Relación de prácticas de la asignatura
METODOLOGÍA DE LA PROGRAMACIÓN
Segundo Cuatrimestre
Curso 2023-2024

1º Grado en Informática

Práctica 3: Ficheros, recursividad y Argumentos en Línea de órdenes

Objetivos

- Practicar conceptos básicos sobre recursividad, ficheros de texto y ficheros binarios en C.
- Usar argumentos en línea de órdenes para que los programas puedan recibir argumentos en el momento de ser ejecutados.

Recomendaciones

- Pasa a tus funciones el nombre del fichero como parámetro y ábrelo y ciérralo dentro de la función.
- Organiza los ejercicios 4 y 7 en al menos tres ficheros *.c* (*main.c, ficheros.c, util.c*) y dos ficheros *.h* (*ficheros.h y util.h*).

Temporización

- 3 sesiones de prácticas:
  - Sesión 1: Ejercicios de recursividad
  - Sesión 2: Ejercicios de ficheros de texto
  - Sesión 3: Ejercicios de ficheros binarios
- Trabajo personal en casa: Terminar de hacer los ejercicios que no haya dado tiempo en clase

Ficheros de texto

1. Codifica un programa en C que, utilizando funciones, abra un fichero de **texto**, cuyo nombre se pedirá al usuario, y genere un nuevo fichero llamado *mayusculas-nombreFicheroDeEntrada.txt*, que tenga el mismo contenido que el fichero original pero en mayúsculas.

2. Codifica un programa en C que, utilizando funciones, cree un fichero **texto** con números enteros generados aleatoriamente en un intervalo especificado por el usuario. El programa le preguntará al usuario el nombre del fichero a crear, el número de elementos que contendrá el fichero y los extremos superior e inferior del intervalo.

3. Codifica un programa en C que, utilizando funciones, calcule la media de los elementos pares que hay en un fichero de texto generado por el ejercicio anterior. El nombre del fichero se pasará como argumento en la línea de órdenes.

4. Construye un programa que gestione mediante ficheros de **texto** el *stock* de libros de una librería. Para cada libro se almacenarán tres líneas en un fichero de texto (*stock*):
   - en la primera línea el *título*,
   - en la segunda línea el *autor,*
   - y en la tercera línea *el precio* y las *unidades* disponibles del libro.

   El programa contará con un **menú** que permitirá realizar las siguientes operaciones:
   - Comprobar la existencia de un determinado libro en el *stock* buscando por su título.
   - Introducir un nuevo libro en el *stock*.
   - Contar el número de libros (títulos) diferentes que hay en el *stock*. Considera que no puede haber títulos de libros repetidos en el fichero.
   - Listar los libros en el *stock* almacenándolos previamente en un vector dinámico con la siguiente estructura:

```
struct libro {
  char titulo[100];
  char autor[50];
  float precio;
  int unidades;
}
```

   - Vender *n* unidades de un libro buscándolo por su título. Si hay menos de *n* unidades en el *stock*, se venderán solo las unidades disponibles.
   - Borrar aquellos registros con 0 unidades disponibles.
   - Salir.

   El nombre del fichero se pasará como argumento en la línea de órdenes. Tienes un ejemplo de menú en Moodle.

Ficheros binarios

5. Codifica un programa en C que, utilizando funciones, cree un fichero **binario** con números enteros generados aleatoriamente en un intervalo especificado por el usuario. El programa guardará los números en un vector dinámico antes de volcarlo a disco. El programa recibirá 4 parámetros como argumentos en la línea de órdenes: nombre del fichero a crear, número de elementos que contendrá el fichero y los extremos superior e inferior del intervalo.

6. Codifica un programa en C que, utilizando funciones, lea números enteros desde un fichero binario generado en el ejercicio anterior, almacene sus valores en un vector dinámico y calcule la media de los números pares. El nombre del fichero se le preguntará al usuario.

7. Construye un programa que gestione, mediante ficheros **binarios**, el *stock* de libros de una librería. Cada libro se almacenará en una estructura *struct libro*, descrita en el ejercicio 4.

   El programa contará con un **menú** que permitirá realizar las siguientes operaciones:
   - Comprobar la existencia de un determinado libro buscando por su título.
   - Introducir un nuevo libro en el *stock*.
   - Contar el número de libros (títulos) diferentes que hay en el *stock*. Considera que no puede haber títulos de libros repetidos en el fichero.
   - Listar los libros en el *stock* almacenándolos previamente en un vector dinámico
   - Vender *n* unidades de un libro buscándolo por su título. Si hay menos de *n* unidades en el *stock*, se venderán solo las unidades disponibles.
   - Borrar (borrado físico) aquellos registros con 0 unidades disponibles.
   - Salir.

