***Sobre las listas enlazadas***

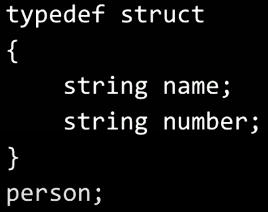
El reemplazo eficiente de las “matrices” para tomar datos o información de otra matriz y, ocupar su copia en otro espacio de memoria dinámica, es una estructura de datos llamada: ***“lista vinculada o enlazada”***; pues, efectivamente, hace el trabajo de forma más eficiente: sin tener que tocar todos los datos de la matriz original.

En principio, si se reemplaza a la matriz original por una lista vinculada; tendremos que, los valores o datos de eso que, en principio era una matriz, ya no se encontrarán juntos de forma contigua (como si lo hace la matriz con sus elementos), no; sino que, dichos valores, los de una lista vinculada, están regados por toda la memoria dinámica que se encuentre disponible, y no juntos del uno al otro; pero, si entonces este sería nuestro caso parcial, ya no sería tan fácil encontrar esos valores puntuales: puede que sepas dónde está uno, pero ya no basta con mirar una ubicación a la derecha para encontrar el siguiente valor. Resulta que ahora, entonces, lo que tienes que hacer para conectar esos valores, de una misma y potencial *“lista enlazada”*, pero ubicados de forma independiente en toda la memoria dinámica accesible… *sería declarar cada uno de esos valores con un espacio de memoria extra: exactamente el doble de espacio de memoria del que realmente requieren.*

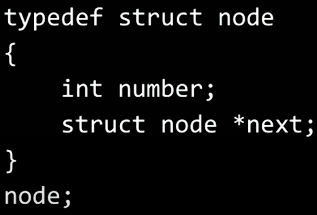
Si por ejemplo estamos trabajando con un número, sería entonces declarar ese valor numérico con un espacio de memoria como si fueran dos en vez de uno; y, ya con esto, me ayudaría para darle un seguimiento real a los valores de mi interés, los que quiero que conformen mi "lista enlazada", al tener algo en común que los relacione: ese espacio de memoria extra. En ese espacio de memoria extra, por ejemplo de mi valor o elemento 1, realmente se ocuparía un puntero que *apunte* hacia la ubicación del valor o elemento posterior a este primero, el valor que le sigue a “1” y que me interesa; es decir, hacia el valor numérico “2”, si vengo del “1” ***(creando una lista vinculada de tamaño 2 parcialmente).*** Así sucesivamente con los demás valores.

Desde luego entonces, ahora sí de manera definitiva, se conforma lo que se llama como: “Lista vinculada” (que se compone por un valor, elemento o variable, propiamente de interés: el que quiere ser declarado; más, la declaración de un espacio de memoria extra, que se declara en C como ***“struct node"***: donde se sitúa el puntero que *apunta* a la ubicación del elemento o variable inmediatamente posterior al primero que ya declaramos:   
***\* “nombre de variable apuntada")***, entendiendo a ésta como un tipo de estructura de datos que nace a partir de una "colección de ***nodos”*** que están conectados mediante punteros.

Teniendo en cuenta que las listas enlazadas se desarrollan a partir de estructuras en C, recordemos su forma más básica tomando como ejemplo el caso del *“bookphone”.*



Ahora, para representar propiamente a las listas enlazadas, la estructura lógica sería más o menos así:



*Ahora bien, en principio para conectar todos estos nodos se requiere de una variable simple que hará de puntero; y que tiene la función, justamente, de apuntar hacia la dirección donde comienza la lista vinculada con el nodo real número 1.*

Cada ***nodo*** es situado en un espacio de memoria dinámico asignado por la función ***malloc;*** quiere decir entonces que, efectivamente, los nodos pueden estar situados en cualquier parte, aleatoriamente, de todo el espacio de memoria dinámico disponible; y, son conectados, unos a otro, precisamente haciendo uso de punteros. Punteros que, como bien sabe, son asignados en cada nodo y que acompañan al valor, variable o elemento de interés definido en las estructuras.

