Programación I

Práctica #7 **Curso 2024-2025** **1º curso**

Ficheros. Argumentos por línea de Comandos

Introducción

En esta práctica, aprenderemos a condicionar el funcionamiento de nuestros programas permitiendo que, al ser invocados, se les puedan pasar diferentes argumentos en la línea de comandos.

Adicionalmente, introduciremos los conceptos necesarios para trabajar sobre ficheros de texto. Aprenderemos a declararlos, abrirlos, leer de ellos y escribir en ellos.

Finalmente, sentaremos la bases para realizar una entrada robusta de datos hacia nuestros programas.

Objetivos

Esta práctica tiene como objetivos fundamentales:

- Aprender a utilizar la línea de comando para pasar argumentos a un programa.
- Empezar a operar sobre ficheros de texto.

Planificación

Sesión 7 de laboratorio (3 horas):

1. Realización de los ejercicios 1 a 3.
2. Realización de los ejercicios 4 y 5.
3. Realización de los ejercicios 6 y 7.

Trabajo fuera del aula:

- Completar los códigos no finalizados en clase.

Paso de argumentos por línea de comandos

La función principal de nuestro programa tiene la siguiente declaración:

```
int main (int argc, char *argv[]);
```

Hasta ahora no hemos utilizado sus parámetros. Estos parámetros se utilizan para poder pasarle argumentos a nuestros programas cuando los ejecutamos, y tienen el siguiente significado:

- `argc`: Es el número de argumentos de entrada que se le han pasado al programa
- `argv`: Contiene los valores de los argumentos en forma de cadenas de caracteres (el primer argumento será `argv[0]`, el segundo `argv[1]`,...)

Ejercicio 1:

Escribe un programa (`ej1.c`) que imprima en pantalla todos sus argumentos (puedes usar el que se ha proporcionado en las transparencias de teoría). Pruébalo con varias combinaciones y número de argumentos.

```
usuario@maquina:~/pr07$ ./ej1
Número de argumentos: 1
Argumento 0: ./ej1

usuario@maquina:~/pr07$ ./ej1 uno dos tres
Número de argumentos: 4
Argumento 0: ./ej1
Argumento 1: uno
Argumento 2: dos
Argumento 3: tres
```

Ejercicio 2:

Copia el programa que escribiste en el ejercicio 4 de la Práctica #6:

```
usuario@maquina:~/pr07$ cp ../pr06/ej4.c ej2.c
```

Modifícalo para que tome la base y la altura del rectángulo por línea de comandos, en vez de solicitárselos por teclado al usuario.

```
usuario@maquina:~/pr07$ ./ej2  3.5  6.7
Altura:     6.70 cm
Base:       3.50 cm
Area:      23.45 cm x cm
Perimetro: 20.40 cm
```

Si el usuario no proporciona en la línea de comandos una base y una altura, el programa debe mostrar un mensaje de *uso* como el siguiente, y salir:

```
usuario@maquina:~/pr07$ ./ej2
Uso: ./ej2 base altura
```

Ficheros de texto

Para poder referirnos a un fichero es necesaria la declaración de una variable del tipo `FILE *` :

```
FILE *fichero;
```

Para trabajar con un fichero, debe estar abierto mediante la función `fopen()`:

```
FILE* fopen (const char* nombre_fich, const char* modo);
```

El modo `"r"` permite abrirlo en lectura. El modo `"w"`, en escritura.

Tras terminar de trabajar con un fichero, es necesario cerrarlo mediante la función `fclose()`:

```
int fclose (FILE *fichero);
```

Operaciones carácter a carácter

Disponemos de funciones para leer y escribir en un fichero, carácter a carácter:

```
int fgetc (FILE* fichero);
int fputc (char caracter, FILE* fichero);
```

Ejercicio 3:

Escribe un programa (`ej3.c`) que:

- reciba el nombre de un fichero como argumento por línea de comandos:

```
usuario@maquina:~/pr07$ ./ej3 elfichero.txt
```

- si el usuario no proporciona en la línea de comandos un nombre de fichero, muestre un mensaje de *uso* y termine:

```
usuario@maquina:~/pr07$ ./ej3
Uso: ./ej3 nombre_fichero
```

- si el fichero no existe, muestre un mensaje de error y termine:

```
usuario@maquina:~/pr07$ ./ej3 no_existe.txt
Fichero inexistente
```

- si el fichero existe:

  - lo lea **carácter a carácter**
  - lo escriba carácter a carácter en un nuevo fichero llamado: `"caracter.txt"`

Reflexiona:
- ¿Qué pasa si el fichero `"caracter.txt"` ya existía?

