo palíndromos, con formato de uno por linea. En caso de error devuelve `NULL`

Errores Posibles:

`REALLOC_ERROR`

`ALLOC_ERROR`

- `bool is_palindrome(char* string)`
  Verifica si una cadena es un palíndromo o no.

  Parámetros:

  `string`: Cadena a analizar

  Return:

  Booleano

  Errores Posibles:

  `REALLOC_ERROR`

  `ALLOC_ERROR`

- `void print_help()`
  Imprime por consola información de los comandos y sobre el uso del programa.

- `void print_version()`
  Imprime por consola la version del programa y los integrantes del grupo.

- `int process_params(int argc, char** argv, char** input_file, char** output_file)`
  Procesa los parámetros de entrada del programa y almacena los paths correspondientes en los parámetros de la función.

  Parámetros:

  `argc`: Cantidad de argumentos del programa

  `argv`: Vector de argumentos del programa

  `input_file`: Puntero al string que contiene el path del input

  `output_file`: Puntero al string que contiene el path del output

  Return:

  `SUCCESS o BAD_ARGUMENTS` , en el segundo caso este valor es verificado y manejado en la función `main`.

2. Ejecución

2.1. Instrucciones para la compilación

Para compilar el programa se debe abrir una consola en el directorio donde se encuentra el archivo fuente (`tp.c`) y correr el comando: `gcc -Wall tp.c [-o OUTPUT]`.

2.2. Instrucciones para la ejecución

Suponiendo que nuestro archivo ejecutable fuera `tp0`, los comandos de consola para ejecutarlo son:

- `./tp0 -h` para ver la ayuda.
- `./tp0 -v` para ver la versión.
- `./tp0 -i  /INPUT -o  /OUTPUT` para correr el programa con `INPUT` como archivo de entrada y `OUTPUT` como archivo de salida. Ambos son opcionales y son reemplazados por `stdin` y `stdout` respectivamente.

2.3. Pruebas

Para probar el correcto funcionamiento del programa se utilizo un set de prueba. A continuación se muestra la composición y resultados de las ejecuciones de dicho set. Ademas, notar que desde la prueba 1 hasta la prueba 9, la entrada es mediante un archivo, es decir que se provee el archivo mediante `-i test_inputN` (con N = número de prueba), y la salida por terminal. Por último, vale aclarar que las pruebas se realizaron tanto en `Ubuntu` como en `NetBSD`.

2.3.1. Prueba 1

Caso de prueba provisto por la cátedra. Vale aclarar que un carácter es considerado palíndromo.

Entrada:

```
Somos los primeros en completar el TP 0.
Ojo que la fecha de entrega del TP0 es el martes 12 de Septiembre.
```

Salida:

```
Somos
0
Ojo
```

2.3.2. Prueba 2

Este caso de prueba intenta demostrar el correcto funcionamiento de la detección de espacios. Como se puede ver, en la primera linea las palabras están separadas por el carácter espacio, pero en la segunda se linea se intenta demostrar que los caracteres no validos (ver sección 1.1) también funcionan como espacios. Ademas, vale notar que la palabra palíndroma se detecta sin importar mayúsculas o minúsculas.

Entrada:

```
MeNEm neUquEn 1a2d323d2a1 adke
pepe)nene/larral=dom-mod?a23_32a
```

Salida:

```
MeNEm
neUquEn
1a2d323d2a1
larral
dom-mod
a23_32a
```

2.3.3. Prueba 3

El objetivo de esta prueba es ver el funcionamiento de los caracteres validos que no sean letras ni números. Como se vio en la sección 1.1, y _ deben ser considerados como caracteres validos. Entonces esta prueba quiere demostrar el funcionamiento de dichos caracteres. Ademas, nuevamente notar que la detección de las palabras no es `sensitive`.

Entrada:

```
aD2eT_R_Te2Da/4004?CheVr
peep23***avion
neUqUeN&NarNran
```

Salida:

```
aD2eT_R_Te2Da
4004
daad
neUqUeN
NarNran
```

2.3.4. Prueba 4

Esta prueba tiene el objetivo de corroborar el caso borde donde la entrada es vaciá.

Entrada:

Salida:

2.3.5. Prueba 5

El objetivo de esta prueba es verificar el correcto funcionamiento de la detección un único carácter.

Entrada:

```
a
```

Salida:

```
a
```

2.3.6. Prueba 6

El objetivo de esta prueba es corroborar el correcto funcionamiento del programa cuando se alcanza el limite inicial de tamaño del string que contiene las palabras palíndromas. En esta prueba, se ingresa una palabra palíndroma de 128 caracteres que es la capacidad máxima inicial.
La entrada para las siguientes pruebas es continua. Se hace un salto de linea con fines ilustrativos.

