**REPRESENTACION DE UN PROGRAMA EN C EN MEMORIA**

Para desarrollar el trabajo asignado, primero voy a desarrollar gráficamente el concepto de listas enlazadas desde la página 1 hasta la pagina 2, después explicare el código paso a paso, de tal forma que se pueda entender la representación en memoria del programa, todo esto desde la página 2 hasta la página 7.

Finalmente se explica conceptos que se pueden aplicar a cualquier programa con estructuras y listas enlazadas, por ende, realizar la representación de un programa en c en memoria

**Listas enlazadas**

Las listas enlazadas permiten representar un grupo de elementos presentados como una secuencia.

Dentro de la lista enlazada podemos guardar un grupo de elementos que sean del mismo tipo (enteros, cadenas de caracteres). La tarea a realizar es embutir un objeto dentro de una estructura llamada nodo:

nodo

16

24

32

Podemos decir que tenemos 3 nodos en la ilustración anterior. Los nodos tienen punteros que les permiten apuntar a otros nodos.

En la siguiente ilustración el nodo 32 está apuntando al 24 y el 24 está apuntando al 16

16

24

32

La siguiente secuencia de nodos apuntados entre sí, es decir una lista, tenemos un nodo principal que es llamado cabeza; es el único que tenemos que conocer para poder trabajar con una lista. Para poder acceder al resto de nodos, tenemos que acceder al puntero siguiente de cada uno de los nodos

cabeza cola

24

72

19

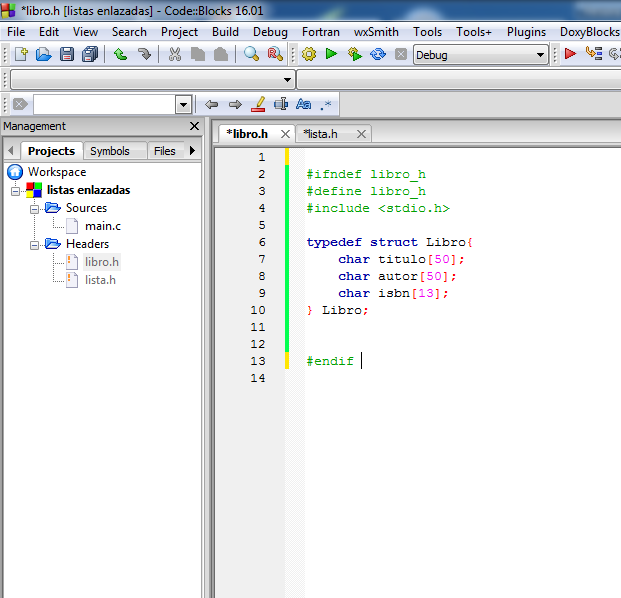
21

32

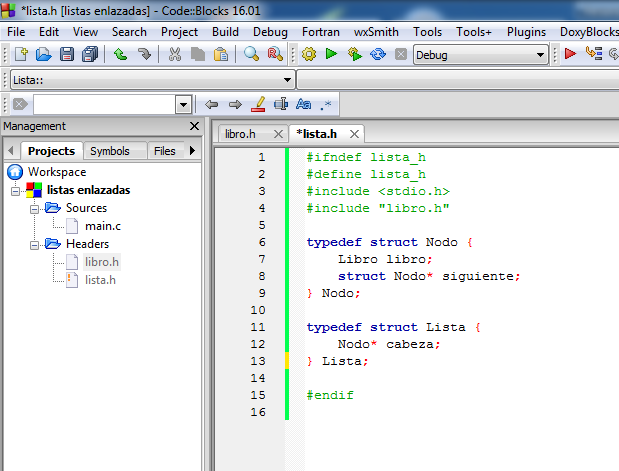
Si accedo al nodo siguiente de la cabeza, accedemos al nodo 21 y si accedemos al siguiente, siguiente de la cabeza, accederemos al 19 y así sucesivamente. Lo que no es cabeza se denomina cola

**Programa en C**

Para hacer una lista enlazada, para ello crearemos una estructura llamada Libro para guardar algunos datos y construiremos una lista para poder guardar elementos, insertar elementos, eliminar elementos y obtener información general de elementos de la lista.