La ventaja de este enfoque, en comparación a las matrices por ejemplo (que entre otras cosas resultan muy conveniente para la búsqueda binaria, cosa de la que las listas enlazadas no pueden alardear; pues, evidentemente sus datos no están organizados o enumerados según un patrón de orden definido; sino, regados de forma aleatoria por toda la memoria dinámica: lo dicho), es que ya no tienes que seguir asignando y copiando más memoria (y todos sus valores) cada vez que desee cambiar el tamaño de, por ejemplo, una matriz (suponiéndose que no estamos trabajando con listas enlazadas para intentar agrupar elementos en común, sino con matrices).

*Trabajar las matrices de esta forma, a la larga, y especialmente en proyectos muy grandes, resultará muy agobiante; pues, te va a matar, en términos de rendimiento, tener que copiar todos tus datos en común, de un lugar a otro, cada vez que quieras incorporar uno nuevo. Cosa que se evita muy bien en las listas enlazadas; pues, sólo se ocupan de un espacio de memoria extra equivalente al espacio de memoria realmente requerido por el nuevo dato en común incorporado (esto, claro, más el espacio de memoria guardado al puntero).*

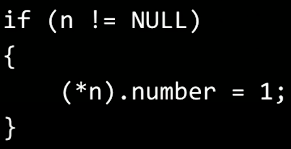
Si reflejamos todo esto en código, la ***variable simple e inicial*** que hace de puntero se podría definir así: nodo \****list*** = NULL; (donde así se declara, en principio, una lista enlazada), tal que así:



Ahora se llama a *malloc* para asignarle un espacio de memoria a nuestro primer nodo, tal que así:

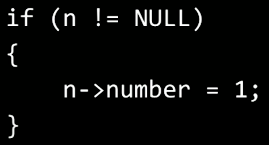


posteriormente, mientras *n* no sea igual a *NULL*, eso significa que *n* va a una dirección válida en algún lugar de la memoria; entonces, queremos que el programa automaticamente lo defina como, por ejemplo:

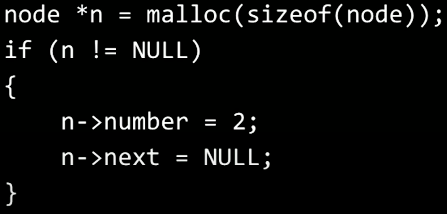


donde, *n* irá a la dirección donde está situada la variable, en este caso, *number* (donde se encuentra definida, continuando con el ejemplo inicial, nuestro primer valor: el número “1”, siendo el nodo real número 1 también).

Que es lo mismo que decir:



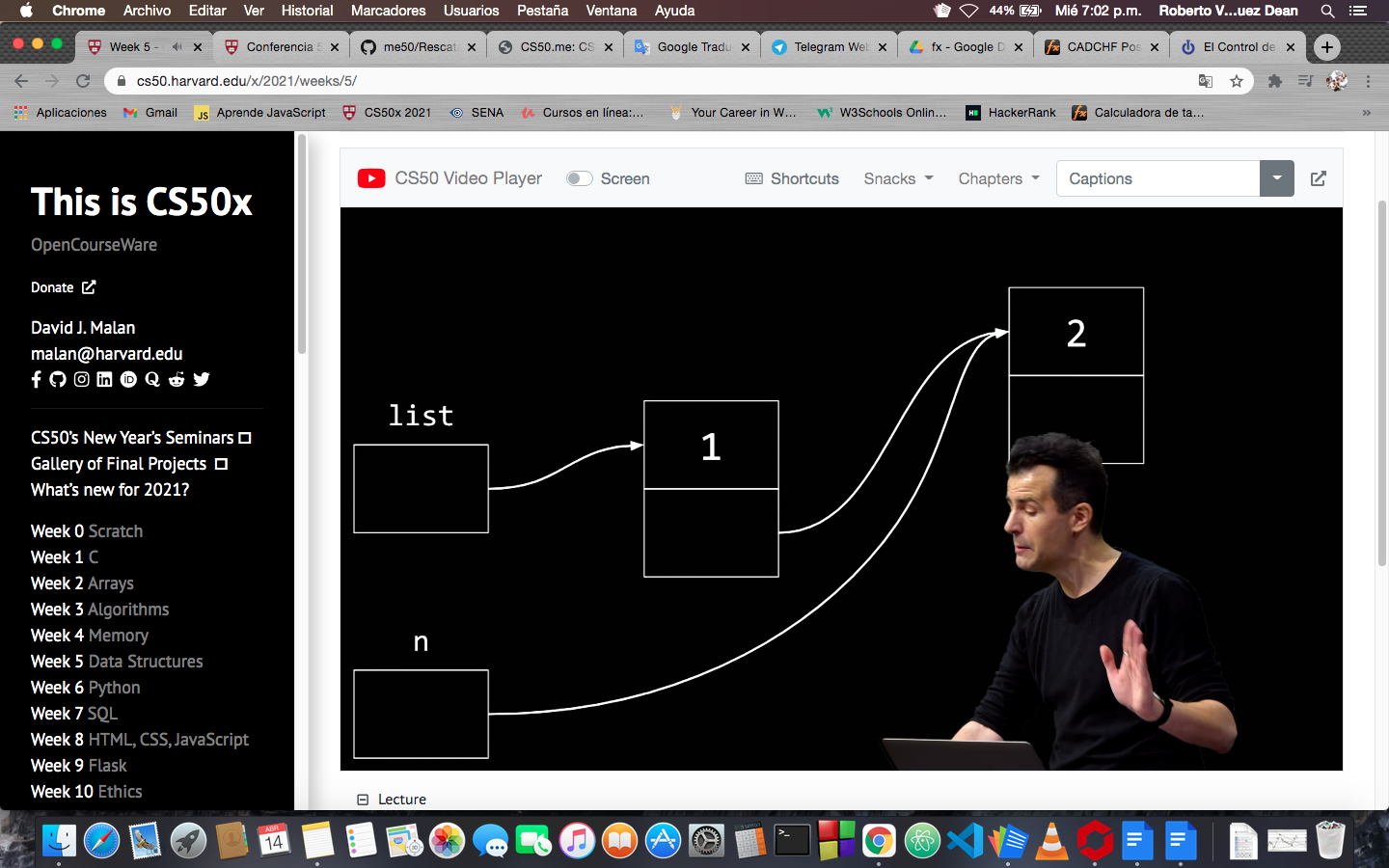
Ahora, para irse al segundo nodo, sería así:



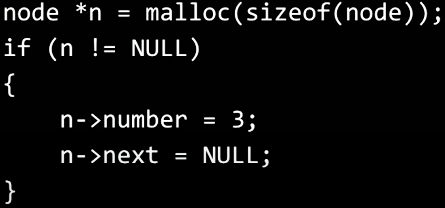
Sin embargo, si se fija bien, por la asignación de un segundo nodo no necesariamente, de forma instantaneamente, se vincula con el primero, no! De hecho, aún no están vinculados: el nodo 2 todavía no hace parte de la estructura de datos. A la lista vinculada que se quiere lograr aún le falta un puntero que conecte a ambos nodos (más precisamente, un puntero del nodo 1 que *apunte* a nodo 2), tal que así:

 ahora, ya con esto, tiene una lista vinculada de tamaño 2 (bajo este ejemplo).

