

Gestión Dinámica de Memoria y Ficheros Binarios. Práctica laboratorio 2

Se desea generar un fichero con números aleatorios comprendidos entre $[0, tam-1]$ **diferentes y ordenados** (es decir, sin repeticiones y en orden creciente). Para ello vamos a proceder en tres pasos:

1. Se van a generar números aleatorios y se van a almacenar en un fichero binario, sin considerar su orden o repeticiones.
2. Se va a leer ese fichero, mostrar su contenido, y se va a cargar en una estructura de árbol binario de búsqueda (una lista enlazada). Durante la creación del árbol vamos a eliminar repeticiones y establecer el orden.
3. Por último, se va a guardar de esta estructura a fichero. Para comprobar que funciona bien, se va a leer de nuevo el fichero y se va a mostrar por pantalla.

Paso 1: Generar números aleatorios

Para ello, vamos a usar la función `rand()` de la librería `#include <stdlib.h>`. Este devuelve un número entre 0 y `RAND_MAX`. Para acotar al tamaño que queremos nosotros $[0, tam-1]$, hacemos uso del operador módulo:

```
rand() % tam
```

Para evitar tener siempre la misma consecución de números aleatorios cuando se llama a `random`, hay que cambiar un parámetro inicial que se llama semilla. Llamando a `srand` se puede cambiar la semilla, tiene como entrada un *unsigned*. Lo usual, suele ser tomar el Epoch, que es el número de segundos desde el 1 de Junio de 1970 ([https://en.wikipedia.org/wiki/Epoch_\(computing\)](https://en.wikipedia.org/wiki/Epoch_(computing))). Para ello hacemos uso de la función `time`, de la librería `<time.h>` con parámetro de entrada NULL:

```
srand(time(NULL));
```

Para escribir en fichero se usa `fopen` ([enlace](#) con más información).

File access mode string	Meaning	Explanation	Action if file already exists	Action if file does not exist
"r"	read	Open a file for reading	read from start	failure to open
"w"	write	Create a file for writing	destroy contents	create new
"a"	append	Append to a file	write to end	create new
"r+"	read extended	Open a file for read/write	read from start	error
"w+"	write extended	Create a file for read/write	destroy contents	create new
"a+"	append extended	Open a file for read/write	write to end	create new

File access mode flag `"b"` can optionally be specified to open a file in binary mode. This flag has no effect on POSIX systems, but on Windows it disables special handling of `'\n'` and `'\x1A'`.
On the append file access modes, data is written to the end of the file regardless of the current position of the file position indicator.

Ilustración 1 - <https://en.cppreference.com/w/c/io/fopen>

Esta función, devuelve un puntero a un manejador de fichero en C, que es de tipo `FILE`. Dado un *string* con el nombre del fichero `nfichero`, escribimos en binario de la siguiente forma:

```
FILE * pfile = fopen(nfichero, "wb");
```

La opción “wb” indica que vamos a escribir en modo binario. Al igual que con los punteros, hay que controlar que funciona correctamente la llamada a `fopen`, si `pfile` es `NULL`, no se ha podido abrir el fichero para su escritura (por ejemplo, si no tenemos permisos de escritura en dónde se quiere abrir...).

Para escribir en un fichero en binario, escribimos los bytes directamente, para ello usamos la función `fwrite`, indicando el manejador de fichero, el tamaño de un elemento a escribir (¿te suena el `sizeof` que se usa `malloc`?) y cuantos elementos se van a escribir de ese tamaño. Puedes escribir uno a uno, o puedes, almacenar los números aleatorios en un array, por ejemplo, `array_rand`, y escribir de una sola vez `tam` elementos de tamaño *unsigned*:

```
fwrite(array_rand, sizeof(unsigned), tam, pfile)
```

IMPORTANTE: Los ficheros se cierran tras su uso:

```
fclose(pfile);
```

Lectura interesante sobre que pasa si se te pasa cerrarlos

<https://stackoverflow.com/questions/8175827/what-happens-if-i-dont-call-fclose-in-a-c-program>

Paso 2a: 2. Se va a leer ese fichero, mostrar su contenido

Para leer de un fichero binario, debemos abrir el manejador de ficheros, igual que en el paso anterior, pero ahora con la opción de sólo lectura:

```
FILE * pfile = fopen(nfichero, "rb");
```

Por extender la solución, asumimos que desconocemos el número de elementos que hay en el fichero, aunque sabemos que son *unsigned*. Para leer de un fichero binario, hacemos uso de la función `fread`, dando:

- un puntero a una zona de memoria para almacenar lo que leamos
- el tamaño de lo que queremos leer
- el número de unidades a leer
- y el puntero al descriptor de fichero del que vamos a leer.

En nuestro caso, queremos leer una sola variable *unsigned* (e.j. basura):

```
fread(&basura, sizeof(unsigned), 1, pfile);
```

Debemos entrar en un bucle y leer mientras el fichero tenga cosas, para preguntar cuando hemos llegado al final del fichero (end-of-file), usamos la siguiente función:

```
!feof(pfile)
```

Usa printf para mostrar el contenido...

Paso2b: y se va a cargar en una estructura de árbol binario de búsqueda (una lista enlazada), eliminando repeticiones y estableciendo el orden.

Del paso anterior, ya tienes la carga de número desde fichero, ahora vamos a aprovechar, para en vez de mostrar, insertar en el árbol binario de búsqueda. Para ello, primero hay que completar la librería arbolbb.

Esta tiene el siguiente tipo de datos:

```
typedef struct T_Nodo* T_Arbol;

struct T_Nodo {
    unsigned dato;
    T_Arbol izq, der;
};
```

Observa como ahora, desde un T_Nodo salen dos punteros, uno a la derecha y otro a la izquierda.