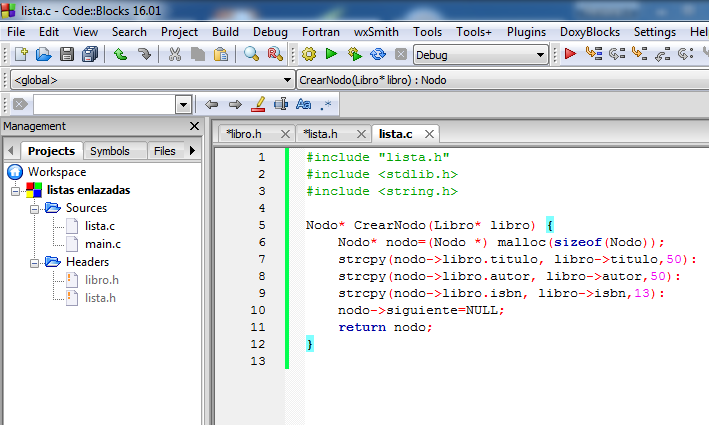
Vamos a definir las estructuras que vamos a utilizar, vamos a usar una estructura llamada Nodo, que va a guardar un libro, también va a guardar un puntero al siguiente elemento, de este modo podemos acceder a los siguientes elementos de un nodo usando la variable siguiente, como si fuese una cabeza y una cola de una lista, incluiremos el archivo libro (libro.h), Entonces creamos la estructura llamada Lista para guardar el puntero a la cabeza de esa lista, la ventaja de usar lista es que se puede mantener más elementos dentro de la estructura



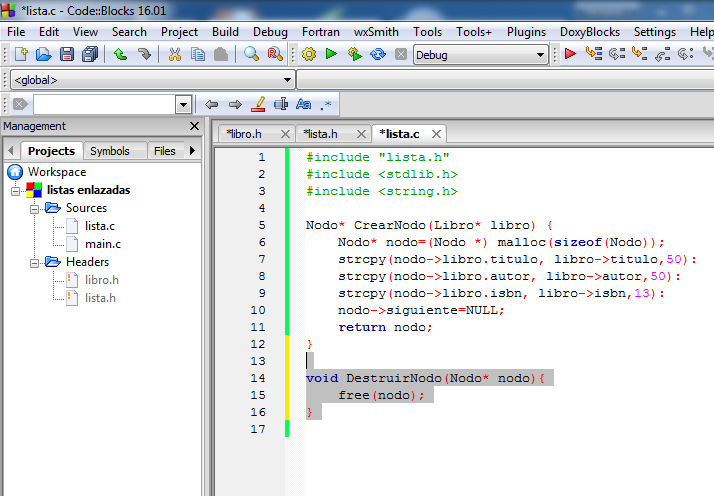
En cuanto a la implementación lo que podemos hacer es por ejemplo crear un nuevo nodo, a partir de los datos de un libro. (en c si queremos usar memoria dinámica tenemos que usar stdlib.h para poder usar las funciones malloc y free)

Luego instanciamos un nuevo nodo usando la función malloc a la que le decimos el tamaño del nodo, luego usamos la función strncpy, vamos a copiar dentro del libro, los datos del libro que pasamos como parámetro.

