Listas Enlazadas

Notas de estructura de datos con lenguaje C

Estructuras de datos dinámicas

Contrariamente a las estructuras de datos estáticas (*arreglos*-listas, vectores y tablas- y estructuras) en las que su tamaño en memoria se establece durante la ejecución del programa, las estructuras de datos dinámicas crecen y se contraen a medida que se ejecuta el programa.

Notas de estructura de datos con lenguaje C

- Estructuras lineales de elementos homogéneas (listas, tablas, vectores) y se utilizaban *arreglos* para implementar estas estructuras (de tamaño fijo y predefinido el espacio a ocupar en memoria).
- De modo que cuando se desea añadir un nuevo elemento que rebase el tamaño prefijado del *arreglo*, no es posible realizar la operación sin que se produzca un error un tiempo de ejecución.
- Se debe a que los arreglos hacen un uso ineficiente de la memoria.

Notas de estructura de datos con lenguaje C

Gracias a la asignación dinámica de variables, se pueden implementar listas de modo que la memoria física utilizada se corresponda con el número de elementos de la tabla.

Para ello se recurre a los *apuntadores (***punteros**) que hacen un uso más eficiente de memoria

Una **lista enlazada** es una colección o secuencia de elementos dispuestos uno detrás de otro, en la que cada elemento se conecta al siguiente elemento por un <<enlace>> o <<apuntador>>.

Notas de estructura de datos con lenguaje C

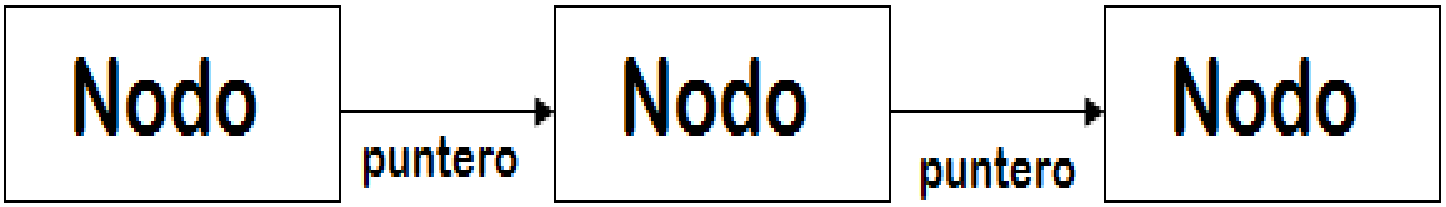
La idea básica consiste en construir una lista cuyos elementos llamados **nodos** se componen de dos partes o *campos*: la primera parte o campo contiene la información y es, por consiguiente, un valor de un tipo genérico (denominados *Dato, Tipo Elemento, Info, etc)* y la segunda parte o *campo* es un puntero (denominado *enlace*) que apunta al siguiente elemento de la lista.

Notas de estructura de datos con lenguaje C

Consiste en construir una lista cuyos elementos llamados **nodos** se componen de dos partes o *campos*:

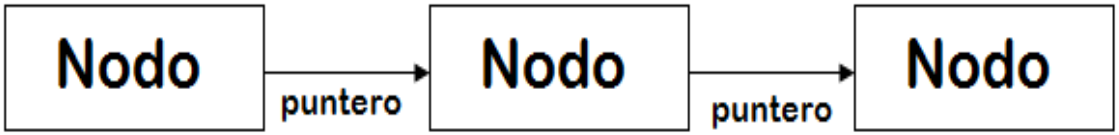
- La primera parte o campo contiene la información y es, por consiguiente, un valor de un tipo genérico (denominados *Dato, Tipo Elemento, Info, etc.).*
- La segunda parte o *campo* es un puntero (denominado *enlace*) que apunta al siguiente elemento de la lista.

Notas de estructura de datos con lenguaje C



Notas de estructura de datos con lenguaje C

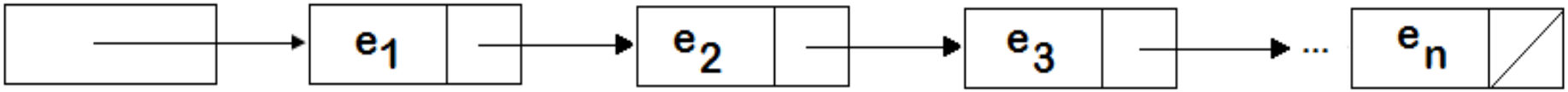
Lista enlazada *(representación simple)*



- La representación gráfica más extendida es aquella que utiliza una caja (un rectángulo) con dos secciones en su interior.

Notas de estructura de datos con lenguaje C

En la primera sección se escribe el elemento o valor del dato y en la segunda sección, el enlace o apuntador mediante una flecha que sale de la caja y apunta al nodo siguiente.



$e_1, e_2, e_3, \ldots e_n$ son valores del tipo TipoElemento

Lista enlazada (representación gráfica típica)

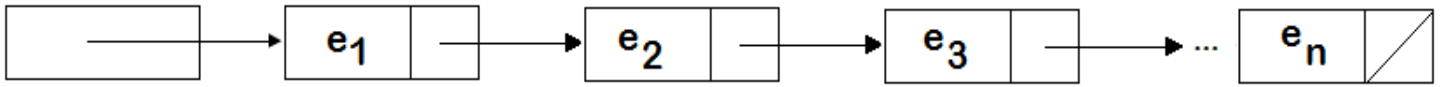
Notas de estructura de datos con lenguaje C

Nota: una **lista enlazada** consta de un número de elementos y cada elemento tiene dos componentes (campos), un puntero al siguiente elemento de la lista y un valor, que puede ser de cualquier tipo.

- Los enlaces representan por flechas para facilitar la comprensión de la conexión entre dos nodos; ello indica que el enlace tiene la dirección en memoria del siguiente nodo.

Notas de estructura de datos con lenguaje C

- Los enlaces también sitúan los nodos en una secuencia.



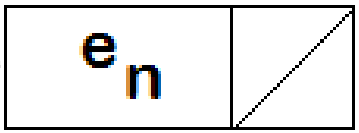
$e_1, e_2, e_3, \ldots e_n$ son valores del tipo TipoElemento

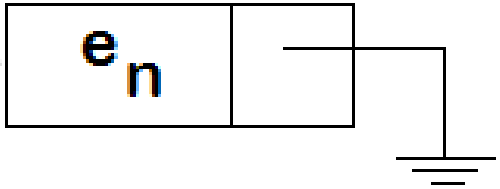
Los nodos forman una secuencia desde el primer elemento (e_1) al último elemento (e_n).

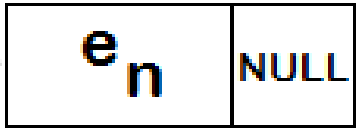
El primer nodo se enlaza al segundo nodo, el segundo nodo se enlaza al tercer y así sucesivamente hasta llegar al último nodo. El nodo último ha de ser representado de forma diferente para significar que este nodo no se enlaza a ningún otro.

Notas de estructura de datos con lenguaje C

Diferentes representaciones gráficas del último nodo:







Notas de estructura de datos con lenguaje C

Las listas se pueden dividir en cuatro categorías:

Listas simplemente enlazadas. Cada nodo (elemento) contiene un único enlace que conecta ese nodo al nodo siguiente o nodo sucesor. La lista es eficiente en recorridos directos (<<adelante>>).

Listas doblemente enlazadas. Cada nodo contiene dos enlaces, uno a su nodo predecesor y el otro a su nodo sucesor. La lista es eficiente tanto en recorrido directos(<<adelante>>) como en recorrido inverso (<<atrás>>).