Entrada:

```
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa-
-aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
```

Salida:

```
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa-
-aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
```

2.3.7. Prueba 7

Esta prueba intenta demostrar el funcionamiento adecuado del programa cuando se supera el tamaño inicial del string donde se guardan las palabras palíndromas. Aquí, se ingresa una palabra palíndroma de 129 caracteres, es decir que se supera en 1 carácter la capacidad máxima. De esta manera, se esta forzando al programa a aumentar la capacidad máxima de almacenado. En otras palabras, se intenta probar el correcto funcionamiento del caso borde de la memoria, donde el programa debe hacer un realloc. Notar que se presentan dos casos de prueba, uno donde una misma palabra supera el limite, y en segundo lugar con palabras distintas.
La entrada para las siguientes pruebas es continua. Se hace un salto de linea con fines ilustrativos.

Entrada 1:

```
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa-
-aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
```

Salida 1:

```
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa-
-aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
```

Entrada 2:

```
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa-
-aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa menem
```

Salida 2:

```
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa-
-aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
menem
```

2.3.8. Prueba 8

Esta prueba tiene como objetivo verificar el funcionamiento del programa cuando se alcanza el máximo inicial lectura. Entonces, se provee al programa una entrada de 256 caracteres.
La entrada para las siguientes pruebas es continua. Se hace un salto de linea con fines ilustrativos.

Entrada:

```
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa-
```

```
-aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa-
-aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa-
-aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
```

Salida:

```
     aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa-
-aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa-
-aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa-
-aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
```

2.3.9. Prueba 9

El objetivo de esta prueba es corroborar el correcto funcionamiento del programa cuando se supera el tamaño inicial de lectura. Con dicho motivo, se ingresa una entrada de 257 caracteres, superando en 1 carácter la capacidad máxima. Al igual que la prueba 7, se esta probando el realloc pero en este caso para la lectura. **La entrada para las siguientes pruebas es continua. Se hace un salto de linea con fines ilustrativos.**

Entrada:

```
     aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa-
-aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa-
-aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa-
-aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
```

Salida:

```
     aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa-
-aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa-
-aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa-
-aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
```

2.3.10. Prueba 10

Se repite la prueba 1, esta vez con el objetivo de probar la entrada-salida vía archivos. La entrada se pasa por parámetro como `-i test_input1` y se pasa el archivo de salida `-i test_output10`. La salida mostrada, es lo que contiene el archivo de salida anteriormente mencionado.

Entrada:

```
Somos los primeros en completar el TP 0.
Ojo que la fecha de entrega del TP0 es el martes 12 de Septiembre.
```

Salida:

```
Somos
0
Ojo
```

2.3.11. Prueba 11

Se repite la prueba 2, esta vez con el objetivo de probar la entrada-salida vía terminal. Se correrá el programa sin ningún parámetro, y se hará el ingreso por entrada estándar (indicando su finalización con `Ctrl+D`) y se obtendrá la salida de la misma forma.

Entrada:

```
MeNEm neUquEn 1a2d323d2a1 adke
pepe)nene/larral=dom-mod?a23_32a
```

Salida:

```
MeNEm
neUquEn
1a2d323d2a1
larral
dom-mod
a23_32a
```

2.3.12. Prueba 12

Se repite la prueba 3, con el objetivo de verificar la entrada estándar y salida vía archivos. La salida se pasa por parámetros como `-i test_input12`. La salida mostrada a continuación, es lo que contiene el archivo de salida anteriormente mencionado.

Entrada:

```
aD2eT_R_Te2Da/4004?CheVr
peep23***avion daad
neUqUeN&NarNran
```

Salida:

```
aD2eT_R_Te2Da
4004
daad
neUqUeN
NarNran
```

3. Conclusiones

La principal conclusión que obtuvimos a partir del desarrollo de este trabajo práctico es que, al trabajar en *NetBSD* se debe ser más cuidadoso al programar que en el caso de, por ejemplo, *Ubuntu*. Con esto nos referimos a ciertas situaciones particulares en las que *Ubuntu* resuelve ciertos problemas de la programación que *NetBSD* no resuelve y resulta en un error.

Por ejemplo, en la línea 291 de `tp.c` se agrega un fin de linea que no es necesario en *Ubuntu*, pues lo 'resuelve solo', pero sí lo es en *NetBSD* pues en este último, el programa no encuentra en fin de linea e imprime caracteres de más.

Por esta misma razón, notamos que es recomendable trabajar constantemente utilizando la carpeta virtual de `sshfs` para poder compilar y ejecutar el programa en *NetBSD* en lugar de hacerlo en *Ubuntu*. De todas maneras, al tener que desarrollar en *Assembler*, no queda otra opción que hacer esto recién mencionado.