Práctica 2: Programación en C y acceso a ficheros mediante la biblioteca estándar

Índice

1 Objetivos

En esta práctica vamos a hacer varios ejercicios orientados a afianzar nuestro conocimiento sobre la programación de sistemas en C y el uso de su biblioteca estándar para operaciones básicas sobre cadenas de caracteres, entrada salida y ficheros.

Se aconseja al alumno que cree un directorio para la práctica con un subdirectorio por ejercicio. En las instrucciones se asume que el ejercicio N se hace en un subdirectorio llamado ejercicio*N* dentro del directorio común para la práctica.

El archivo ficheros_p2.tar.gz contiene una serie de ficheros que pueden usarse como punto de partida para el desarrollo de los ejercicios de esta práctica, así como unos *makefiles* que pueden ser usados para la compilación de los distintos proyectos.

2 Ejercicios

Ejercicio 1: Manejo básico de ficheros con librería estándar

Analiza el código del programa `show_file.c`, que lee byte a byte el contenido de un fichero, cuyo nombre se pasa como parámetro, y lo muestra por pantalla usando funciones de la biblioteca estándar de "C". Compila y comprueba el funcionamiento correcto del programa. Después modifica el código reemplazando el uso de `getc()` por el de la función `fread()` y el uso de `putc()` por el de la función `fwrite()`. Consulta las páginas de manual correspondientes.

```c
#include <stdio.h>
#include <stdlib.h>

int main(int argc, char* argv[]) {
    FILE* file=NULL;
    int c,ret;
```

```
    if (argc!=2) {
        fprintf(stderr,"Usage: %s <file_name>\n",argv[0]);
        exit(1);
    }

    /* Open file */
    if ((file = fopen(argv[1], "r")) == NULL)
        err(2,"The input file %s could not be opened",argv[1]);

    /* Read file byte by byte */
    while ((c = getc(file)) != EOF) {
        /* Print byte to stdout */
        ret=putc((unsigned char) c, stdout);

        if (ret==EOF){
            fclose(file);
            err(3,"putc() failed!!");
        }
    }

    fclose(file);
    return 0;
}
```

Ejercicio 2: Escritura y lectura de cadenas de caracteres en ficheros

Desarrollar dos programas sencillos `write_strings.c` y `read_strings.c` que permitan respectivamente escribir y leer de un fichero un conjunto de cadenas de caracteres de longitud variable terminadas por `'\0'`. Dicho caracter terminador deberá almacenarse en el fichero con el resto de caracteres de cada cadena. Para el desarrollo de los dos programas se utilizarán las siguientes funciones de la biblioteca estándar: `fopen()`,`fclose()`, `fread()`, `fwrite()`, `fseek()` y `malloc()`

El programa `write-strings.c` aceptará como primer parámetro el nombre de un fichero de texto donde se escribirán los strings pasados a continuación a la línea de comandos (argumento 2, argumento 3, etc.). Si el fichero destino existe, el programa reescribirá su contenido.

El programa `read-strings.c` aceptará como parámetro el nombre del fichero de texto donde se almacenen las cadenas de caracteres terminadas en `'\0'`. Este programa leerá las cadenas y las imprimirá por pantalla separadas por un salto de línea, como se muestra en el siguiente ejemplo de ejecución:

```
## Write strings to file
usuarioso@debian:~/exercise2$ ./write_strings out London Paris Madrid Barcelona Berlin Lisbon

## Check whether file structure is correct (null-terminated strings)
usuarioso@debian:~/exercise2$ $ xxd out
00000000: 4c6f 6e64 6f6e 0050 6172 6973 004d 6164  London.Paris.Mad
00000010: 7269 6400 4261 7263 656c 6f6e 6100 4265  rid.Barcelona.Be
00000020: 726c 696e 004c 6973 626f 6e00            rlin.Lisbon.

## Read strings from file
usuarioso@debian:~/exercise2$ ./read_strings out
London
Paris
Madrid
Barcelona
Berlin
Lisbon
```