Notas de estructura de datos con lenguaje C

Lista circular simplemente enlazada.

Una lista enlazada simplemente es la que el último elemento (cola) se enlaza al primer elemento (cabeza) de tal modo que la lista puede ser recorrida de modo circular (<<en anillo>>).

Lista circular doblemente enlazada.

Una lista doblemente enlazada es la que el último elemento se enlaza al primer elemento y viceversa. Esta lista se puede recorrer de modo circular (en anillo) tanto en dirección (<<adelante>>) como viceversa (<<atrás>>).

Notas de estructura de datos con lenguaje C

Para la construcción de las listas se puede elegir una implementación basada en arreglos o basada en apuntadores.

Como ya se ha comentado estas implementaciones difieren en el modo en que asigna la memoria para los datos de los elementos, cómo se enlazan juntos los elementos y cómo se accede a dichos elementos. De forma más específica, las implementaciones pueden hacerse con:

- *Asignación fija o estática, de memoria* mediante *arreglos*.
- *Asignación dinámica de memoria* mediante apuntadores.

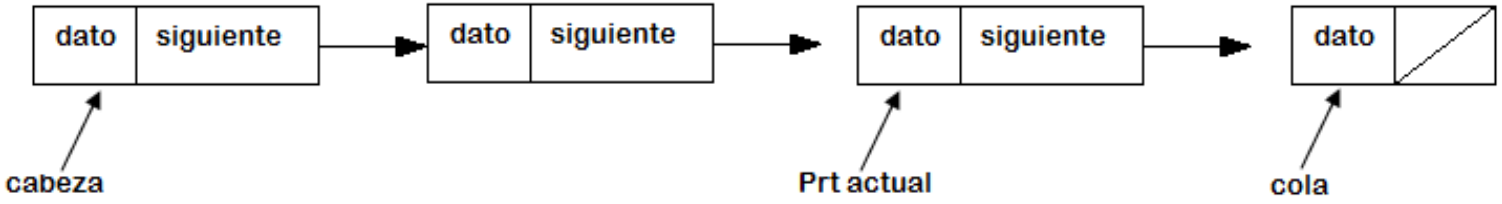
Es importante comentar que la asignación fija de memoria mediante *arreglos* es más ineficiente.

Por lo que solo desarrollaremos -la asignación de memoria mediante apuntadores.

Notas de estructura de datos con lenguaje C

Conceptos de listas

- Una lista enlazada consta de un conjunto de nodos.
- Un **nodo** consta de un campo dato y un puntero que apunta al <<siguiente>> elemento de la lista.

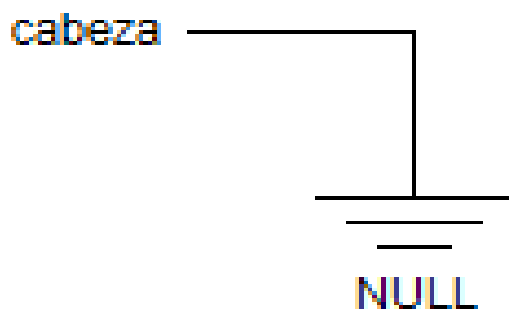


- El primer nodo, **frente**, es el nodo apuntador por **cabeza**. La lista encadenada nodos juntos desde el frente al final (**cola**) de la lista. El final se identifica como el nodo cuyo campo apuntador tiene el valor NULL = 0.

Notas de estructura de datos con lenguaje C

La lista se recorre desde el primero al último nodo; en cualquier punto del recorrido la posición actual se referencia por el apuntador Ptr_actual.

En el caso que la lista está vacía el apuntador cabeza es nulo.



Notas de estructura de datos con lenguaje C

OPERACIONES EN LISTAS ENLAZADAS

Las lista enlazada requiere controles para la gestión de los elementos contenidos en ellas.

Estos controles se expresan en forma de operaciones que tendrán las siguientes funciones:

- *Declaración de los tipos nodo y apuntador a nodo.*
- *Inicialización o creación.*
- *Insertar elementos en una lista.*
- *Buscar elementos de una lista* (comprobar la existencia de elementos en una lista).
- *Recorrer una lista enlazada* (visitar cada nodo de la lista).
- *Comprobar si la lista está vacía.*

Notas de estructura de datos con lenguaje C

Como se declara de un nodo:

Una lista enlazada se compone de una serie de nodos ***enlazados mediante apuntadores***.

Cada nodo es una combinación de dos partes: **un tipo de dato** (entero, real, doble, carácter o tipo predefinido) y **un enlace** (apuntador) al siguiente nodo.

En C se puede declarar un nuevo tipo de dato para un nodo mediante las palabras reservadas *struct* que contiene las dos partes.

Notas de estructura de datos con lenguaje C

```
struct Nodo
{
int dato;
    struct Nodo* enlace;
};
```

La declaración utiliza el tipo struct que permite agrupar campos de diferentes tipos, el campo dato y el campo enlace.

```
typedef struct Nodo
{
    Int dato;
struct Nodo *enlace
}NODO;
```

Con typedef se puede declarar a la vez un nuevo identificador de struct Nodo, en este caso se ha elegido NODO.

Notas de estructura de datos con lenguaje C

Dado que los tipos de datos que se pueden incluir en una listas pueden ser de cualquier tipo (enteros, reales, caracteres o incluso cadenas).

Con el objeto de que el tipo de dato de cada nodo se pueda cambiar con facilidad.

Se utiliza la sentencia typedef para declarar el nombre de Elemento como un sinónimo del tipo de dato de cada campo.

Notas de estructura de datos con lenguaje C

El tipo Elemento se utiliza entonces dentro de la estructura nodo:

```
typedef double Elemento;
        struct nodo
        {
Elemento dato;
struct nodo *enlace;
};
```

Si se necesita cambiar el tipo de elemento en los nodos, sólo tendrá que cambiar la sentencia de declaración de tipos que afecta a Elemento.

Siempre que una función necesite referirse al tipo del dato del nodo, puede utilizar el nombre Elemento.

Ejemplo se declara un tipo denominado PUNTO que representa un punto en el plano con su coordenada x & y, se declara el tipo NODO con el campo dato del tipo PUNTO. Se define un apuntador a NODO.

```
#include <stdlib.h>
typedef struct punto
{
float x, y;
} PUNTO;
typedef struct Nodo
{
PUNTO dato;
Struct Nodo* enlace;
}NODO;
NODO* cabecera;
cabecera = NULL;
```

