**Testing-Text**

**PRÁCTICA FUNDAMENTOS DE PROGRAMACIÓN**

La práctica consiste en elaborar un programa en C que responda “preguntas” sobre el contenido de un fichero de texto plano (.txt) o realice sobre el mismo algún tipo de modificación. Cada tipo de orden se indicará introduciendo parámetros al propio programa ejecutable por medio de los argumentos “argc” y “argv” de la función ‘main’.

**int main(int argc, char\* argv[])**

Por tanto, la ejecución del programa se realizará mediante la consola del sistema operativo (la ventana de fondo negro de línea de comandos → comando ***cmd***). Cada vez que el programa sea ejecutado, este analizará los argumentos o parámetros introducidos, los analizará para ver si son correctos o no, en caso de ser correctos se realizará la operación correspondiente, mostrando el resultado de la misma por pantalla (cuando corresponda mostrar un resultado); si los argumentos no son correctos o la operación indicada no se puede realizar, se mostrará por pantalla un mensaje de error indicando cuál ha sido el fallo. Por último, tanto si la operación se ha realizado correctamente, como si se ha producido un error, el programa imprimirá el texto “**FIN\n**” y finalizará, es decir, en la consola de comandos volverá a verse el prompt del sistema operativo.

|  |
| --- |
| **// Esquema/Algoritmo general del programa**  **// El programa recibe diversos parámetros de entrada**  **1. Analizar los parámetros recibidos**  **2. Si los parámetros son correctos**  **3. Ejecutar la operación que corresponda**  **4. Si los parámetros son incorrectos**  **5. Mostrar mensaje de error**  **6. Imprimir “FIN\n” y finalizar programa** |

Como más adelante se verá, muchas de las operaciones que habrá que hacer consistirán en identificar o contar ciertas palabras o tokens dentro de un fichero de texto, por ‘token’ se entiende un conjunto de caracteres de cualquier tamaño que está delimitado o separado del siguiente token (o del anterior) por algún carácter concreto, el carácter más usual como separador es el espacio en blanco, pero hay otros como el punto (.), la coma (;), etc. En la siguiente tabla se enumeran los once caracteres separadores (entre comillas simples) que se deberán considerar en la práctica para determinar dónde empieza y termina un token:

|  |  |  |  |
| --- | --- | --- | --- |
| ' ' espacio  '\t' tabulador  '\n' Salto de línea | ',' coma  '.' punto  ';' punto y coma | ':' dos puntos  '(' parént. izq.  ')' parént. der. | '[' corchete izq.  ']' corchete der. |

Ya se ha indicado que para ejecutar el programa, este deberá ser invocado desde la línea de comandos, a continuación se van a enumerar los comandos válidos a los que el programa deberá poder responder. Nótese que cada comando empieza con el nombre del programa, este podrá ser diferente para cada caso, según como haya sido generado, en los siguientes ejemplos se va a suponer que nuestro programa ejecutable se ha generado con el nombre **prog** (o **prog.exe**). Las palabras en negrita de cada comando son palabras que deberán escribirse literalmente tal cual se muestran, el resto serán argumentos que podrán variar. Cuando un argumento aparezca entre corchetes, eso quiere decir que se trata de un argumento opcional.

Hay un parámetros denominado ‘fichero’ que se utiliza en la mayoría de los comando válidos, se refiere al nombre de un fichero de texto que el programa deberá abrir para hacer algún tipo de operación. Si en algún momento dicho fichero no se pudiera abrir (porque no existe o porque no se tienen permisos, por ejemplo) el programa no realizará ninguna operación, salvo indicar con un mensaje de error que el fichero indicado no se puede abrir (o no existe).

**prog**

Va sin parámetros (solo el nombre del programa ejecutable). Escribe en pantalla la sintaxis de todos los comandos disponibles a modo de ayuda para el usuario.

**prog info**

Muestra por pantalla la siguiente información personal del alumno:

- Nombre y apellidos

- email

**prog** fichero **cuenta** palabra1 [palabra2] [palabra3] [palabra4] ...

Abre el fichero de texto indicado y cuenta cuantas veces se repita cada una de las palabras de la lista en dicho fichero. Al menos ha de indicarse una palabra (palabra1), de manera opcional se podrán indicar más palabras si se desea.

**prog** fichero **busca** fichero2

Análogo al anterior, salvo que ahora las palabras que hay que contar se encuentran en el fichero2, que también habrá que abrir leer para ver cuales son, dichas palabras estarán separadas por los mismos separadores que los tokens del primer fichero. Si el fichero2 no se pudiera abrir también se indicará con un mensaje de error y no se hará nada más. También sería un error que fichero2 estuviera vacío (que no tuviera ninguna palabra).

**prog** fichero **numeros**

Cuenta cuántos de los tokens del fichero son números, positivos o negativos, enteros o reales con decimales. IMPORTANTE, el carácter para indicar los decimales es el punto (.)

