Grado en Ingeniería Informática. Metodología de la Programación. Relación de Problemas VI: Gestión de E/S. Ficheros (II)

1. Escribir un programa que reciba los nombres de dos ficheros de *texto* de la línea de órdenes. El programa creará un fichero (cuyo nombre se especifica en el segundo argumento) a partir de un fichero existente (cuyo nombre se especifica en el primer argumento) copiando su contenido y añadiendo al principio de cada línea, su número.

2. Escribir un programa similar a `diff` para comparar dos ficheros de texto. El programa imprimirá el número de la primera línea en la que difieren y el contenido de éstas. Por ejemplo, la ejecución de `diff Fich1 Fich2` producirá como resultado:

   ```
   (  20) Fich1: en formato binario. Estos ficheros son especialmente adecuados para
          Fich2: en formato binario. Estos ficheros, aunque no son legibles, son especialmente
   ```

   si las 19 primeras líneas de `Fich1` y `Fich2` son idénticas, y la primera diferencia se encuentra en la línea 20.

   Nota: Este programa puede ser útil para comprobar si después de encriptar y desencriptar un fichero (problema 4), obtenemos un fichero idéntico al original.

3. Escribir un programa que reciba como parámetros tres nombres de ficheros de *texto*. Los dos primeros ficheros contienen números reales y están *ordenados*. El programa tomará los datos de esos ficheros y los irá copiando ordenadamente en el tercer fichero, de forma que al finalizar esté también ordenado.

4. Escribir un programa que permita encriptar y desencriptar el contenido de un fichero de texto. Para *encriptar* sustituiremos cada letra (mayúsculas y minúsculas) por la letra que está p posiciones más adelante en el alfabeto (para las últimas letras ciclamos el alfabeto). Los caracteres que no sean letras se quedarán igual. Para *desencriptar* la sustitución será a la inversa. La llamada al programa se realizará con este esquema:

   ```
   codifica <ficheroE> <ficheroS> <p> <tipo>
   ```

   donde:

   - `<ficheroE>` y `<ficheroS>` son los nombres de los ficheros de entrada y salida, respectivamente
   - `<p>` es el número entero positivo que se aplica para codificar/descodificar cada uno de los caracteres.
   - `<tipo>` es una cadena de caracteres que puede valer: `enc` para encriptar y `desenc` para desencriptar.

5. Un fichero de *texto* contiene números enteros que se disponen en líneas y en cada línea están separados por espacios en blanco. Todas las líneas tienen el mismo número de elementos.

   Se trata de escribir un constructor para las clases `Matriz2D_1` y `Matriz2D_2` que reciba el nombre de un fichero con la estructura descrita y rellene las casillas de la matriz con los datos contenidos en el fichero. Se sobreentiende que los datos están guardados *por filas* en el fichero.

   Las restricciones que se imponen, y que se deben cumplir en la resolución son:

   a) El fichero sólo puede ser leído una única vez, y no pueden copiarse completo en memoria.
   b) Se desconoce *a priori* el número de líneas del fichero.
   c) Las líneas del fichero tiene una longitud *indeterminada*, aunque nunca mayor de 500.
   d) El número de datos de cada línea es *indeterminado*, aunque éste es *común* para todas las líneas.

e) No puede emplearse una matriz con un número de filas "tentativo" sino que la matriz ocupará en cada momento el espacio estrictamente necesario y los datos se copiarán conforme se lean cada una de las filas.

El número de líneas de datos del fichero debe coincidir con el número de filas de la matriz, y el número de elementos de cada fila con el número de columnas.

Nota: Es posible que sea necesario añadir un nuevo método a las clases `Matriz2D_1` y `Matriz2D_2` que permita redimensionar la matriz, añadiendo una nueva fila.

6. Se dispone de ficheros de *texto* que contienen un número indeterminado de líneas, cada una de ellas con los datos correspondientes a una serie de grupos de valores reales.

   Por ejemplo, una línea de entrada podría ser la siguiente:

   ```
   3 2 3.1 0.4 5 1.0 1.0 1.0 1.0 1.0 2 5.2 4.7
   ```

   donde puede observar que se distinguen tres grupos de datos (indicado por el primer número de la línea) y cada grupo empieza por un valor entero (2, 5 y 2) seguido por tantos valores reales como indique el valor entero que encabeza cada grupo:

3	**2**	*3.1*	*0.4*	**5**	*1.0*	*1.0*	*1.0*	*1.0*	*1.0*	**2**	*5.2*	*4.7*

   Escribir un programa que escriba en la salida estándar una línea de resultado por cada línea de entrada, y en cada línea mostrará las sumas de los valores de cada grupo que la componen.

   Por ejemplo, en el caso anterior, debería escribir:

   ```
   3.5    5.0    9.9
   ```

   El programa se ejecutará desde la línea de órdenes y permitirá:

   a) Llamarlo sin ningún argumento. En este caso, los datos de entrada se leerán desde la entrada estándar.

   b) Llamarlo con un argumento. El argumento corresponde al nombre del archivo con las líneas de entrada.

