Ejercicios básicos

1. Escritura de un fichero de texto

Escribe un programa que solicite al usuario un número entero entre 1 y 10 y el nombre de un fichero y guarde en un fichero con el nombre introducido por el usuario la tabla de multiplicar del número introducido por el usuario.

Puedes basarte en el código del programa «tabla-multiplicar» del tema 4 (https://github.com/prog1-eina/tema-04-instrucciones/blob/master/3-tabla-multiplicar.cpp).

Ejemplo de ejecución:

```
Introduzca un número: 4
Nombre del fichero: tabla.txt
```

Tras la ejecución del ejemplo anterior, se ha creado en el directorio de ejecución un fichero denominado «tabla.txt», con el siguiente contenido:

```
LA TABLA DEL 4
  4 x  0 =   0
  4 x  1 =   4
  4 x  2 =   8
  4 x  3 =  12
  4 x  4 =  16
  4 x  5 =  20
  4 x  6 =  24
  4 x  7 =  28
  4 x  8 =  32
  4 x  9 =  36
  4 x 10 =  40
```

2. Lectura de un fichero de texto

Escribe un programa que solicite al usuario el nombre de un fichero de texto y un carácter. Si el fichero existe y se puede abrir, el programa escribe en la pantalla el número de veces que el carácter introducido aparece en el fichero.

Ejemplos de varias ejecuciones distintas:

```
Nombre del fichero: quijote.txt
Escriba el carácter a contar: a
El carácter 'a' aparece 191457 veces.
```

```
Nombre del fichero: quijote.txt
Escriba el carácter a contar: A
El carácter 'A' aparece 1887 veces.
```

```
Nombre del fichero: 2-contar-caracter.cpp
Escriba el carácter a contar: {
El carácter '{' aparece 9 veces.
```

```
Nombre del fichero: fichero-que-no-existe.txt
Escriba el carácter a contar: t
No ha podido leerse el fichero "fichero-que-no-existe.txt".
```

3. Lectura y escritura conjuntas ficheros de texto

Escribe un programa que solicite al usuario el nombre de un fichero de texto existente, el nombre de un fichero nuevo y dos enteros que representan un intervalo de líneas. El programa debe copiar en el fichero nuevo las líneas del fichero existente cuyo número se encuentren dentro del intervalo introducido por el usuario.

Si el intervalo no es válido (es decir, el segundo entero es menor que el primero) o si el número de líneas del fichero existente es menor que el valor inicial del intervalo, creará el fichero nuevo, pero lo dejará sin contenido. Si el número de líneas del fichero existente es inferior al valor final del intervalo, se copiarán únicamente las líneas que se pueda del fichero existente, hasta llegar a su final.

Ejemplo de ejecución:

```
Nombre de un fichero existente: quijote.txt
Nombre del fichero nuevo: principio.txt
Escriba el intervalo de líneas a copiar: 601 606
Fichero "principio.txt" creado con éxito.
```

Tras la ejecución del ejemplo anterior, se ha creado en el directorio de ejecución un fichero denominado «principio.txt», con el siguiente contenido:

```
En un lugar de la Mancha, de cuyo nombre no quiero acordarme, no ha mucho
tiempo que vivía un hidalgo de los de lanza en astillero, adarga antigua,
rocín flaco y galgo corredor. Una olla de algo más vaca que carnero,
salpicón las más noches, duelos y quebrantos los sábados, lantejas los
viernes, algún palomino de añadidura los domingos, consumían las tres
partes de su hacienda. El resto della concluían sayo de velarte, calzas de
```

Otros ejemplos de ejecución, en los que no se puede localizar el fichero a leer o no se puede crear el fichero a escribir:

```
Nombre de un fichero existente: fichero-que-no-existe.txt
Nombre del fichero nuevo: principio.txt
Escriba el intervalo de líneas a copiar: 32 42
No ha podido leerse el fichero "fichero-que-no-existe.txt".
```

```
Nombre de un fichero existente: quijote.txt
Nombre del fichero nuevo: fichero>que>no>se>puede>crear.txt
Escriba el intervalo de líneas a copiar: 16 32
No ha podido escribirse el fichero "fichero>que>no>se>puede>crear.txt".
```

Problemas

4. Filtrado de un fichero de texto

Se dispone de ficheros de texto que responden a la siguiente sintaxis expresada en notación de Backus-Naur:

```
<fichero-alumnos> ::= { <alumno> }
<alumno> ::= <nip> <separador> <grupo> <separador> <nombreCompleto> fin_línea
<nip> ::= literal_entero
<grupo> ::= literal_entero
<nombre-completo> ::= literal_string
<separador> ::= “ ”
```