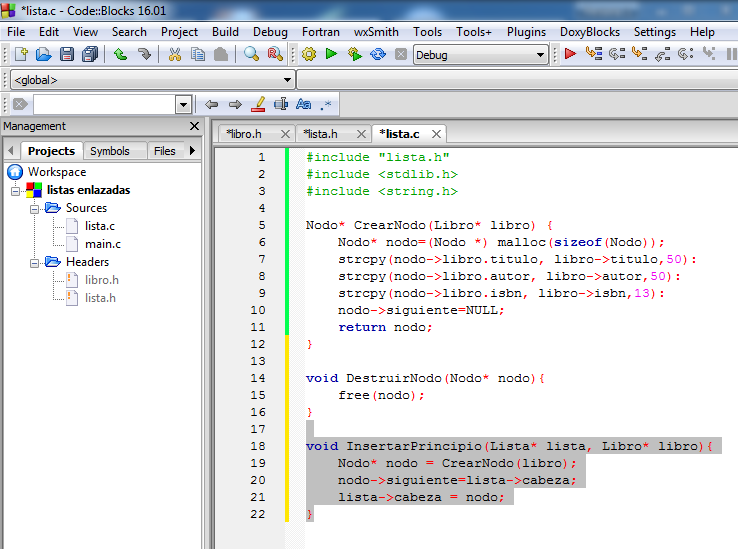
Como no sabemos a qué va apuntar siguiente, lo apuntaremos a NULL, para no tener punteros sueltos y finalmente lo devolvemos (return nodo;)



También tenemos que hacer una función DestruirNodo para destruir un nodo que ya no queremos mantener en memoria, esta función lo que tiene que hacer es un free

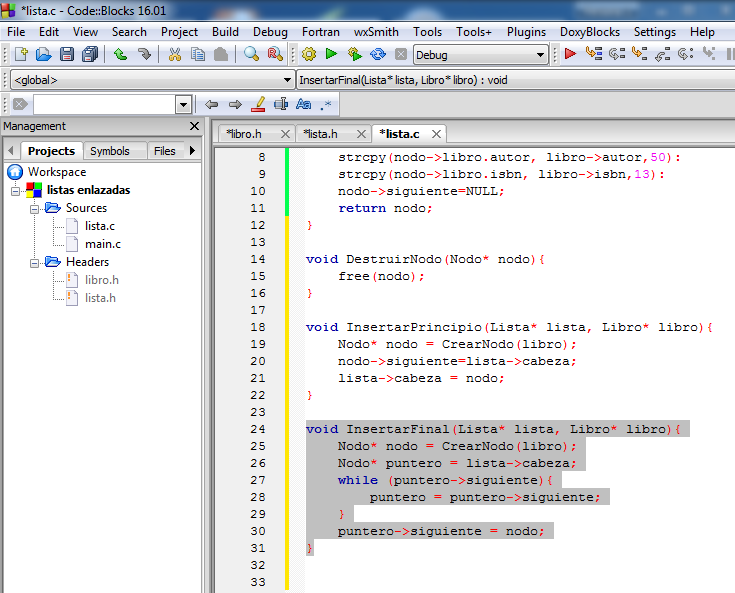


Una de las operaciones a seguir es insertar al principio de una lista un libro, esto lo hacemos creando una función tipo void llamada InsertarPrincipio, dentro de esta función lo primero que hacemos es crear un nuevo nodo (Nodo\* nodo = CrearNodo(libro);) a partir de los datos de nuestro libro, luego tenemos que poner como cola del nodo que acabamos de crear la actual cabeza de la lista (nodo->siguiente=lista->cabeza;), después movemos el puntero cabeza para que apunte al nuevo nodo y con todo esto ya lo tenemos insertado al principio.