Por simplicidad para la implementación del programa `read-strings.c`, se ha de desarrollar una función auxiliar

`char* loadstr(FILE* input)`. Esta función lee una cadena de caracteres terminada en `'\0'` del fichero cuyo descriptor se pasa como parámetro, reservando dinámicamente la cantidad de memoria adecuada para la cadena leída y retornando dicha cadena. La función tendrá que averiguar primero el número de caracteres de la cadena que comienza a partir de la ubicación actual del puntero de posición del fichero, leyendo caracter a caracter. Una vez detectado el caracter terminador, restaurará el indicador de posición del fichero (moviéndolo hacia atrás) y, finalmente realizará una lectura de la cadena completa.

Ejercicio 3: Gestión de ficheros de texto y binarios con la biblioteca estándar de C

En este ejercicio de la práctica se desarrollará un programa C más elaborado que lea y escriba de ficheros regulares tanto de texto, como en formato binario. Para su implementación, los estudiantes deberán utilizar al menos las siguientes funciones de la biblioteca estándar de C: `getopt`, `printf`, `fopen`, `fclose`, `fgets`, `fscanf`, `feof`, `fprintf`, `fread`, `fwrite` y `strsep`. Se deben consultar las páginas de manual de estas funciones en caso de duda sobre su comportamiento.

El ejercicio consta de 3 partes (más extensiones opcionales), donde se irán implementando gradualmente distintas características del programa:

Parte A

Desarrollar un programa `student-records.c` que lea un fichero de texto con información de distintos estudiantes, e imprima por la salida estándar la información leida en un formato amigable. El fichero de texto de estudiantes almacena un registro de 4 campos por estudiante (identificador numérico único, NIF, nombre, y apellido). Para simplificar el *parsing* del fichero, su primera línea contiene el número de registros de estudiantes, y a continuación se encuentran los distintos registros de estudiante, uno por línea, con campos separados por ":", como en el siguiente ejemplo:

```
4
27:67659034X:Chris:Rock
34:78675903J:Antonio:Banderas
3:58943056J:Santiago:Segura
4:6345239G:Penelope:Cruz
```

El fichero de ejemplo almacena 4 registros de estudiantes, donde la información del primer estudiante es la siguiente:

- Identificador numérico: `27`
- NIF: `67659034X`
- Nombre: `Chris`
- Apellido: `Rock`

Para leer un fichero de texto de estudiantes e imprimir su contenido en formato amigable, el programa `student-records` deberá invocarse especificando las opciones `-i` (*input*) y `-p` (*print*) simultáneamente en la línea de comando, donde la opción `-i` acepta un parámetro indicando la ruta del fichero de texto. Así por ejemplo, asumiendo que existe un fichero *students-db.txt* en el directorio actual que contiene el texto de ejemplo mostrado anteriormente, la ejecución del programa producirá la siguiente salida:

```
usuarioso@debian:~/exercise3$ ./student-records -i students-db.txt -p
[Entry #0]
        student_id=27
        NIF=67659034X
        first_name=Chris
        last_name=Rock
[Entry #1]
        student_id=34
        NIF=78675903J
        first_name=Antonio
        last_name=Banderas
```

```
[Entry #2]
        student_id=3
        NIF=58943056J
        first_name=Santiago
        last_name=Segura
[Entry #3]
        student_id=4
        NIF=6345239G
        first_name=Penelope
        last_name=Cruz
```

Por simplicidad en la implementación, cada registro se imprimirá por la salida estándar tan pronto como se procese la línea de texto del fichero de entrada correspondiente a dicho registro. De este modo no será necesario almacenar en memoria la información completa del fichero. Para representar en memoria de los distintos campos del registro, se recomienda el uso de la estructura `student_t`, definida en el fichero de cabecera `defs.h` proporcionado con la práctica.