Esta librería tiene los siguientes métodos que debes implementar:

```
// Crea la estructura utilizada para gestionar la
memoria disponible.
void Crear(T_Arbol* arbol);

// Destruye la estructura utilizada.
void Destruir(T_Arbol* arbol);
```

```
// Inserta num en el arbol, no se permiten dos
iguales (revisa el concepto de árbol binario de
búsqueda)
void Insertar(T_Arbol* arbol,unsigned num);

// Muestra el contenido del árbol en InOrden
void Mostrar(T_Arbol arbol);

// Guarda en disco el contenido del arbol en un
void Salvar(T_Arbol arbol, FILE* fichero);
```

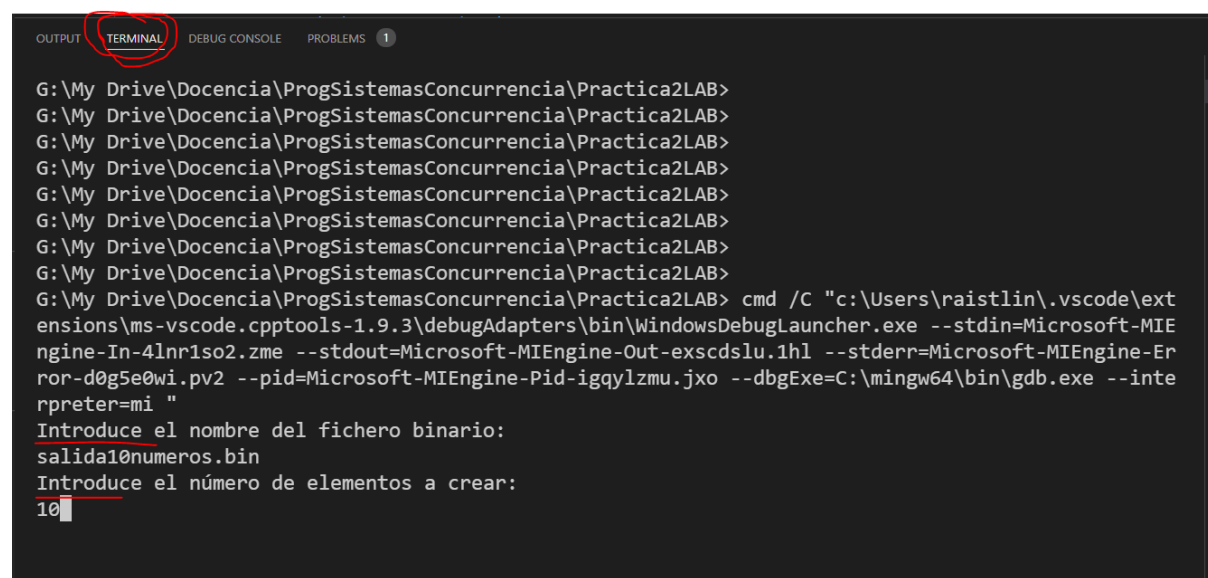
Todos, a excepción del Crear se deben implementar de forma recursiva.

Paso 3. Por último, se va a guardar de esta estructura a fichero. Para comprobar que funciona bien, se va a leer de nuevo el fichero y se va a mostrar por pantalla.

Este paso ya lo tienes implementado en el main, pasa y disfruta.

Ejemplo de salida e interacción:

En el terminal metemos los datos del nombre del fichero y el número de aleatorios a crear:



```

G:\My Drive\Docencia\ProgSistemasConcurrencia\Practica2LAB>
G:\My Drive\Docencia\ProgSistemasConcurrencia\Practica2LAB>
G:\My Drive\Docencia\ProgSistemasConcurrencia\Practica2LAB>
G:\My Drive\Docencia\ProgSistemasConcurrencia\Practica2LAB>
G:\My Drive\Docencia\ProgSistemasConcurrencia\Practica2LAB>
G:\My Drive\Docencia\ProgSistemasConcurrencia\Practica2LAB>
G:\My Drive\Docencia\ProgSistemasConcurrencia\Practica2LAB>
G:\My Drive\Docencia\ProgSistemasConcurrencia\Practica2LAB>
G:\My Drive\Docencia\ProgSistemasConcurrencia\Practica2LAB> cmd /C "c:\Users\raistlin\.vscode\ext
ensions\ms-vscode.cpptools-1.9.3\debugAdapters\bin\WindowsDebugLauncher.exe --stdin=Microsoft-MIE
ngine-In-4lnr1so2.zme --stdout=Microsoft-MIEngine-Out-exscdslu.1hl --stderr=Microsoft-MIEngine-Er
ror-d0g5e0wi.pv2 --pid=Microsoft-MIEngine-Pid-igqylzmu.jxo --dbgExe=C:\mingw64\bin\gdb.exe --inte
rpreter=mi "
Introduce el nombre del fichero binario:
salida10numeros.bin
Introduce el número de elementos a crear:
10

```

Salida (ojo que los números cambian en cada ejecución):

```
Introduce el nombre del fichero binario:  
salida10numeros.bin  
Introduce el número de elementos a crear:  
10
```

```
Ahora lo leemos y mostramos  
6 9 7 3 0 8 2 8 8 2 2  
Ahora lo cargamos en el arbol
```

```
Y lo mostramos ordenado  
0 2 3 6 7 8 9  
Ahora lo guardamos ordenado
```

```
Y lo mostramos ordenado  
0 2 3 6 7 8 9 9
```