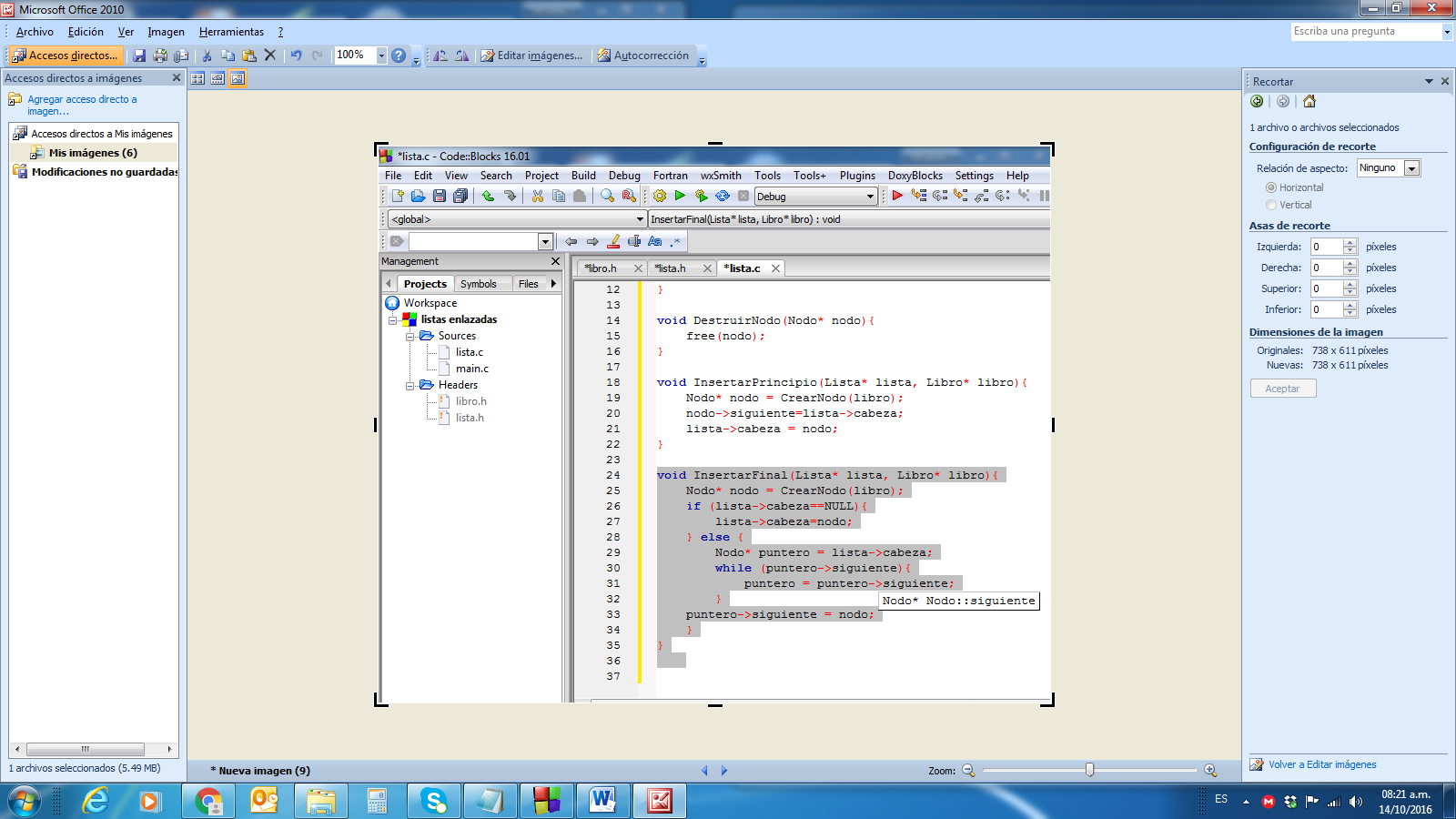


Ahora insertaremos un libro al final de la lista, para esto tenemos que recorrer la lista en busca del último elemento, para que su siguiente apunte al nodo recién creado, para esto creamos un nodo, luego mantenemos un nodo llamado puntero que inicialmente será la cabeza de la lista, ( Nodo\* puntero = lista->cabeza;) tenemos que hacer ese puntero vaya avanzando de una posición a otra hasta que haya alcanzado el final de la lista, sabremos esto porque su siguiente tendrá el valor NULL , así que tenemos que avanzar mientras el puntero tenga siguiente (while (puntero-> siguiente){), y luego dentro del while para avanzar el puntero escribimos puntero es igual puntero siguiente, (puntero = puntero-> siguiente;)