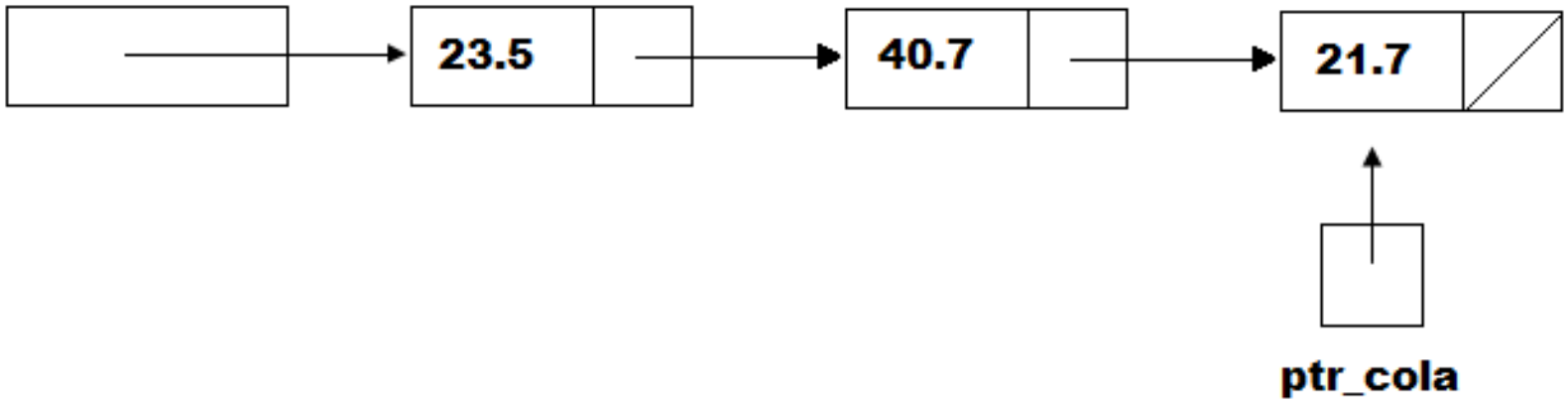
Notas de estructura de datos con lenguaje C

Apuntador al nodo inicial y al nodo final

- Los programas no declaran variables de nodos cuando se construye y emplea una lista enlazada, a la lista se accede a través de uno o más a*puntadores* a los nodos.
- El acceso más común a una lista enlazada es a través del primer nodo de la lista que se llama nodo inicial (**cabeza** o **cabecera)** de la lista.
- Un apuntador al primer nodo se llama, apuntador al inicial o **apuntador cabecera**.
- Nota: algunas veces se tiene un apuntador al último nodo de una lista enlazada.

Notas de estructura de datos con lenguaje C

- El último nodo es la final (**cola)** de la lista, y un apuntador al último nodo es el nodo final (a**puntador cola)**.
- También se pueden tener apuntadores a otros nodos de la lista enlazada.



Declaraciones de tipo en lista enlazada

Declaración del nodo

```
typedef double elemento;
struct nodo
{
   elemento dato;
   struct nodo *enlace;
};
```

Definición de apuntadores

```
struct nodo *ptr_cabeza;

struct nodo *ptr_cola;
```

Notas de estructura de datos con lenguaje C

Cada apuntador de acceso a la lista debe estar declarado como una variable apuntador.

Si se requiere una lista enlazada con un apuntador inicial (cabecera) y final (cola) es necesario declararlas variables apuntador.

Por ejemplo:

```
struct nodo *ptr_cabeza;
struct nodo *ptr_cola;
```

Notas de estructura de datos con lenguaje C

Nota: El tipo struct a veces se simplifica utilizando la declaración typedef.

Por ejemplo:

```
typedef struct nodo NODO;
typedef struct nodo* ptrnodo;
ptrnodo ptr_cabeza;
ptrnodo ptr_cola;
```

Notas de estructura de datos con lenguaje C

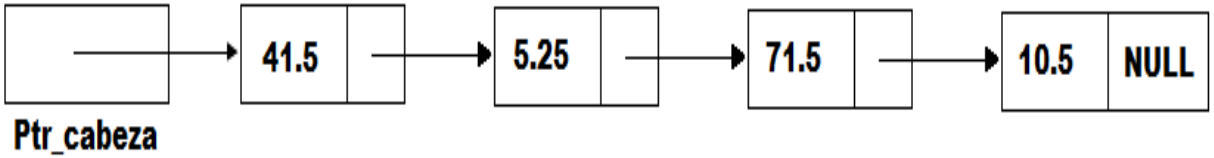
Notas importantes:

La construcción y manejo de una lista enlazada requiere el acceso a los nodos de la lista a través de uno o más apuntadores a nodos.

Normalmente, un programa incluye un apuntador al primer nodo (cabeza) y un apuntador al último nodo (cola).

El último nodo de la lista contiene un valor de 0, esto es, un apuntador nulo (NULL) que señala el final de la lista.

Notas de estructura de datos con lenguaje C



La palabra NULL representa el **apuntador nulo**, que es una constante especial de C.

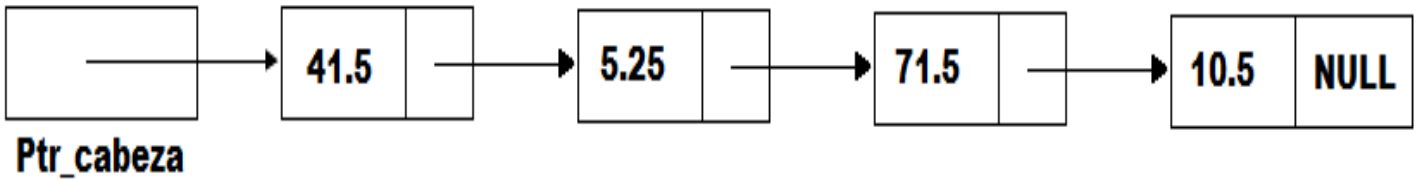
El apuntador nulo se utiliza en dos situaciones:

- Usar el apuntador en el campo enlace o siguiente nodo final de una lista enlazada.
- Cuando una lista enlazada no tiene ningún nodo, se utiliza el apuntador NULL como apuntador de inicial (cabeza) y de final (cola).
- A lista se denomina **lista vacía.**

Notas de estructura de datos con lenguaje C

El apuntador nulo se puede escribir como NULL, que es una constante de la **biblioteca estándar stdlib.h.**

El apuntador nulo se puede asignar a una variable apuntador con una sentencia de asignación ordinaria.



Notas de estructura de datos con lenguaje C

Nota importante:

El apuntador de inicial (cabeza) y de final (cola) es una lista enlazada puede ser NULL, lo que indicará que la lista es vacía (no tiene nodos).

Éste suele ser un método usual para construir una lista.

Cualquier función que se escriba para manipular listas debe manejar un apuntador de inicial (cabeza) y un puntero de final(cola) nulos.

Notas de estructura de datos con lenguaje C

El operador **->** de selección de un miembro:

- Si pato es un apuntador a una estructura y mono es un miembro de la estructura, entonces *pato* **->** *mono* accede al miembro *mono* de la estructura apuntada por *pato*.
- **”->”** se considera como un operador simple.
- Se denomina *operador de selección de miembro* o también *operador de selección de componente*
- operador pato ->mono recuerda a una flecha que apunta del apuntador pato al objeto que contiene al miembro mono.

Notas de estructura de datos con lenguaje C

Suponiendo que un programa ha de construir una lista enlazada y crear un apuntador de inicial (cabecera) ptr_cabeza a un nodo Nodo, el operador * de dirección aplicado a una variable apuntador representa **el contenido del nodo** apuntado por ptr_cabeza. Es decir, *ptr_cabeza es un tipo de dato Nodo.

- Al igual que con cualquier objeto se puede acceder a los dos miembros de *ptr_cabeza
- sentencia describe datos del nodo inicial (cabecera)

printf (“%f”, (*ptr_cabeza).dato);

(*ptr_cabeza) miembro dato del nodo apuntado por ptr_cabeza

Notas de estructura de datos con lenguaje C

Nota importante:

- Los paréntesis son necesarios en la primera parte de la expresión (*ptr_cabeza) ya que los operadores unitarios que aparecen a la derecha tienen prioridad más alta que los operadores unitarios que aparecen en el lado izquierdo (el asterisco de dirección).
- Sin los paréntesis, el significado de ptr_cabeza producirá un error de sintaxis, al intentar evaluar ptr_cabeza.dato antes de la dirección o referencia.

