EJERCICIO 1 (Recuperación del primer parcial)

El programa (ejer01.js) que se muestra a continuación recibe como argumento el nombre de dos ficheros de texto, ordena el contenido de cada uno, y genera nuevos ficheros con el sufijo 2 (ej. node ejer01.js a b genera ficheros a2 y b2).Tras escribir cada fichero ordenado muestra un mensaje informativo, y al completar todas las operaciones un mensaje final.

Asume que no hay errores en las operaciones con los archivos, ni en el número de argumentos. Debes tener un cuidado especial en los momentosen que se producen las operaciones de escritura (console.log). Modifica este programa para estos casos independientes:

1. Puede comenzar a procesar el segundo fichero sin necesidad de esperar a que el primero finalice.
2. Puede procesar un número arbitrario de ficheros (argumentos), sin obligar a una ordenación secuencial, pero manteniendo el significado y corrección de las operaciones (especialmente escrituras en fichero y en pantalla).

```
1:  var fs = require('fs')
2:  var f1 = process.argv[2]
3:  var f2 = process.argv[3]
4:
5:  function sl(s) {
6:      return s.toString().split("\n").sort().join("\n")
7:  }
8:  s.readFile(f1, 'utf-8', function (err, data) {
9:      s.writeFile(f1 + "2", sl(data), 'utf-8', () => {
10:           console.log('ok '+ f1)
11:           fs.readFile(f2, 'utf-8', function (err, data) {
12:               fs.writeFile(f2+"2",sl(data),'utf-8',() => {
13:                  console.log('ok '+ f2)
14:                  console.log('terminated')
15:               }) // writeFile(f2)
16:           }) // readFile(f2)
17:       }) // writeFile(f1)
18:   }) // readFile(f1)
```

Solución

Primer apartado (2 ficheros)

```
1:  var fs = require('fs')
19:   var f1 = process.argv[2]
20:   var f2 = process.argv[3]
21:   var cont = 2 // Files remaining to process
22:   function sl(s) { return s.toString().split("\n").sort().join("\n") }
23:
24:   fs.readFile(f1, 'utf-8', function (err, data) {
25:     fs.writeFile(f1 + "2", sl(data), 'utf-8', () => {
26:       console.log('ok '+f1)
27:       if (--cont == 0) console.log('terminated')
28:     }) // writeFile(f1)
29:   }) // readFile(f1)
30:   fs.readFile(f2, 'utf-8', function (err, data) {
31:     fs.writeFile(f2 + "2", sl(data), 'utf-8', () => {
32:       console.log('ok '+f2)
```

```
33:       if (--cont == 0) console.log('terminated')
34:     }) // writeFile(f2)
35:   }) // readFile(f2)
```

En ella, como puede observarse, el tratamiento del segundo fichero ha sido extraído del callback que gestionaba el resultado de la escritura del primer fichero. Con ello, por ser `readFile()` una función asincrónica, las lecturas de ambos ficheros empezarán casi a la vez. Tras esto, cuando cada lectura finalice, se iniciará la ordenación del contenido del fichero leído (llamando para ello a la función `sl()` y pasándole "`data`" como su argumento, pues esa variable mantiene el contenido del fichero que estamos procesando) y posteriormente se realizará la escritura de ese contenido ordenado.

Como el enunciado nos decía que no habría errores en el procesamiento de cada fichero ni en el número de argumentos recibidos, no tenemos por qué consultar el parámetro "`err`" en el callback de `readFile()`. Tampoco necesitamos comprobar si `writeFile()` ha comunicado un error mediante el argumento de su callback. Gracias a las simplificaciones propuestas podemos asumir que tanto "`f1`" como "`f2`" han recibido valores con sentido para este ejercicio.

Hay otro aspecto que debe destacarse. Cuando ambos ficheros hayan sido escritos, tendremos que mostrar en pantalla el mensaje "`terminated`". Eso debe comprobarse en el callback que gestiona el resultado de cada escritura. A priori no podemos saber cuál de los dos ficheros finalizará su escritura más tarde. Por ejemplo, si el primer fichero fuera mucho mayor que el segundo, lo normal sería que ese primero finalizara más tarde. Para que el mensaje se muestre cuando debe, hemos utilizado en esta solución un contador con valor inicial igual al número de ficheros a procesar. Cada vez que uno termine, se decrementará su valor, y cuando llegue a cero se mostrará ese mensaje.

Segundo apartado (Número arbitrario de ficheros)

```
1:  var fs = require('fs')
36:   var args = process.argv.slice(2)
37:   var cont = args.length
38:
39:   function sl(s){return s.toString().split("\n").sort().join("\n")}
40:
41:   function myread(fich){
42:     fs.readFile(fich, 'utf-8', function (err,data) {
43:       fs.writeFile(fich + "2", sl(data), 'utf-8', () => {
44:         console.log('ok '+fich)
45:         if (--cont == 0) console.log('terminated')
46:       })
47:     })
48:   }
49:
50:   for (var i in args) {
51:     myread(args[i])
52:   }
```

En este caso, el bloque formado por la lectura de cada fichero y su posterior tratamiento y escritura se han separado en una función `myread()` que incluye toda esa gestión y recibe el nombre de fichero como su único argumento. Así, basta con utilizar un bucle para recorrer la

lista de nombres de fichero recibida como argumentos por el programa y llamar en cada iteración a `myread()`.