Ejercicios propuestos

Se proponen dos ejercicios, en la línea de los presentados ya resueltos y propuestos en la Lección 4, pero esta vez para realizar entrega en una tarea de Studium.

Igual que en los anteriores ejercicios, se trata de programas independientes y todo el código de creación, gestión y liberación de los nodos de la lista enlazada se realiza en la función `main()` del programa.

Esta semana, del 27 de abril, se abrirán dos tareas para que realicéis la entrega de ambos ejercicios. Solo hay que subir un fichero `.c` por cada ejercicio, conteniendo la función `main()` que resuelva el correspondiente problema propuesto. Las tareas se cerrarán **el jueves, 30 de abril, a las 23:55 horas**.

Ejercicio 1

Fusionar dos listas enlazadas manteniendo el orden ascendente

En este primer ejercicio se va a avanzar por etapas. En la primera etapa se van a crear dos listas enlazadas, de unos diez nodos cada una, que contienen números enteros aleatorios. La creación de ambas se hará de tal manera que, se crea un nuevo nodo, se carga su campo `info` con un número entero aleatorio, y se inserta en la lista enlazada en la posición que le corresponde siguiendo una ordenación ascendente de los valores almacenados, tal y como se puede apreciar en la Figura 1. Esta parte del ejercicio es sencilla porque solo hay que añadir, como pudisteis ver en la sesión de aula virtual del 22 de abril, un bucle `for`, de la forma adecuada, y correcta, a la operación explicada en la subsección 3.8 de la **Lección 4**.

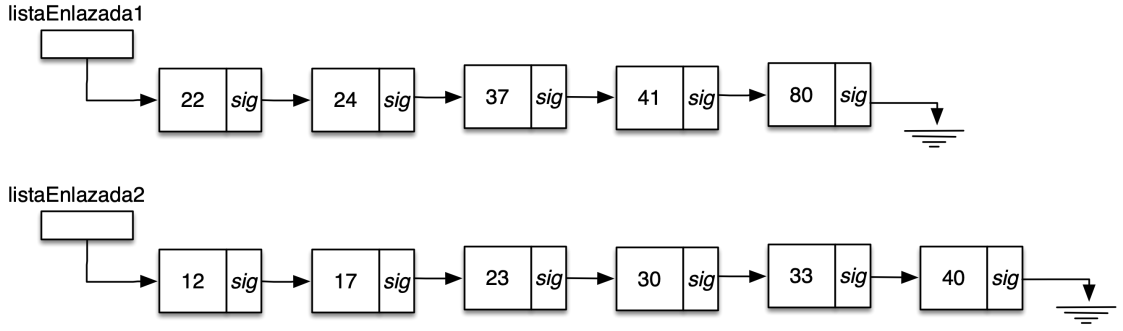

Figura 1: Dos listas enlazadas, con números enteros aleatorios, creadas de forma ordenada ascendente

Una vez están creadas las dos listas enlazadas, se pueden realizar recorridos secuenciales de ambas, para que muestren los valores añadidos en cada una de las listas.

A continuación, la segunda etapa del ejercicio: fusionar ambas listas en una única lista enlazada, conteniendo los nodos de ambas, manteniendo el orden ascendente de los valores. Para ello, se realizara el siguiente algoritmo:

- Se mantiene la lista enlazada resultante mediante un puntero, `listaResultado` en la Figura 2, que, inicialmente estará vacía (`NULL`).
- Se inicia un bucle que debe iterar mientras que **ambas** listas de entrada, `listaEnlazada1` y `listaEnlazada2` en la Figura 2, no estén vacías.
- En cada iteración del bucle indicado se compararán los campos `info` de los nodos raíz de ambas listas de entrada. El nodo, de los dos, que contenga el campo `info` **menor** quedará referenciado por una variable puntero auxiliar, `aux` en la Figura 2.

- A continuación se "puentea" ese nodo raíz referenciado por `aux`, es decir, se elimina de la lista enlazada a la que pertenece. Esta operación de "puenteo" se refleja en la en la Figura 2 mediante una línea punteada. Es conveniente terminar correctamente este nodo extraído, poniendo su campo `sig` a `NULL`.
- Se añade el nodo extraído, y referenciado por la variable `aux`, al final de la lista enlazada resultado, `listaResultado`, utilizando la variable `indice` para esta tarea. Obviamente, se debe controlar la situación inicial, cuando esta lista resultante está vacía. Nuevamente, estas operaciones de añadir al final se pueden observar en la Figura 2 mediante líneas punteadas.
- Por último, según se van extrayendo nodos de una y otra lista enlazada, se llega la situación en la que una de las dos listas se queda vacía. En ese instante debe finalizar el bucle, se debe comprobar cuál de las dos listas enlazadas de entrada aún contiene nodos, y se debe concatenar al final de la lista resultante. Esta situación es la que se puede apreciar en la Figura 3, donde la `listaEnlazada2` ha quedado vacía, así que se debe añadir, concatenar, la `listaEnlazada1` al final de la lista resultante. Importante, no es necesario ya ningún recorrido, simplemente se "engancha" el último nodo de la lista enlazada con el nodo raíz de `listaEnlazada1`. Nuevamente, la operación se sugiere mediante una línea punteada en la Figura 3.