Notas de estructura de datos con lenguaje C

A recordar:

alumno -> matricula significa lo mismo que (*alumno).matricula.

Utilizando el operador de selección **->** se puede imprimir los datos del primer nodo de la lista.

```
printf("%lf", ptr_cabeza ->dato);
```

Notas de estructura de datos con lenguaje C

Error frecuente:

Los errores típicos en el tratamiento de apuntadores es escribir la expresión ***p** o bien **p->** cuando el valor del apuntador p es el apuntador nulo, ya que como se sabe el apuntador nulo no apunta a nada.

Notas de estructura de datos con lenguaje C

Para crear una lista:

Paso 1. Declarar el tipo de dato y el apuntador de inicial (cabeza) o primero.

Paso2. Asignar memoria para un elemento del tipo definido anteriormente utilizando alguna de las funciones se asignación de memoria (malloc ( ), calloc ( ), realloc( )) y un *cast* para la conversión de void* al tipo apuntador a nodo; la dirección del nuevo elemento es ptr_nuevo.

Paso 3. Crear iterativamente el primer elemento (cabeza) y los elementos sucesivos de una lista enlazada simplemente.

Paso 4. Repetir hasta que no haya más entrada para el elemento.

Notas de estructura de datos con lenguaje C

Ejemplo: desarrollar una lista enlazada de elementos que almacenen datos de tipo entero.

Un elemento de la lista se puede definir con la ayuda de la estructura siguiente:

```
struct Elemento  {
    int dato;
    struct Elemento * siguiente;
}; typedef struct Elemento Nodo;
```

En la estructura Elemento hay dos miembros, dato, que contiene el valor del elemento de la lista y siguiente que es un apuntador al siguiente nodo.

También se declara un nuevo tipo: Nodo que es sinónimo de struct Elemento.

Notas de estructura de datos con lenguaje C

- El siguiente paso para construir la lista es declarar la variable Primero que apuntará al primer elemento de la lista:

```
Nodo *Primero = NULL
```

El puntero Primero (también se puede llamar Cabeza) se ha inicializado a un valor nulo, lo que implica que la lista está vacía (no tiene elementos).

- Se crea un elemento de la lista, para ello hay que reservar memoria, tanta como tamaño tiene cada nodo, y asignar la dirección de la memoria reservada al apuntador Primero:

```
Primero = (Nodo*) malloc (sizeof(Nodo));
```

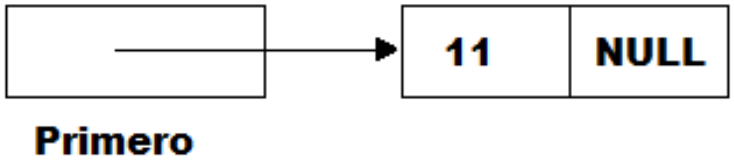
El operador sizeof se obtiene el tamaño de cada nodo de la lista, la función malloc( ) devuelve un apuntador genérico (void*), por lo que se convierte a Nodo*.

Notas de estructura de datos con lenguaje C

Se puede asignar un valor al campo dato:

Primero -> dato = 11;

Primero -> siguiente =NULL;



Apuntador Primero apunta al nuevo elemento, se inicializa a 11.

El campo siguiente del nuevo elemento toma el valor nulo, por no haber un nodo siguiente.

La operación de *crear un nodo* se puede hacer en una función a la que se pasas el valor del campo dato y del campo siguiente.

Notas de estructura de datos con lenguaje C

La función devuelve un apuntador al nodo creado:

```
Nodo* Crearnodo (int x, Nodo* enlace)
{
    Nodo *p;
    p = (Nodo*)malloc(sizeof (Nodo));
    p->dato = x;
    p-> siguiente = enlace;
    return p;
}
```

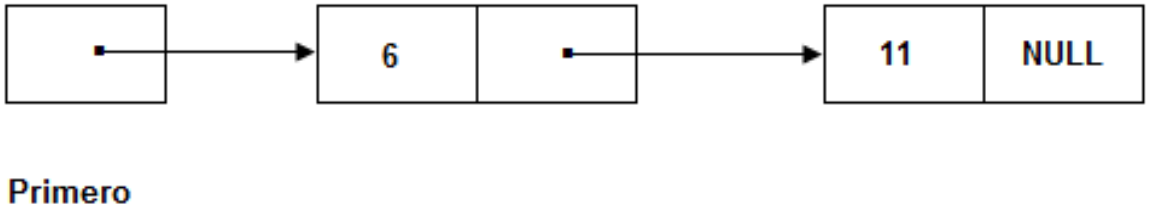
la función CrearNodo () para crear el primer nodo de la lista:

```
    Primero = Crearnodo (11, NULL);
```

Notas de estructura de datos con lenguaje C

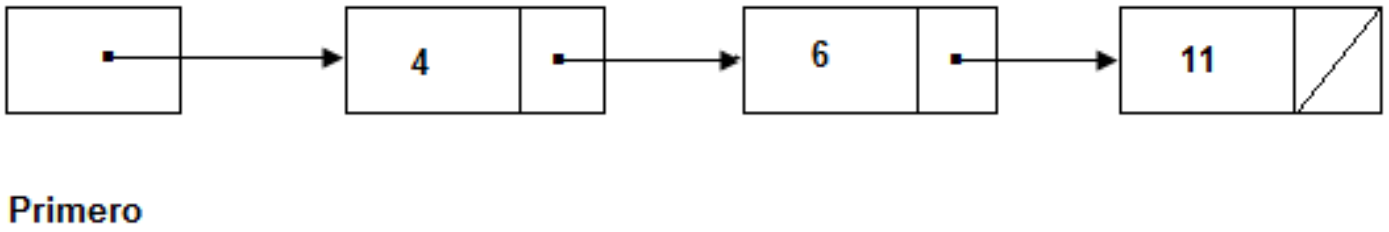
Para añadir un nuevo elemento con un valor 6, y agregarlo en el primer lugar de la lista:

Primero = Crearnodo (6, Primero);



Para una lista con los datos 4, 6, 11:

Primero = Crearnodo (4, Primero);



Notas de estructura de datos con lenguaje C

Insertar un elemento en una lista:

Para añadir o insertar un elemento en una lista enlazada el algoritmo varía dependiendo de la posición en que se insertar el elemento.

Inserción puede ser:

- Al principio de la lista (cabeza o elemento primero) de la lista.
- Al final de la lista (elemento último).
- Antes de un elemento especificado.
- Después de un elemento especificado.

Notas de estructura de datos con lenguaje C

Insertar un nuevo elemento en la cabeza de una lista:

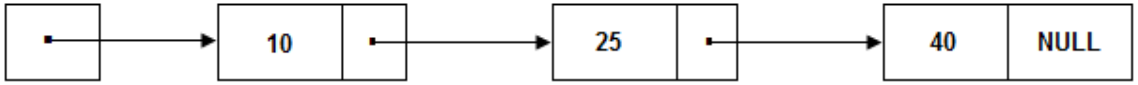
Es más fácil y más eficiente insertar un elemento nuevo al principio de la lista.

El proceso de inserción:

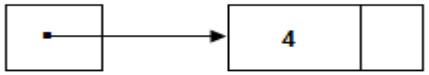
- Asignar un nuevo nodo apuntado por el nuevo que es una variable apuntador local que apunta al nuevo nodo que se va a insertar en la lista.
- Situar el nuevo elemento en el campo dato del nuevo nodo.
- Hacer que el campo enlace siguiente del nuevo nodo apunte a primer nodo (cabeza) de la lista original.
- Hacer que primer nodo (apuntador cabeza) apunte al nuevo nodo que se ha creado.