   Las restricciones que se imponen, y que se deben cumplir en la resolución son:

   a) El fichero sólo puede ser leído una única vez, y no pueden copiarse completo en memoria.

   b) Se desconoce *a priori* el número de líneas del fichero.

   c) Las líneas del fichero tiene una longitud *indeterminada*, aunque nunca mayor de 500.

7. Escriba dos programas para transformar ficheros con datos correspondientes a una serie de grupos de valores reales (como están descritos en el problema 6), para transformar entre formato binario y texto:

   a) Un programa que transforme un fichero de texto a binario:

   ```
   text2bin <FichText> <FichBin>
   ```

   b) Un programa que transforme un fichero de binario a texto:

   ```
   bin2text <FichBin> <FichText>
   ```

   Debe optimizarse el uso de los recursos, y por tanto, se aplican las restricciones enumeradas en el problema 6.

8. Escribir un programa que reciba el nombre de dos ficheros. El programa copiará, en el mismo orden, los números que contiene el fichero de entrada en el fichero de salida.

   - El primer fichero (entrada) contiene una serie indefinida de números enteros:
     - Es un fichero de *texto*.
     - Se pueden usar espacios, tabuladores o saltos de líneas (en cualquier número y combinación) para separar dos números enteros consecutivos.
   - El segundo fichero (salida) es un fichero *binario*.
   - El programa leerá los números y los copiará **de uno en uno**.

9. Escribir un programa con las mismas características que las descritas en el problema 8 pero que escriba en el fichero de salida **bloques de 512 bytes**.

10. Escribir un programa que lea un fichero *binario* como los generados en los problemas 8 y 9 y que muestre en la salida estándar la suma de todos esos números. Para la lectura se empleará un **buffer de 512 bytes**.

11. Construir un programa que divida un fichero de *texto* en diferentes ficheros indicando como argumentos el nombre del fichero original y el **máximo número de líneas** que contendrá cada fichero resultante.

    Se creará un fichero de control que contendrá con los datos necesario para la reconstrucción del fichero original.

    Por ejemplo, si `Fichero` contiene 1600 líneas, la ejecución de `parte_lineas Fichero 500` genera como resultado los ficheros `Fichero_1`, `Fichero_2`, `Fichero_3` y `Fichero_4`. Los tres primeros contienen 500 líneas de `Fichero` y el último, las 100 restantes. Se creará un fichero *oculto* llamado `.Fichero.ctrl` que contendrá (formato texto, en dos líneas separadas): nombre del fichero original y número de ficheros resultantes de la partición.

    Nota: Utilizad *cadenas* para construir los nombres de los ficheros resultantes.

12. Construir un programa que divida un fichero de *cualquier tipo* en diferentes ficheros, indicando como argumentos el nombre del fichero original y el **máximo número de bytes** que contendrá cada fichero resultante.

    Por ejemplo, si el tamaño de `Fichero` es 1800 bytes, la ejecución de `parte_bytes Fichero 500` genera como resultado los ficheros `Fichero_1`, `Fichero_2`, `Fichero_3` y `Fichero_4`. Los tres primeros contienen 500 bytes de `Fichero` y el último, los 300 restantes. Se creará un fichero *oculto* llamado `.Fichero.ctrl` que contendrá (formato texto, en dos líneas separadas): nombre del fichero original y número de ficheros resultantes de la partición.

    Nota: Utilizad *cadenas* para construir los nombres de los ficheros resultantes.

13. Construir un programa que reconstruya un fichero a partir de una serie de ficheros que contienen sus "partes". Los ficheros que pueden emplearse como origen se han creado con los programas descritos en los problemas 11 y 12 y por ese motivo se emplará el fichero de control creado por esos programas.

    Por ejemplo, la ejecución de `reconstruye Fichero` genera como resultado `Fichero`. Usará `.Fichero.ctrl` para conocer los ficheros que debe usar y el orden en que se debe hacer la reconstrucción.

    Nota: Utilizad *cadenas* para construir los nombres de los ficheros que intervienen.

14. Repetir los problemas 11, 12 y 13 usando la clase `stringstream` para componer los nombres de los ficheros.

15. Escribir un programa similar a `grep` que busque una cadena en una serie de ficheros de texto. La cadena a buscar y los ficheros en los que buscar se proporcionan en la línea de órdenes. Por ejemplo:

    `busca Olga fich1 fich2 fich3`

    busca la cadena `Olga` en los ficheros `fich1`, `fich2` y `fich3`.

    Cada vez que encuentre la cadena buscada, debe indicar el fichero en el que es localizada, el número de línea y la línea completa que la contiene. Un ejemplo de salida de este programa es:

    ```
    fich1 (línea 33): Mi amiga Olga ha aprobado MP aunque no se
    fich3 (línea 2): ya se lo dije ayer a Pepe, pero ni caso
    fich3 (línea 242): finalmente, Olga se puso a estudiar
    ```

    Las restricciones que se imponen, y que se deben cumplir en la resolución son:

    a) El número de ficheros que se pueden proporcionar es *ilimitado*.

    b) Cada uno de los ficheros sólo puede ser leído una única vez, y no pueden copiarse completos en memoria.

    c) Se desconoce *a priori* el número de líneas de los ficheros.

    d) Las líneas de los ficheros tienen una longitud *indeterminada*, aunque nunca mayor de 500.