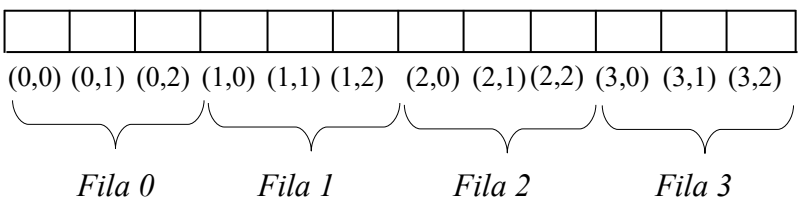
   El nombre del fichero se pasará como argumento en la línea de órdenes.

8. Escribe un programa que, dada una imagen de niveles de gris en formato PGM, obtenga otra nueva imagen con la representación de la imagen original en negativo. El programa recibirá como parámetros los nombres de las imágenes original y resultado.

   Un fichero PGM tiene un **formato mixto texto-binario** con cabecera:
   - La cabecera está en formato texto y consta de:
     - Un número mágico para identificar el tipo de fichero. En nuestro caso, de imágenes de niveles de gris, tiene asignado la cadena "P5".
     - Un número indeterminado de comentarios marcados delante con #. Los ejemplos que se dejan para probar el programa no contienen ningún comentario (no es necesario comprobar si existen y leerlos). Si se utilizan otros ficheros de imágenes, éstas pueden presentar comentarios en la cabecera.
     - Dos enteros *c* y *f* que representan, **en este orden**, el número de columnas y filas, separados por un número indeterminado de espacios.
     - Un entero con el valor del mayor nivel de gris que puede tener la imagen (*m*).
     - Cada una de estas informaciones está terminada, por un salto de línea.
   - El contenido de la imagen está en formato binario.
     - Una secuencia binaria de *f x c* `unsigned char`, con valores entre 0 y *m*. Cada uno de estos valores representa un nivel de gris de un pixel. La imagen se almacena por filas, es decir, el primer pixel es la esquina superior izquierda, el segundo el que está a su derecha, etc.

   En la imagen siguiente se muestra, a modo de ejemplo, como se almacena una imagen de 4 filas y 3 columnas en este formato:

```
P5\n
#ejemplo de comentario\n
3 4\n
255\n
```



   - El fichero se abrirá en modo "*wb*" para escritura y "*rb*" para lectura.
   - Para visualizar las imágenes en Linux local puede utilizar *gimp* o *eog* cuya sintaxis es:

eog <fichero con la imagen>

El negativo es una operación píxel a píxel en la que se genera una imagen similar al negativo de una fotografía. Para ello:

```
valorPixelNegativo=255-valorPixel
```

Imagen original

Negativo de la imagen

Recursividad

9. Escribe una solución recursiva que calcule el algoritmo de *Euclides*, usado para calcular el máximo común divisor de dos enteros. El algoritmo de *Euclide*s se describe del siguiente modo:

```
mcd(x, y) = x, si y = 0
mcd(x, y) = mcd (y, mod(x,y)) si y > 0
Antes de llamar a la función, comprobar que  x >= y
```

10. Construye una función recursiva que calcule la división entera de dos números mediante el métodos de restas sucesivas. Implementa un pequeño programa para probarla.7. Por ejemplo, si 28 es el dividendo y 9 el divisor, el programa irá restando el divisor al dividendo de forma sucesiva, mientras el dividendo sea mayor o igual que 0. el número de veces que se haga esta resta será la división entera.

    28 – 9 = 19 > 0 (1 vez)
    19 -9 = 10 > 0 (2 veces)
    10 -9 = 1 > 0 (3 veces)
    1 -9 = -8 < 0 (esta no cuenta).
    Por tanto, la división entera es 3, ya que se han realizado 3 iteraciones.

11. Construye una función recursiva que permita invertir los elementos de un vector. Por ejemplo, si el vector original es V=(3,7,2,6,7,8), el resultado de la llamada será V=(8,7,6,2,3,7). Puedes pasar como parámetros el vector y dos posiciones a intercambiar.

12. Dada una cadena *c,* diseña una función recursiva que cuente la cantidad de veces que aparece un carácter *x* en *c*.
    Ejemplo*: para c = "elementos de programacion" y x = 'e', el resultado es 4*.

13. Codifica una función recursiva que permita sumar los dígitos de un número entero positivo. Implementa un programa para probarla. Ejemplo: Entrada: 124 Resultado: 7.

14. Codifica una función recursiva que permita calcular el valor de π usando la serie de Leibniz. Escribe un programa para probarla que pedirá al usuario que introduzca el número *n* de términos a usar en la aproximación.

$$\pi \approx \sum_{n=0}^{\infty} 4\frac{(-1)^n}{2n+1} = 4 - 4/3 + 4/5 - 4/7 + 4/9 - 4/11 + 4/13 \ldots$$