Notas de estructura de datos con lenguaje C

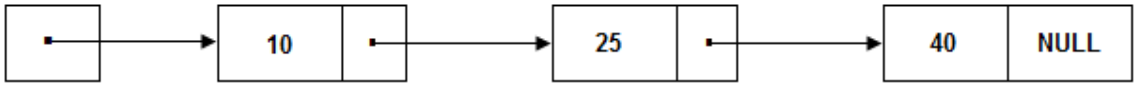
Ejemplo: se tiene una lista contiene tres elementos, 10, 25 y 40 se requiere insertar un nuevo elemento 4, al principio de la lista.



Paso 1 y paso 2



nuevo



cabeza

Notas de estructura de datos con lenguaje C

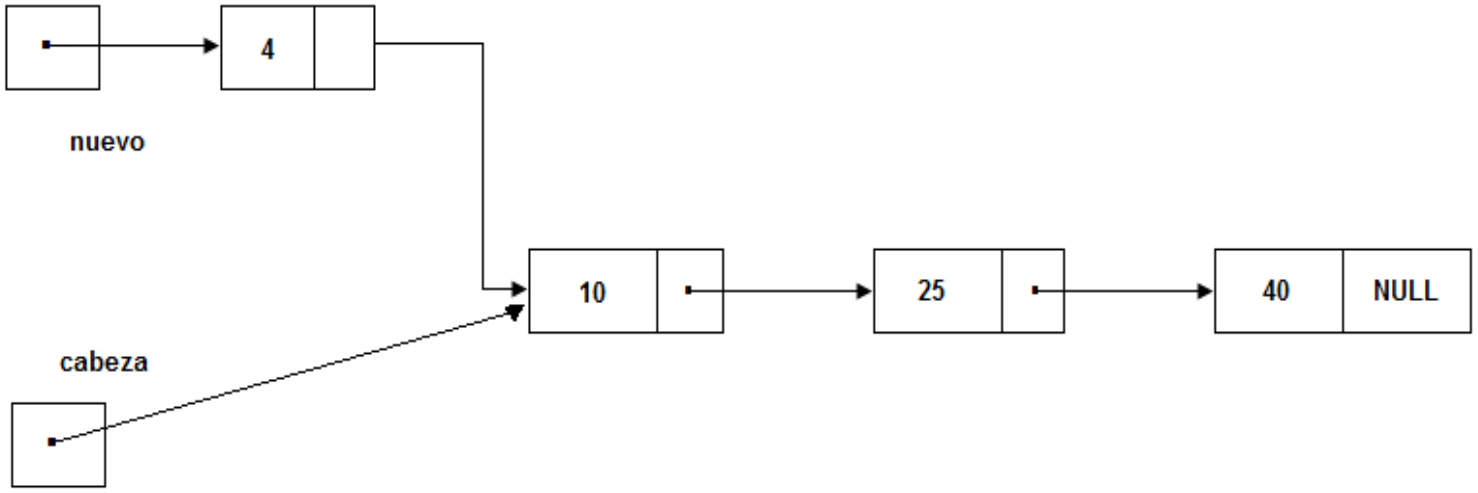
```
typedef int Item;
typedef struct tipo_nodo
{
    Item dato;
struct tipo_nodo* siguiente;
} Nodo; /*declaración del tipo Nodo*/
Nodo* nuevo;
nuevo = (Nodo*)malloc (sizeof (Nodo)); /*se asigna un nuevo
    nodo*/
nuevo-> dato = entrada;
```

Notas de estructura de datos con lenguaje C

Paso 3

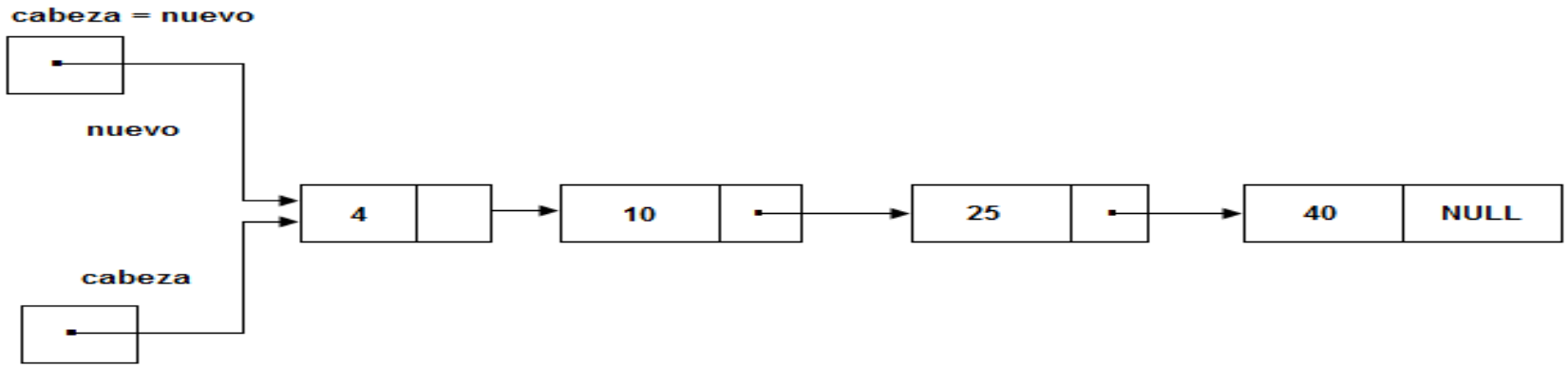
El campo enlace (siguiente) del nuevo nodo apunta a la cabeza actual de la lista.

nuevo -> siguiente = cabeza;



Paso 4

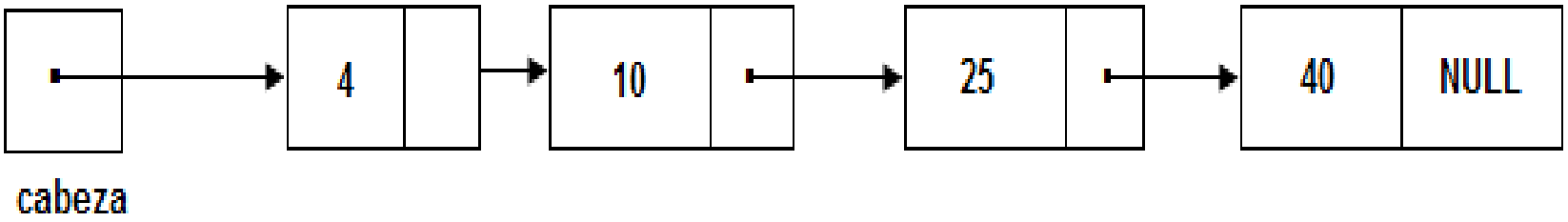
Se cambia el apuntador de cabeza para apuntar al nuevo nodo creado; es decir, el puntero de cabeza apunta al mismo sitio que apunte nuevo.



En este momento, la función de insertar un elemento termina su ejecución la variable local nuevo desaparece y sólo permanece el apuntador de cabeza que apunta a la nueva lista enlazada.