Además de las opciones arriba mencionadas, se implementará una opción `-h` (*help*), que imprima el listado de opciones soportadas por el programa:

```
usuarioso@debian:~/exercise3$ ./student-records -h
Usage: ./student-records [ -h | -p | -i file ]
```

La salida generada por esta opción deberá modificarse a medida que se implementen opciones adicionales en el programa, correspondientes a los distintos apartados.

Nota importante: Para la implementación de este apartado se aconseja reutilizar código del programa `show-passwd.c`, suministrado con el Ejercicio 4 de la Práctica 1. En ese programa se realiza el *parsing* de un fichero de texto con un formato similar al que se emplea en este ejercicio (campos de distinto tipo separados por ":").

Parte B

Extender la funcionalidad del programa `student-records` implementando una nueva opción `-o`. Esta opción permitirá generar una representación binaria de los registros de estudiantes, y volcarla en un fichero de salida cuya ruta se pasará como parámetro a la opción `-o`. Al leer cada entrada del fichero de texto, el programa almacenará la información del estudiante en un registro representado mediante el siguiente tipo de datos:

```c
#define MAX_CHARS_NIF  9

typedef struct {
    int student_id;
    char NIF[MAX_CHARS_NIF+1];
    char* first_name;
    char* last_name;
} student_t;
```

El fichero binario a generar tendrá la siguiente estructura. Los primeros 4 bytes del fichero (int) almacenarán el número de registros de estudiantes. A continuación se escribirán los datos de cada registro de estudiantes, uno detrás del otro, almacenando por cada uno de ellos su ID de estudiante (entero de 4 bytes), su NIF, su nombre y apellido (en este orden). Para todas las cadenas de caracteres a escribir en el fichero se almacenará también el caracter terminador, lo cual es clave para poder leer el fichero a posteriori (parte C del ejercicio).

En el siguiente ejemplo de ejecución se hace uso del comando `xxd` para mostrar el contenido del fichero generado (nótese que este fichero no puede imprimirse satisfactoriamente con `cat`, al tratarse de un fichero binario):

```
## Check usage
usuarioso@debian:~/exercise3$ ./student-records -h
Usage: ./student-records [ -h | -p | -i file | -o <output_file> ]

## Generate binary file
usuarioso@debian:~/exercise3$ ./student-records -i students-db.txt -o students-db.bin
4 student records writen successfully to binary file students-db.bin

## Check file contents
usuarioso@debian:~/exercise3$ xxd students-db.bin
00000000: 0400 0000 1b00 0000 3637 3635 3930 3334  ........67659034
00000010: 5800 4368 7269 7300 526f 636b 0022 0000  X.Chris.Rock."..
00000020: 0037 3836 3735 3930 334a 0041 6e74 6f6e  .78675903J.Anton
00000030: 696f 0042 616e 6465 7261 7300 0300 0000  io.Banderas.....
00000040: 3538 3934 3330 3536 4a00 5361 6e74 6961  58943056J.Santia
00000050: 676f 0053 6567 7572 6100 0400 0000 3633  go.Segura.....63
00000060: 3435 3233 3947 0050 656e 656c 6f70 6500  45239G.Penelope.
00000070: 4372 757a 00                             Cruz.
```

Parte C

Implementar una nueva opción `-b` (*binary*) en el programa que permita imprimir el contenido de un fichero binario de estudiantes existente usando el mismo formato de salida que el especificado en la Parte A del ejercicio. Al indicar la opción `-b` (en lugar de `-p`) en la línea de comandos, el programa asumirá que el formato del fichero de entrada será binario en lugar de texto.

El siguiente ejemplo ilustra la funcionalidad que ha de implementarse en este apartado:

```
## Check usage
usuarioso@debian:~/exercise3$ ./student-records -h
Usage: ./student-records [ -h | -p | -i file | -o <output_file> | -b ]

## Dump contents of binary file
usuarioso@debian:~/exercise2$ ./student-records -i students-db.bin -b
[Entry #0]
        student_id=27
        NIF=67659034X
        first_name=Chris
        last_name=Rock
[Entry #1]
        student_id=34
        NIF=78675903J
        first_name=Antonio
        last_name=Banderas
[Entry #2]
        student_id=3
        NIF=58943056J
        first_name=Santiago
        last_name=Segura
[Entry #3]
        student_id=4
        NIF=6345239G
        first_name=Penelope
        last_name=Cruz
```

Pista: Se aconseja reutilizar para la implementación la función `loadstr()`, desarrollada en el ejercicio 2.