16. Implementar un programa que similar a `head` que muestre las primeras líneas de un fichero de texto.

    Por ejemplo, la ejecución de `cabecera 15 reconstruye.cpp` mostrará las primeras 15 líneas del fichero de texto `reconstruye.cpp`

    Se aplican las mismas restricciones que las indicadas en el problema 15(excepto la primera, evidentemente).

17. Implementar un programa que similar a `tail` que muestre las últimas líneas de un fichero de texto.

    Por ejemplo, la ejecución de `final 15 reconstruye.cpp` mostrará las últimas 15 líneas del fichero de texto `reconstruye.cpp`

    Se aplican las mismas restricciones que las indicadas en el problema 15 (excepto la primera, evidentemente).

18. Una empresa de distribución mantiene la información acerca de sus vendedores, productos y ventas en ficheros informáticos. Los ficheros almacenan la información en formato *binario* y la estructura de los *registros* es:

    - Fichero `Vendedores`:
      `RegVendedor`: `CodVendedor` (`unsigned char`), `Nombre` (50*`char`) y `CodZona` (`unsigned char`).
    - Fichero `Articulos`:
      `RegArticulo`: `CodArticulo` (10*`char`), `Descripcion` (30*`char`) y `PVP` (`float`).
    - Fichero `Ventas`:
      `RegVenta`: `NumFactura` (`int`), `CodVendedor` (`unsigned char`), `CodArticulo` (10*`char`) y `Unidades` (`int`).
      Se supone que el fichero `Ventas` contiene las ventas realizadas en un mes.

    Se trata de realizar programas para:

    a) Mostrar el total (número de ventas y cantidad total de ventas) de las ventas realizadas por un vendedor, dado su código (`CodVendedor`).

b) Pedir una cantidad y crear un fichero (*binario*) llamado `VendedoresVIP` cuyos registros tengan la siguiente estructura:

`RegVendedorVIP`: `CodVendedor` (`unsigned char`), `CodZona` (`unsigned char`), `TotalVentas` (`float`).

donde `TotalVentas` es la cantidad total de ventas realizadas por el vendedor.

El fichero `VendedoresVIP` tendrá únicamente los registros de los vendedores cuyo total de ventas sea superior a la cantidad leída.

19. Algunos ficheros se identifican mediante una "cabecera" especial, como los archivos PGM que tienen en la posición cero los caracteres `P5`. Este tipo de ficheros, y las marcas que los identifican son los que se gestionarán en este problema.

Un fichero de descripciones es un fichero *binario* que almacena las distintas *marcas* que identifican a tipos de ficheros, así como información acerca de dónde se ubican en los archivos.

Los registros del fichero `Descripciones` tienen longitud variable y su formato es el siguiente:

`RegDescripcion`: `PosInicioMarca` (`int`), `LongMarca` (`int`), `Marca` (`LongMarca`*`char`) y `Comentario` (100*`char`)

Un ejemplo (en forma tabular) de los contenidos de este archivo podría ser:

0	2	P5	Imagen PGM
0	2	P6	Imagen PPM
10	8	DAT CIEN	Datos Científicos

Donde podemos ver que si un archivo tiene la cadena "DAT CIEN" (8 caracteres) a partir de la posición 10 del archivo, se trata de un archivo de tipo "Datos Científicos".

Las tareas a realizar son:

a) Escribir una función (`Insertar`) para añadir descripciones.

La función recibirá el nombre de un archivo de descripciones junto con una nueva entrada (un dato de tipo `RegDescripcion`) y añadirá esa entrada a dicho archivo (creándolo si es necesario).

b) Implementar una función (`TipoArchivo`) que determine el tipo de un fichero.

La función recibirá el nombre del fichero del que queremos averiguar si tipo junto con el nombre del archivo de descripciones. Como resultado devuelve una cadena con el comentario asociado, o la cadena *Tipo desconocido* si no se ha localizado su tipo.

c) Escribir un programa que, usando la función del apartado anterior, reciba de la línea de órdenes el nombre de un archivo y escriba en la salida estándar la descripción (*comentario*) asociado a su tipo. Tenga en cuenta los posibles casos de error.

20. Hacer un programa que permita formar el nombre de un fichero de la forma: `salidaXXX.Z.dat` a partir de dos números que recibe de la línea de órdenes:

- `XXX` es un número de 3 dígitos (se rellena con ceros a la izquierda si es necesario), y
- `Z` es un número con cualquier cantidad de dígitos.

A continuación se presentan varios ejemplos de ejecución (nuestro programa se llamará `componer`):

`componer 45 6` generaría el nombre de fichero `salida.045.6.dat`

`componer 5 67` generaría el nombre de fichero `salida.005.67.dat`

Nota: usar la clase `stringstream`