Notas de estructura de datos con lenguaje C

La variable local nuevo desaparece y sólo permanece el apuntador de cabeza que apunta a la nueva lista enlazada:



El código de la función InsertarCabezaLista:

```
Void InsertarCabezaLista (Nodo** cabeza, ítem entrada);
{
    Nodo *nuevo;
    nuevo = (Nodo*)malloc (sizeof (Nodo));          /*asigna nuevo
    nodo*/
    nuevo -> dato = entrada;                        /*pone elemento en
    nuevo*/
    nuevo -> siguiente = *cabeza;                   /* enlaza nuevo
    nodo al frente de la lista*/
*cabeza = nuevo                                     /* mueve puntero cabeza y apunta al
    nuevo nodo*/
    }
```

Notas de estructura de datos con lenguaje C

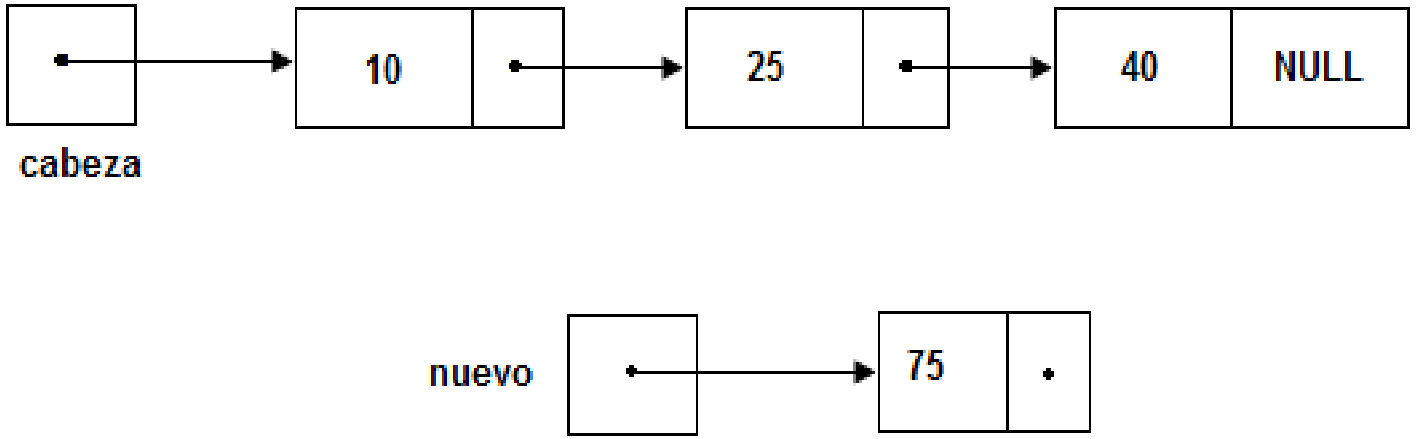
La función InsertarCabezaLista actúa también correctamente si se trata el caso de añadir un primer nodo o elemento a una lista vacía como ya se ha comentado cabeza apunta a NULL y termina apuntando al nuevo nodo de la lista enlazada.

Notas de estructura de datos con lenguaje C

Insertar un nodo nuevo (no por al inicio) en la lista un nuevo nodo no siempre inserta al principio.

Es posible insertar en el centro o al final de la lista.

Por ejemplo: *insertar un nuevo elemento 75 entre el elemento 25 y el elemento 40 en la lista enlazada 10, 25, 40.*



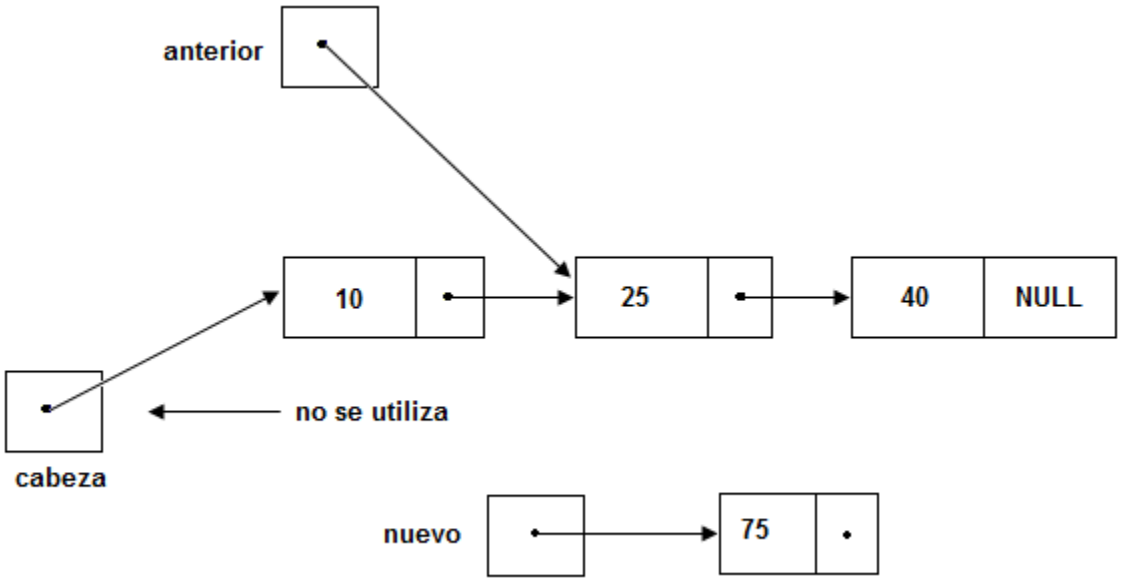
Notas de estructura de datos con lenguaje C

Los pasos son:

1. Asignar el nuevo nodo apuntado por el puntero nuevo.
2. Situar el nuevo elemento en el campo dato del nuevo nodo.
3. Hacer que el campo enlace siguiente del nuevo nodo apunte al nodo que va después de la posición del nuevo nodo (o bien NULL si no hay ningún nodo después de la nueva posición).
4. En la variable puntero anterior tener la dirección del nodo que está antes de la posición deseada para el nuevo nodo. Hacer que anterior -> siguiente apunte al nuevo nodo que se acaba de crear.

Notas de estructura de datos con lenguaje C

En el paso 1 y 2:

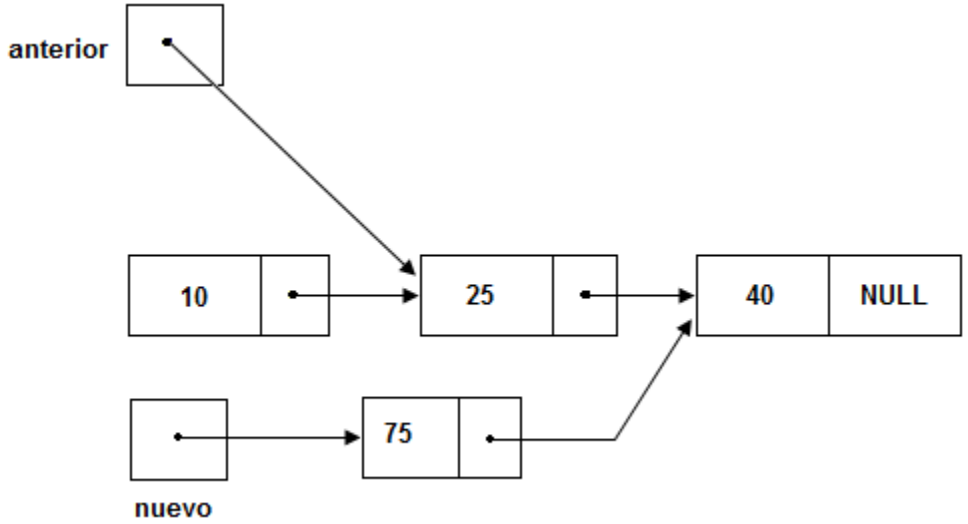


nuevo = (Nodo*)malloc (sizeof(Nodo));

nuevo -> dato= entrada;