**prog** fichero **fechas**

Cuenta cuántos de los tokens del fichero son fechas correctas en el formato DD/MM/AAAA (2 dígitos para el día, 2 para el mes y 4 para el año).

**prog** fichero **remplazar** palabra1 palabra2

Sustituye, en el fichero leído, la palabra1 por la palabra2. El resultado se guardará en un fichero nuevo cuyo nombre estará basado en el primero. Por ejemplo, si el fichero se llama “mi\_texto.txt”, el nuevo fichero deberá llamarse “mi\_texto\_new.txt”, en caso de que ya existe un fichero con ese nombre, el nuevo fichero deberá llamarse “mi\_texto\_new(2).txt” (o “mi\_texto\_new(3).txt”, etc...). El programa responderá (además de creando el nuevo fichero) mostrando por pantalla la cantidad de veces que ha reemplazado ‘palabra1’ por ‘palabra2’.

Y hasta aquí, los comandos y parámetros que debe reconocer el programa que hay que realizar. Cada alumno puede diseñar libremente las estructuras de datos que considere más apropiadas para implementar cada una de las operaciones que se han descrito. El único requisito que se exige es que la gestión de memoria ha de ser óptima, de tal forma que en cada momento el programa reservará exactamente la memoria que necesite y la liberará cuando ya no le sea necesaria, y siempre, antes de finalizar.

La prueba y corrección de la práctica se realizará usando métodos automáticos, por lo tanto, la estructura del programa deberá constar **OBLIGATORIAMENTE** de los tres fichero de código siguientes:

* **main.c** → debe contener únicamente la función main() del programa (y los includes necesarios).
* **lib.h** → para los prototipos de las funciones y las estructuras de datos que se vayan a declarar (si son necesarias).
* **lib.c** → para el código del resto de funciones (todas menos main).

Para compilar y ejecutar el código de la práctica con el editor Visual Studio Code será necesario un fichero ***tasks.json*** con el siguiente contenido:

|  |
| --- |
| **{**  **// See https://go.microsoft.com/fwlink/?LinkId=733558**  **// for the documentation about the tasks.json format**  **"version": "2.0.0",**  **"tasks": [**  **{ // tarea 1: Compilar el código**  **"label": "Compilar",**  **"type": "shell",**  **"command": "gcc",**  **"args": ["-g","-o","prog.exe","lib.c","main.c"],**  **"group": "build",**  **"problemMatcher": ["$gcc"]**  **},**  **{ // tarea 2: Ejecutar el programa en el terminal de VS-Code**  **"label": "Ejecutar",**  **"dependsOn": ["Compilar"],**  **"type": "shell",**  **"command": "${workspaceFolder}\\prog.exe",**  **"args": ["poner","argumentos","aquí"],**  **"group": {**  **"kind": "build",**  **"isDefault": true**  **},**  **"problemMatcher": ["$gcc"]**  **}**  **]**  **}** |

**NOTA:** Cualquier duda o cuestión importante que los alumnos tengan sobre la práctica se deberá plantear en el foro de la asignatura del curso actual:

<https://campus.umh.es/mod/forum/view.php?id=43768>

**Algunas consideraciones importantes**

* El estudiante deberá dominar su práctica de tal manera que pueda realizar ciertos cambios y/o mejoras de cierta complejidad en el periodo de tiempo que dura un examen (el estudiante que no conozca bien su código no será capaz de realizar esos cambios).
* La asignatura se evaluará mediante la realización de un único examen práctico en aula de ordenadores utilizando el mismo entorno de desarrollo que se ha visto en las sesiones prácticas.
* El examen consistirá en una prueba práctica en ordenador en la que se solicitará al estudiante que realice cambios en su propia práctica, se espera que el alumno modifique su práctica dando como resultado un nuevo programa sin errores ni warnings, que se pueda ejecutar, y que incorpore las citadas modificaciones.
* A la hora de calcular la nota final, se valorarán aspectos como la claridad del código, la creación de módulos y funciones útiles, la no duplicidad de código, si las práctica contiene warnings, la correcta gestión de ficheros y memoria dinámica, y que todos los requerimientos de la práctica original estén correctamente implementados. Estos aspectos pueden reducir la nota final obtenida en el examen.
* Un programa que no se pueda ejecutar por contener errores o por no cumplir con la estructura de librerías solicitada en la práctica implicará una calificación de “No Presentado”.
* Es condición necesaria para poder presentarse al examen final el haber realizado la práctica, haberla entregado dentro del plazo que se indique, y que dicha práctica pueda compilarse y ejecutarse (excepto en la convocatoria de diciembre). De no cumplirse estos criterios el alumno no podrá presentarse al examen y su calificación será “No Presentado”.
* Queda a criterio del docente el convocar a los alumnos que considere a una revisión del examen para que estos prueben la autoría del mismo. Si un estudiante es convocado a la revisión de su examen, en caso de que no demuestre ser el autor del mismo obtendrá una nota final de 0.0 (cero).
* Si en el examen se detectarán copias entre alumnos también se asignará una nota de 0.0 (cero) a todos los alumnos implicados.