Por ejemplo, se supone la existencia de un fichero denominado «`alumnos.txt`» que sigue dicha sintaxis y cuyo contenido aparece a continuación (la negrita no forma parte del fichero de texto, se ha añadido exclusivamente para facilitar su lectura en este enunciado):

```
747752 411 JORGE RODRIGUEZ SIPAN
705318 412 PEDRO CUERDA ESCUER
580368 412 LORETO BERGUA MONGE
600107 411 JOSE ANDRES ZAMORA BADENES
502491 411 TIBURCIO JOSA GALO
742611 411 JACINTO BORAO BERNARDEZ
513678 411 ESCOLASTICO MATEU FRANCES
666327 412 ROLDAN ANTUNEZ ZAMORA
509239 412 AZUCENA AZNAR POBLADOR
761907 412 ROLDAN MARTINEZ NAVARRO
614206 411 LIDIA BERNARDEZ MOSQUERA
710384 412 RODRIGO AZANZA SIPAN
700648 411 RUBEN CANDAS ADANEZ
516733 411 FERNANDO CARPINTERO DUQUE
564584 411 JUAN IGNACIO ABADIA AZANZA
587194 412 IVAN REMACHA JUAN
609680 411 LUCAS GALO OTANO
573900 411 RUBEN JULIAN MOLINS
630947 411 REMEDIOS MONGE MURIEL
623042 411 JACINTA ZAPATERO ALONSO
```

Se pide completar el diseño de la función `filtrar` cuya especificación aparece a continuación:

```
/*
 * Pre:  «nombreOrigen» es el nombre de un fichero de texto donde cada línea
 *       corresponde a los siguientes datos de un alumno, separados por espacios en
 *       blanco: NIP, grupo y nombre completo.
 * Post: Ha filtrado un fichero cuyo nombre es el valor de «nombreOrigen» para dejar,
 *       en un fichero cuyo nombre es el del valor del parámetro «nombreDestino»,
 *       solo el NIP y el nombre completo de aquellos alumnos que pertenecen
 *       al grupo indicado por el valor del parámetro «grupo».
 */
void filtrar(const char nombreOrigen[], const char nombreDestino[],
             unsigned int grupo);
```

Es decir, la invocación `filtrar("alumnos.txt", "alumnos-tardes.txt", 412)` debería generar un fichero denominado «`alumnos-tardes.txt`» con el siguiente contenido:

```
705318 PEDRO CUERDA ESCUER
580368 LORETO BERGUA MONGE
666327 ROLDAN ANTUNEZ ZAMORA
509239 AZUCENA AZNAR POBLADOR
761907 ROLDAN MARTINEZ NAVARRO
710384 RODRIGO AZANZA SIPAN
587194 IVAN REMACHA JUAN
```

5. Notación Backus-Naur

Describe la sintaxis de los ficheros producto de las invocaciones a la función filtrar anterior utilizando notación Backus-Naur.

6. Cálculo de las frecuencias de aparición de las letras del alfabeto

Se pide diseñar las dos funciones siguientes:

```
/*
 * Pre:  «nombreFichero» es el nombre de un fichero de texto válido y el número de
 *       componentes de la tabla «frecuencias» es igual al número de letras del
 *       alfabeto inglés.
 * Post: Ha asignado a la tabla «frecuencias» el número de apariciones de cada una de
 *       las letras del alfabeto inglés en el fichero cuyo nombre es «nombreFichero»,
 *       no distinguiendo entre mayúsculas y minúsculas.
 */
void analizar(const char nombreFichero[], unsigned int frecuencias[]);

/*
 * Pre:  ---
 * Post: Ha escrito en la pantalla una tabla con las frecuencias de apariciones de
 *       las letras según los valores de la tabla «frecuencias». La primera
 *       componente de la tabla es el número de veces que aparece la letra A, la
 *       segunda, la letra B y así sucesivamente.
 */
void escribirFrecuencias(unsigned int frecuencias[]);
```

A modo de ejemplo, la siguiente secuencia de declaraciones e instrucciones debería escribir en la pantalla el resultado que se reproduce a continuación:

```
const unsigned int NUM_LETRAS = 'Z' - 'A' + 1;
unsigned int frecuencias[NUM_LETRAS];
analizar("quijote.txt", frecuencias);
escribirFrecuencias(frecuencias);
```

```
A:  193344
B:   24146
C:   59435
D:   87237
E:  221979
F:    7581
G:   17225
H:   19920
I:   77615
J:   10530
K:       0
L:   89141
M:   44658
       ...
```

```
       ...
N:  108440
O:  153359
P:   35464
Q:   32483
R:  100953
S:  125726
T:   61749
U:   78193
V:   17855
W:       2
X:     377
Y:   25115
Z:    6491
```