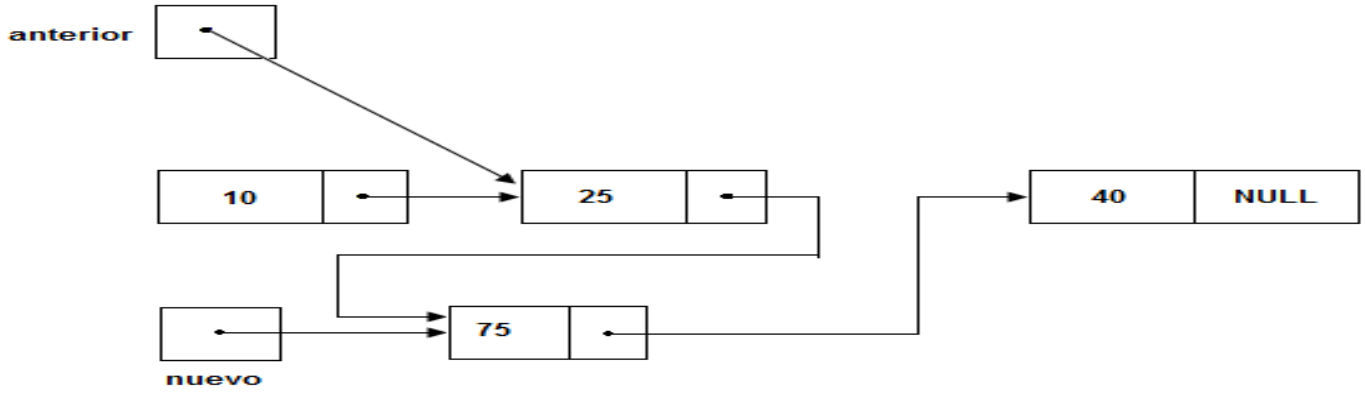
Notas de estructura de datos con lenguaje C

Paso 3



nuevo-> siguiente = anterior -> siguiente

Paso 4:



Después de ejecutar todas las sentencias de las sucesivas etapas, la nueva lista comienza en el nodo10, siguen 25, 75 y, por último, 40.

```
void InsertarLista (Nodo* anterior, Item entrada);
{
    Nodo *nuevo;
nuevo = (Nodo*)malloc (sizeof (Nodo));
nuevo -> dato = entrada;
nuevo -> siguiente = anterior -> siguiente;
anterior -> siguiente = nuevo;
}
```

Notas de estructura de datos con lenguaje C

Inserción al final de la lista:

La inserción al final de la lista es menos eficiente:

Porque normalmente, no se tiene un apuntador al último elemento de la lista entonces se ha de seguir el recorrido desde el inicio (cabeza) de la lista hasta el último nodo y a continuación realizar la inserción

Cuando ultimo es una variable apuntador que apunta al último nodo de la lista, las sentencias siguientes insertan un nodo al final de la lista.

Notas de estructura de datos con lenguaje C

ultimo -> siguiente = (Nodo*)malloc(sizeof (Nodo));

En esta instrucción se asigna un nuevo nodo que está apuntando por el campo siguiente al último nodo de la lista (antes de la inserción) de modo que el nuevo nodo ahora es el último nodo de la lista

ultimo -> siguiente -> dato = entrada;

Establece el campo dato del nuevo último nodo al valor de entrada.

ultimo -> siguiente -> siguiente = NULL;

Establece el campo siguiente del nuevo nodo a NULL.

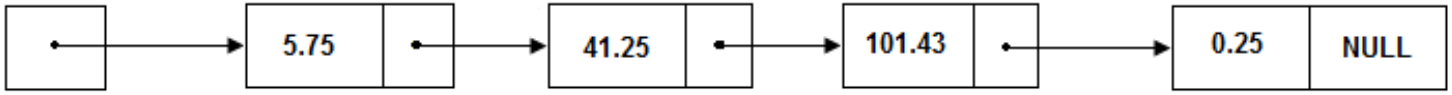
ultimo = ultimo -> siguiente;

Pone la variable ultimo al nuevo último nodo de la lista.

Notas de estructura de datos con lenguaje C

Para buscar un elemento en la lista

Como una función en C puede devolver el valor apuntador entonces al ubicar un elemento en la lista se puede regresar el apuntador a ese elemento



Si tenemos una función BuscarLista emplea una variable apuntador llamada índice que va recorriendo la lista nodo a nodo.

Si usamos una estructura de ciclo.

Como se puede observar en el siguiente código .

```
Nodo* BuscarLista (Nodo* cabeza, Item destino)
/* cabeza:        apuntador de inicial (cabeza) de una lista enlazada.
   destino:       dato que se busca en la lista.
   indice:        valor de retorno, apuntador que apunta al primer nodo que
    contiene el destino (elemento buscado); si no existe el nodo devuelve
    apuntador nulo.*/
{
    Nodo* índice;
    for (indice=cabeza; índice !=NULL; indice=indice->siguiente)
          if (destino==indice ->dato)
                return indice;
    return NULL;
}
```

índice apunta a los nodos de la lista

entonces si se encuentra el nodo buscado devuelve un apuntador al nodo buscado con la sentencia de retorno (return)

caso contario la función devuelve NULL (return NULL).

Notas de estructura de datos con lenguaje C

Eliminar un nodo en una lista

Para eliminar un nodo en la lista se debe enlazar el nodo anterior con el nodo siguiente del nodo que se quiere eliminar y así liberara la memoria ocupada.

Pasos de eliminación de un nodo:

1. Búsqueda del nodo que contiene el dato. Se obtiene la dirección del nodo a eliminar y la dirección del anterior.
2. El apuntador siguiente del nodo anterior apunta al siguiente del nodo a eliminar.
3. En caso de que el nodo a eliminar sea el primero(cabeza) se modifica cabeza para que tenga la dirección del nodo siguiente.
4. Se libera la memoria ocupada por el nodo.

La función que recibe la cabeza de la lista y el dato del nodo que se quiere borrar.

```
viod eliminar (Nodo** cabeza, item entrada)
{
    Nodo* actual, *anterior;
    int encontrado = 0;
    actual = cabeza; anterior = NULL;
    /*ciclo de búsqueda*/
    while ((actual =! NULL)&&( !encontrado))
    {
        encontrado= (actual-> dato == ebtrada);
        if (!encontrado)
        {
            anterior=actual;
            actual=actual->siguiente;
}
}
```

```
    /*Enlace de nodo anterior con siguiente*/
    if (actual !=NULL)
    {
/*Se distingue entre que el nodo inicial o del resto de la lista*/
if (actual == *cabeza)
{
    *cabeza = actual-> siguiente;
}
else{
    anterior-> siguiente = actual -> siguiente
  }
free (actual);
}
}
```

LISTA DOBLEMENTE ENLAZADA

Notas de estructura de datos con lenguaje C

- En numerosas aplicaciones es conveniente poder acceder a los elementos o nodos de una lista en cualquier orden.
- En este caso se recomienda el uso de una **lista doblemente ligada**.