*El proceso, generalizado, se vería más o menos así:*



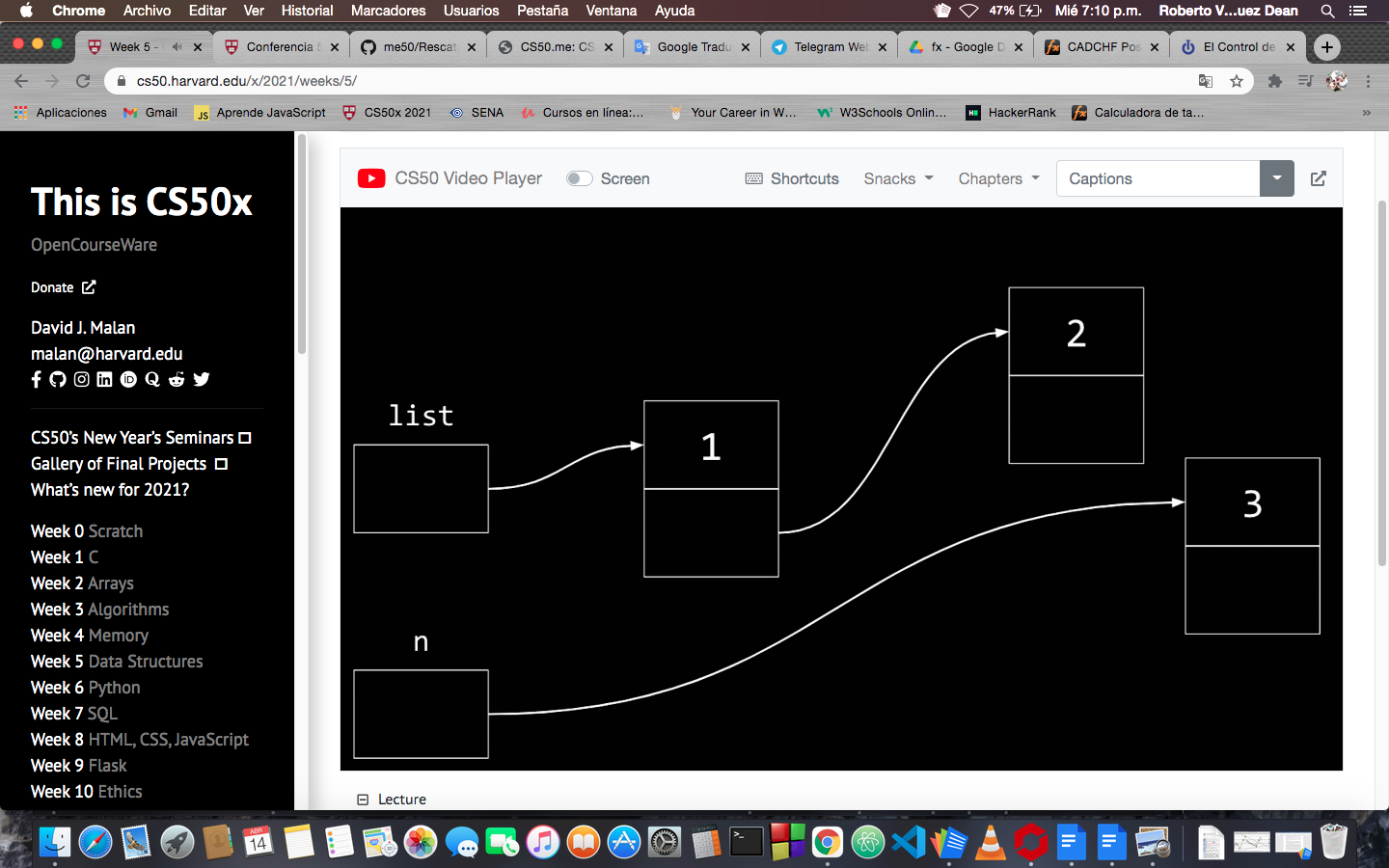
*y si por ejemplo quisieramos agregar un tercer nodo, sería así:*



***+***

 *(eventualmente, en un ejercicio real, con la inclusión de más nodos; sería tedioso tener que agregar, por cada nodo, estas “flechas”: por lo que hará uso de bucles en código).*

*El proceso, generalizado, se vería, ahora, más o menos así:*



***No se asuste, todos estos procesos serán simplificados en un ejemplo real en código (vealo en el archivo tipo c llamado: “list.c”). Y trataremos de ser lo menos tedioso posible.***