Operaciones cadena a cadena

Disponemos de funciones para leer y escribir en un fichero cadena a cadena:

```
char *fgets (char *cadena, int num, FILE* fichero);
int fputs (char *cadena, FILE* fichero);
```

Ejercicio 4:

Copia el programa que escribiste en el ejercicio 3:

```
usuario@maquina:~/pr07$ cp ej3.c ej4.c
```

Modifícalo para que, si el fichero existe:

- lo lea **línea por línea**
- lo escriba línea por línea en un nuevo fichero llamado: `"linea.txt"`

Operaciones de entrada/salida formateada

Disponemos de funciones para leer y escribir en un fichero mediante entrada/salida formateada:

```
int fscanf (FILE* fichero, const char *formato,…);
int fprintf (FILE* fichero, const char *formato,…);
```

Ejercicio 5:

Escribe un programa (`ej5.c`) que:

- de forma repetitiva:
  - le solicite al usuario un **entero,** una **cadena** de caracteres y un número **real**.
  - escriba esos datos en una sola línea en un nuevo fichero llamado: `"formato.txt"`, con el formato:

    `cadena entero real`

    (donde `real` es el número real con 2 decimales)
- termine cuando el usuario introduzca un entero negativo

Por ejemplo, si la entrada del usuario es:

```
Dame un entero: 345
Dame una cadena: Programacion
Dame un real: 2.343
Dame un entero: -4
```

El contenido del fichero será:

```
Programacion 345 2.34
```

La forma más habitual de realizar la entrada formateada de información desde un archivo de texto (o desde la entrada estándar) es hacer lo siguiente, de forma repetitiva:

- leer una línea completa, dejándola en una cadena de caracteres, mediante `fgets()`
- procesar el contenido de dicha línea con la función `sscanf()`

Ejercicio 6:

Escribe un programa (`ej6.c`) que:

- lea línea por línea el fichero `"formato.txt"` del ejercicio 5, mediante `fgets()`,
- procese el contenido de cada línea separando sus 3 datos, mediante `sscanf()`,
- muestre dicho contenido de forma separada.

Por ejemplo, si el fichero contiene:

```
Programacion 345 2.34
Arquitectura 1345 34.68
```

La salida del programa será :

```
Cadena: Programacion
Entero: 345
Real: 2.34

Cadena: Arquitectura
Entero: 1345
Real: 34.68
```

Pero, si el fichero no existe, o no se puede leer, la salida será:

```
No se puede acceder al fichero formato.txt
```

Entrada robusta de datos

El uso conjunto de las funciones `fgets()` y `fscanf()` nos permite detectar si la entrada proporcionada por el usuario (en caso de estar trabajando con `stdin`) o el fichero que se está leyendo respetan o no el formato de datos establecido.

Para ello, deberemos comprobar el valor de retorno de `sscanf()`.

Ejercicio 7:

Copia el programa que escribiste en el ejercicio 6:

```
usuario@maquina:~/pr07$ cp ej6.c ej7.c
```

Modifícalo para que, si alguna línea tiene un contenido que no respeta el formato especificado, se emita un mensaje de error. Por ejemplo, si el fichero contiene:

```
Programacion 345 2.34
Cprogramming 321 ABC

Java XYZ 4.56
Algoritmos 654 23.46 MNO
Arquitectura 1345 34.68
```

El 3º dato no es un número real

Línea vacía

Faltan datos

Sobran datos

La salida del programa será :

```
Cadena: Programacion
Entero: 345
Real: 2.34

Mal formato: Cprogramming 321 ABC
Mal formato:
Mal formato: Java XYZ 4.56
Mal formato: Algoritmos 654 23.46 MNO
Cadena: Arquitectura
Entero: 1345
Real: 34.68
```

Reflexiona: tras realizar este ejercicio, con lo que has aprendido, ¿serías capaz de proporcionar una solución **robusta** para el ejercicio 6 de la Práctica #4?

Resumen

Los principales resultados esperados de esta práctica son:

- Saber manejar los ficheros de texto.
- Saber cómo utilizar la línea de comando para pasar argumentos a un programa.

Como posibles líneas de trabajo adicional del alumno, se proponen las siguientes:

- Terminar los ejercicios no completados en clase.