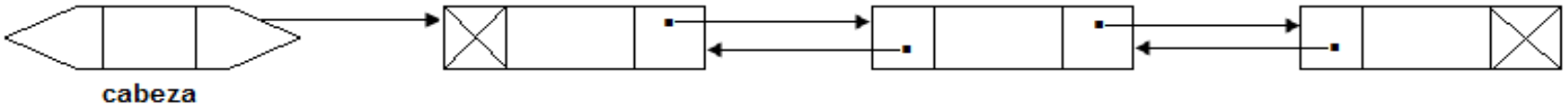
  La característica de estas es que cada elemento aparte del valor almacenado en el elemento tienen dos apuntadores:

a) Uno que apunta al siguiente elemento de la lista.

b) eE otro apuntador apunta al elemento anterior.

Notas de estructura de datos con lenguaje C

Lista doblemente ligada:

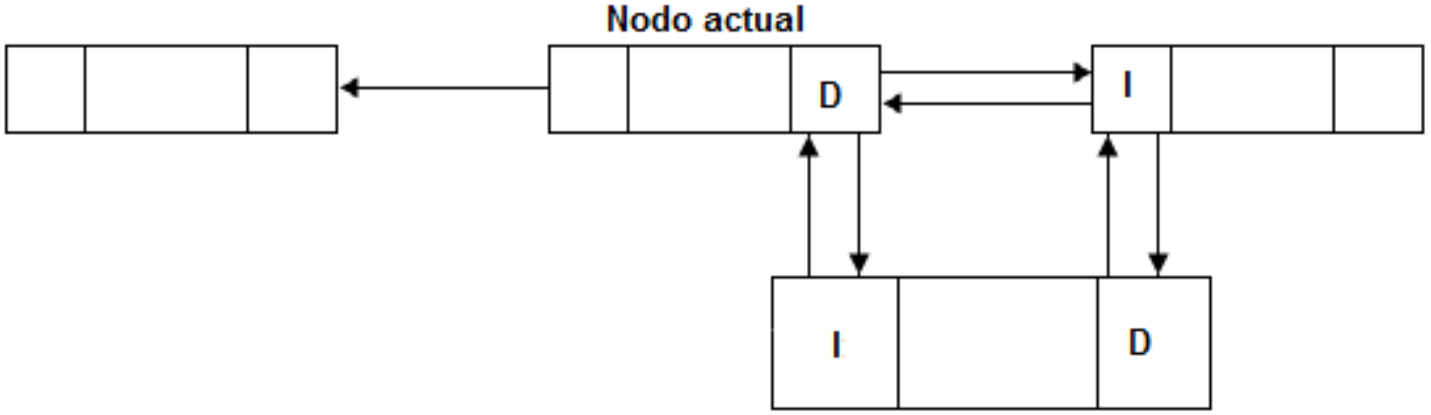


(a)

(b)

Notas de estructura de datos con lenguaje C

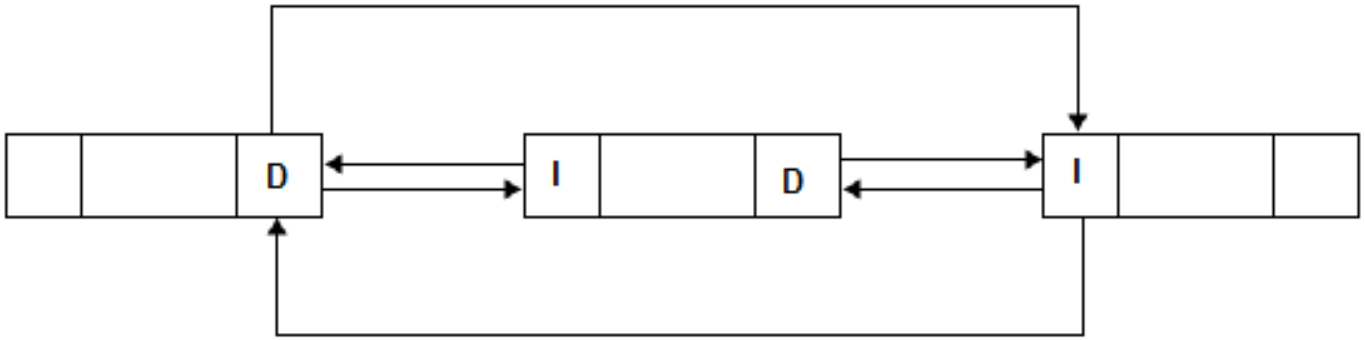
- Existe una operación de *insertar y eliminar* (borrar) en cada dirección.
- Para insertar un nodo *p* a la derecha del nodo actual.



- Deben asignarse cuatro nuevos enlaces.

Notas de estructura de datos con lenguaje C

- Para *eliminar* (borrar) un nodo de una lista doblemente enlazada es preciso cambiar dos apuntadores.



- Eliminación de un nodo en una lista doblemente enlazada.

Notas de estructura de datos con lenguaje C

Como se declara una lista doblemente enlazada

Para lista doblemente enlazada con valores de tipo int necesita dos apuntadores y el valor del campo datos. En una estructura se agrupan estos datos:

```
typedef int Item;
struct unnodo
{
Item dato;
struct unnodo *adelante;
struct unnodo *atras;
};
typedef struct unnodo Nodo;
```

Notas de estructura de datos con lenguaje C

Para insertar un elemento en la lista doblemente ligada

Para añadir o insertar un elemento en una lista doblemente ligada depende de la posición en que se inserta el elemento.

La posición de inserción puede ser:

- En el elemento primero (cabeza) de la lista.
- En el final de la lista (elemento último).
- Antes de un elemento especificado.
- Después de un elemento especificado.

Notas de estructura de datos con lenguaje C

Para insertar un nuevo elemento al principio de la lista doble

El proceso puede ser:

1. Asignar un nuevo nodo apuntado por nuevo que es una variable apuntador que apunta al nuevo nodo que se va a insertar en la lista doble.
2. Situar al nuevo elemento en el campo dato del nuevo nodo.
3. Hacer que el campo enlace adelante del nuevo nodo apunte al primer nodo (cabeza) de la lista original, y que el campo enlace atrás del nodo cabeza apunte al nuevo nodo.
4. Hacer que cabeza (apuntador cabeza) apunte al nuevo nodo que se ha creado.

```
typedef int Item;
typedef struct tipo_nodo
{
    Item dato;
    struct tipo_nodo* adelante;
    struct tipo_nodo* atras;
}Nodo;
Nodo* nuevo;
nuevo = (Nodo*)malloc (sizeof (Nodo));
nuevo -> dato = entrada
nuevo -> adelante = cabeza ;
nuevo -> atrás = NULL;
cabeza -> atrás = nuevo;
cabeza = nuevo;
```

Notas de estructura de datos con lenguaje C

La función de insertar un elemento en la lista termina su ejecución, la variable local nuevo desaparece y sólo permanece el apuntador de cabeza que apunta a la nueva lista doblemente enlazada.

Notas de estructura de datos con lenguaje C

Inserción de un nuevo nodo(no por el principio de la lista) en cualquier parte de la lista.

1. Asignar el nuevo nodo apuntado por el apuntador nuevo.
2. Situar el nuevo elemento en el campo dato del nuevo nodo.
3. Hacer que el campo enlace adelante del nuevo nodo apunte al nodo que va después de la posición del nuevo nodo (o bien a NULL si no hay ningún nodo después de la nueva posición). El campo atrás del nodo siguiente al nuevo tiene que apuntar a nuevo.
4. La dirección del nodo que está antes de la posición deseada para el nuevo nodo está en la variable apuntador anterior. Hacer que anterior -> adelante apunte al nuevo nodo. El enlace atrás del nuevo nodo debe de apuntar a anterior.