De este modo el puntero va avanzando una posición y cuando lleguemos al último como ya no tiene siguiente se sale del bucle y una vez que hayamos salido del bucle, puntero estará apuntando al último elemento, entonces hacemos que apunte al nodo recién creado



Cuando la lista este vacía, lo anterior no funcionará, entonces preguntaremos si la cabeza de la lista este vacía, (if (lista->cabeza ==NULL)) es decir no haya lista, entonces, si la cabeza de la lista esta vacía, la posicionamos como la cabeza de la lista (lista->cabeza=nodo;) y si la lista no está vacía tenemos que hacer que vaya avanzando (puntero=puntero->siguiente;)



**REPRESENTACION EN MEMORIA**

La creación y mantenimiento de estructuras dinámicas de datos, requiere de la asignación dinámica de memoria – la capacidad por parte de un programa de obtener, en tiempo de ejecución, más espacio de memoria para contener nuevos nodos, y de poder liberar espacio ya no requerido. El límite de la asignación dinámica de memoria puede ser tan grande como la totalidad de memoria física disponible en la computadora, o la cantidad virtual posible, en un sistema de memoria virtual. A menudo, los límites son mucho menores, porque la memoria disponible deberá ser compartida entre muchos usuarios.

Las funciones malloc y free y el operador sizeof, son esenciales a la asignación dinámica de memoria. La función malloc toma como argumento el número de bytes a asignarse, y regresa un apuntador del tipo void\* (apuntador a void) a la memoria asignada. Un apuntador void\* puede asignarse a una variable de cualquier tipo de apuntador. Normalmente la función malloc se utiliza conjuntamente con el operador sizeof.

NewPtr=malloc(sizeof(struc node));

Evalúa sizeof(struct node) para determinar el tamaño en bytes de una estructura del tipo struct node, asigna en memoria una nueva área de tamaño sizeof(struct node) bytes, y almacena en la variable newPtr un apuntador a la memoria asignada. Si no existe memoria disponible, malloc regresa un apuntador NULL.

La función free cancela la asignación de la memoria – es decir regresa la memoria al sistema, de tal forma que en el futuro esta pueda ser vuelta a asignar. Para liberar memoria asignada dinámicamente mediante una llamada malloc previa.

Free(newPtr);

Cuando se usan listas, pilas, colas de espera y árboles, cada una de estas estructuras de datos se crea y se mantiene utilizando la asignación dinámica de memoria y las estructuras autorreferenciadas.

El tamaño de una estructura no es necesariamente la suma de los tamaños de sus miembros. Esto es debido a varios requisitos de alineación de límites que son dependientes de la máquina.

Suponer que el tamaño de una estructura es simplemente la suma de los tamaños de sus miembros es un error. Para determinar el tamaño de una estructura se debe utilizar el operador sizeof.

Al utilizar malloc, compruebe si es NULL el valor de regreso de apuntador. Si la memoria solicitada no ha sido asignada imprima un mensaje de error.

No regresar memoria dinámicamente asignada cuando esta ya no es necesaria, puede hacer que el sistema se quede sin memoria prematuramente. Esto se conoce a veces como “fuga de memoria”.

Las listas enlazadas son dinámicas, por lo que conforme sea necesario la longitud de una lista puede aumentar o disminuir. Por su parte, el tamaño de un arreglo no puede ser modificado, porque la memoria del arreglo es asignada en tiempo de compilación. Los arreglos pueden llenarse. Las listas enlazadas solo se llenan cuando el sistema no tiene suficiente memoria para satisfacer las solicitudes de asignación dinámica de almacenamiento.

Podría declararse un arreglo que contenga más elementos que el número esperado de elementos de datos, pero esto puede desperdiciar memoria. En estas situaciones las listas enlazadas pueden obtener una mejor utilización de memoria.