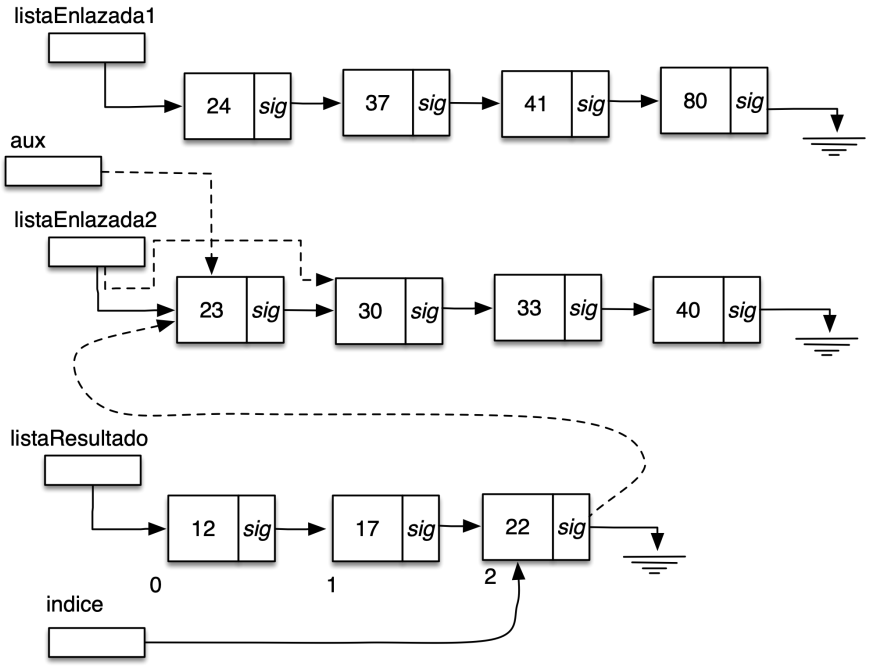

Figura 2: Fusión de os listas enlazadas en una única lista resultante, manteniendo el orden creciente

Obviamente, en las Figura 2 y Figura 3, no se reflejan todas las operaciones a realizar. El estudiante debe completarlas, de forma correcta, para conseguir el objetivo deseado, primero dibujando, apoyándose en el entregado en el enunciado, y, posteriormente, escribiendo un borrador del algoritmo. Una vez se tienen claros los pasos concretos a dar, se puede pasar a código C.

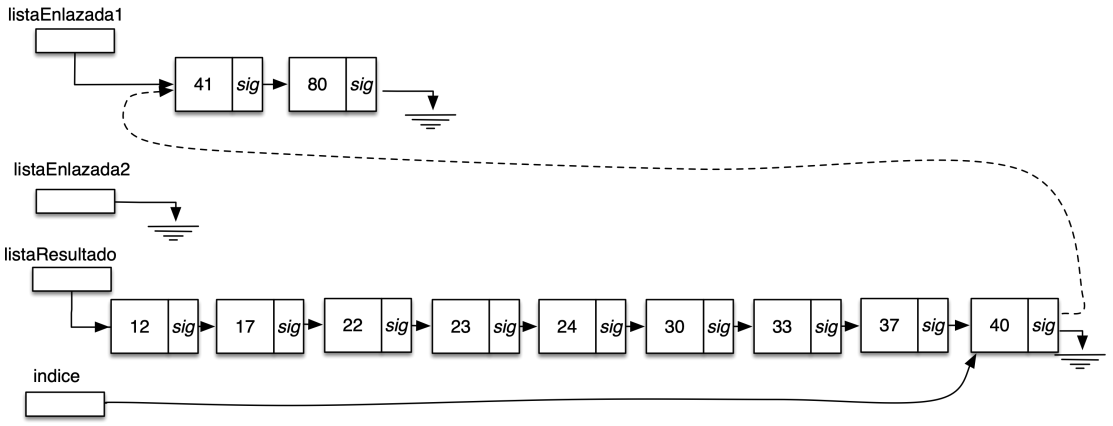

Figura 3: Finalización del bucle. La lista `listaEnlazada2` ha quedado vacía

El paso final será recorrer la lista enlazada resultante para mostrar el correcto funcionamiento del programa escrito. Y, por supuesto, no nos podemos olvidar de liberar todos los nodos antes de finalizar la ejecución del programa.