Partes opcionales

Opcional 1

En los apartados obligatorios de la práctica se asume que el programa procesa los registros leidos del fichero de entrada uno a uno, o bien escribiendolos en la salida estándar (opciones `-p` y `-b`) o volcando cada registro leído en el fichero de salida (opción `-o`) según se lee. En esta parte opcional se propone refactorizar el programa del código `student_records.c` de tal forma que los registros del fichero de entrada –ya esté en formato binario o texto– se lean todos de forma consecutiva y se almacenen en un vector de registros de tipo `student_t`. Para ello se han de definir 2 funciones auxiliares:

- `student_t* read_student_text_file(FILE* students, int* nr_entries)`

  Esta función lee toda la información de un fichero de texto de registros de estudiantes ya abierto, y devuelve tanto el número de registros en el fichero (parámetro de retorno `nr_entries`), como el array de registros de estudiantes (valor de retorno de la función). La memoria del array que se retorna debe reservarse con `malloc()` dentro de la propia función.

- `student_t* read_student_binary_file(FILE* students, int* nr_entries)`

  Igual que la función `read_student_text_file()` pero leyendo un fichero binario de registros de estudiantes

Para completar esta parte opcional, se ha de reescribir el resto de funciones implementadas para usar la nueva funcionalidad proporcionada por estas funciones. Así por ejemplo, la función que antes leía del fichero de texto de entrada, e imprimía uno a uno los registros en la salida estándar, debe ahora leer todos los registros de golpe usando `read_student_text_file()`, y a continuación imprimir los registros almacenados en el vector devuelto usando un bucle.

Opcional 2

Extender la funcionalidad del programa `student_records.c` con una nueva opción `-a` que permita añadir registros extra de estudiantes al final de un fichero existente, ya sea en formato binario o texto. Los nuevos registros a añadir se especificarán en modo texto en la línea de comando, con campos separados por `:`, y donde los registros se separarán por espacios. El programa inferirá automáticamente el formato del fichero existente en base a su extensión (".txt": texto; ".bin": binario).

Ejemplo de ejecución:

```
## Check usage
usuarioso@debian:~/exercise3$ ./student-records -h
Usage: ./student-records [ -h | -p | -i file | -o <output_file> | -b | -a ]

## Add two new records
usuarioso@debian:~/exercise3$ ./student-records -i students-db.txt -a \
> 23:43159076B:Michael:Jordan 30:34651129G:Stephen:Curry
2 records written succesfully to existing text file

## Check contents
usuarioso@debian:~/exercise3$ cat students-db.txt
6
27:67659034X:Chris:Rock
34:78675903J:Antonio:Banderas
3:58943056J:Santiago:Segura
4:6345239G:Penelope:Cruz
23:43159076B:Michael:Jordan
30:34651129G:Stephen:Curry
```

Nótese que en este caso la opción -a no acepta ningún argumento, sino que los registros se proporcionan como argumentos extra en la línea de comandos (sin opción asociada). Para procesar estos argumentos, ha de emplearse la variable global `optind` de `getopt()`, que al finalizar el procesamiento de opciones almacena el índice del primer argumento extra proporcionado. En el ejemplo de comando mostrado anteriormente, `optind` valdrá 4 al acabar el bucle de procesamiento

de opciones, ya que el primer argumento extra constituye el *token* número 4 de la línea de comandos. De este modo la expresión de C `&argv[optind]` puede emplearse para acceder al vector de argumentos extra, teniendo el siguiente contenido en el ejemplo: `{"23:43159076B:Michael:Jordan", "30:34651129G:Stephen:Curry", NULL}`