Ejercicio 2

Programa purga: eliminar o purgar todos los valores repetidos

En este ejercicio se creará una lista enlazada, nuevamente, con valores enteros aleatorios. Para asegurarnos de que van a aparecer números repetidos, lo que se hará es obtener los valores aleatorios mediante una expresión del tipo

```
rand() % 100
```

Si se crean un número elevado de nodos, por ejemplo, unos 50, van a aparecer varios números repetidos. En este ejercicio lo valores no van a seguir ningún tipo de ordenación, así que se puede seguir la estrategia que se desee en la creación: insertando cada nodo creado por el comienzo o añadiéndolo al final de la lista, tal y como se explica en la susbsección 3.3 de la **Lección 4**.

Una vez se ha creado la lista, se puede realizar un recorrido secuencial de la misma mostrando los valores por pantalla y, así, comprobar el resultado.

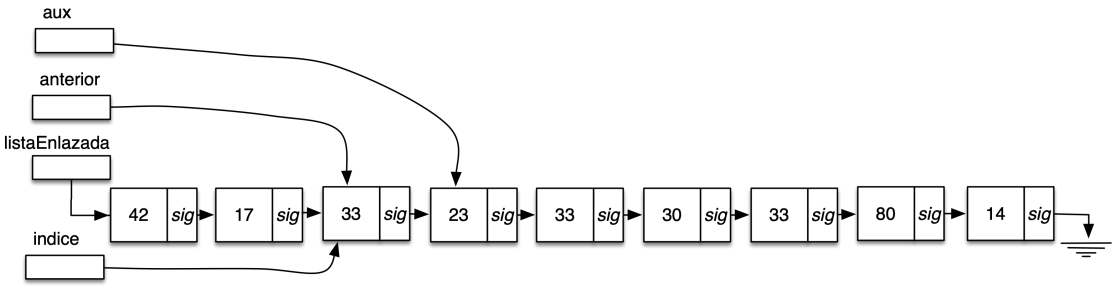

Figura 4: Inicio del recorrido del bucle más anidado

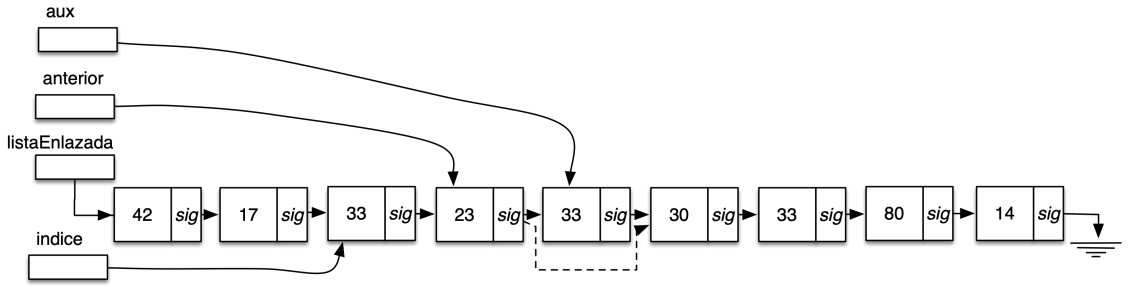

Figura 5: Se ha localizado un nodo con un valor repetido: se puentea y se libera la memoria reservada

El siguiente paso es el algoritmo encargado de purgar, eliminar los valores repetidos de la lista. Serán dos bucles anidados:

- El primer bucle, el más externo, comienza en el nodo raíz de la lista enlazada y la recorre secuencialmente. Para dicho recorrido se utilizará una variable puntero, `indice`, para referenciar, en cada iteración, el nodo que almacena el valor cuyas repeticiones se van a localizar y eliminar en los restantes nodos de la lista.
- Para realizar la búsqueda de repetidos, se inicia el segundo bucle que realizará el recorrido utilizando dos punteros, `aux`, que apunta al nodo que contiene el campo `info` que se comprueba si es repetido del valor almacenado en el nodo apuntado por `indice`, y `anterior`, que siempre apuntará al nodo anterior al referenciado por `aux`. Este recorrido se inicia en nodo siguiente al apuntado por `indice`, tal y como se puede ver en Figura 4.

- Cada vez que, en una iteración de este bucle, se comprueba que el valor almacenado en el nodo referenciado por `aux`, es un valor repetido del almacenado en el nodo apuntado por `indice`, se "puentea", tal y como se puede observar en línea punteada de Figura 5, y se libera la memoria. Es aquí donde se utiliza el puntero `anterior`, para poder modificar el campo `sig` del nodo anterior. Una vez eliminado el nodo es importante realizar la asignación oportuna del puntero `aux` para que se pueda continuar correctamente el recorrido del bucle.
- Una vez el bucle más anidado llega al final de la lista enlazada, se han eliminado todos los repetidos del valor almacenado en el nodo apuntado por `indice`, se avanza con dicho puntero al siguiente nodo en el bucle más externo, de manera que pase a apuntar el siguiente nodo y se vuelve a repetir la misma operación, ya descrita, de localización y purga de repetidos para dicho nodo.

El paso final será recorrer la lista enlazada resultante para mostrar el correcto funcionamiento del programa escrito. Y, por supuesto, no nos podemos olvidar de liberar todos los nodos antes de finalizar la ejecución del programa.