Tipos Abstratos de Datos (TADs)

Tipos Abstractos de Datos (TADs)

- Programación estructurada: utiliza fundamentalmente instrucciones secuenciales, de selección y repetitivas.

Algoritmos + datos = Programa

- El diseño de un programa estructurado se consigue dividiendo el programa (o problema) en partes más pequeñas (funciones en C, C++, Java).

Tipos Abstractos de Datos (TADs)

- A veces, este modelo (datos y funciones que manipulan datos) es deficiente para modelar cosas y objetos del mundo real.
- Por ejemplo, personas, casas, bicicletas tienen a su vez incorporados ***atributos*** (datos) y ***comportamiento*** (funciones).
- Los objetos tienen atributos y comportamiento.

Tipos Abstractos de Datos (TADs)

- Atributos o características: propiedades de los objetos:

Persona = Estatura (1,75m)
Edad (23)
DNI (123456789)

Auto = Marca (Fiat)
Modelo (1987)
Precio (123456)

Libro = Autor (Cacho Castaña)
Título (Programando la vida)
Editorial (McGrawHill)
Año(1987)

Atributos = Datos)

Tipos Abstractos de Datos (TADs)

- Comportamiento: acción que realizan los objetos en respuesta a un estímulo.

Persona = imprimir_Datos()
aAgregar_Altura()
agregar_Edad()

Auto = acelerar()
frenar()
encenderr()
apagar()

Libro = imprimir_datos()
asignar_autor()

Comportamiento = Funciones)

Tipos Abstractos de Datos (TADs)

Combinar en 1 sola entidad los datos y las funciones que actúan sobre los datos

Objetos

Identificación de datos y de operaciones!!

Tipos Abstractos de Datos (TADs)

Los TADs son el primer paso hacia la programación Orientada a objetos (en lenguaje C).

Para los Tipos Abstractos de Datos
Debemos a detectar atributos y funcionalidad

Atributos
Marca
Número de serie
Capacidad
Potencia

Operaciones
encender()
apagar()
lavar()
aclarar()

Listas

Listas

- Es una colección de elementos (del mismo tipo). En dicha colección podremos agregar elementos, imprimirlos, sacar elementos, etc.
- Estructura de datos dinámicas.
- Crecen y se contraen a medida que se ejecuta el programa.

Listas

Las listas se pueden dividir en cuatro categorías:

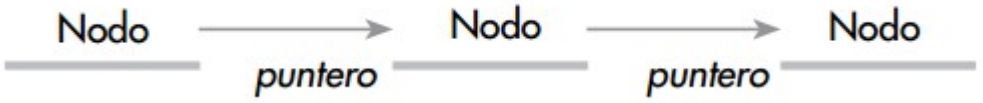
- Listas simplemente enlazadas. Cada nodo (elemento) contiene un único enlace que conecta ese nodo al nodo siguiente o nodo sucesor. La lista es eficiente en recorridos directos (“adelante”)
- Listas doblemente enlazadas. Cada nodo contiene dos enlaces, uno a su nodo predecesor y el otro a su nodo sucesor. La lista es eficiente tanto en recorrido directo (“adelante”) como en recorrido inverso (“atrás”).
- Lista circular simplemente enlazada. Una lista enlazada simplemente en 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 en 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 directa (“adelante”) como inversa (“atrás”).

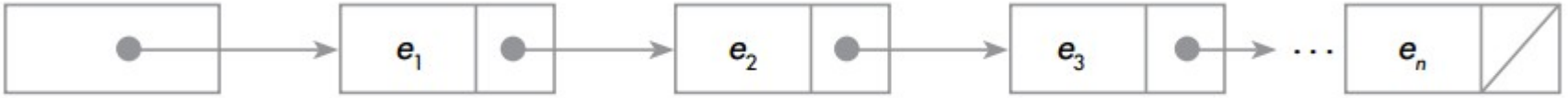
Listas

- Por cada uno de estos cuatro tipos de estructuras de listas, se puede elegir una **implementación basada en arreglos** o una **implementación basada en punteros**.
- 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, según la forma de reservar memoria las implementaciones pueden hacerse con:
    - Asignación fija, o estática, de memoria mediante arreglos.
    - Asignación dinámica de memoria mediante punteros.

- Dado que la asignación fija de memoria mediante arrays es más ineficiente, se presentará la implementación usando punteros, quedando la opción de usar arreglos para la práctica.

Listas simplemente enlazadas

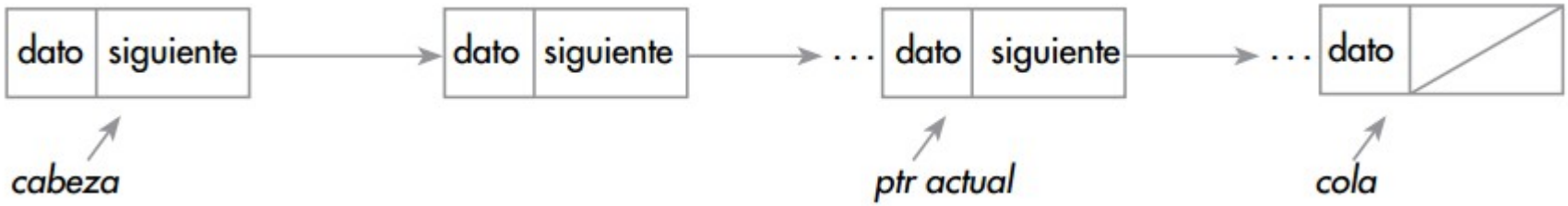


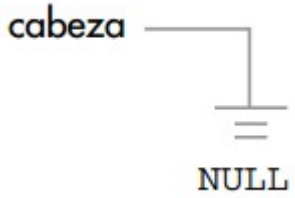


e_1, e_2, ... e_n son valores del tipo `TipoElemento`

Estructura de una lista: consta de un número de elementos y cada elemento tiene 2 componentes (campos), **un puntero al siguiente elemento** de la lista y un **valor**, que puede ser cualquier tipo.

Listas simplemente enlazadas





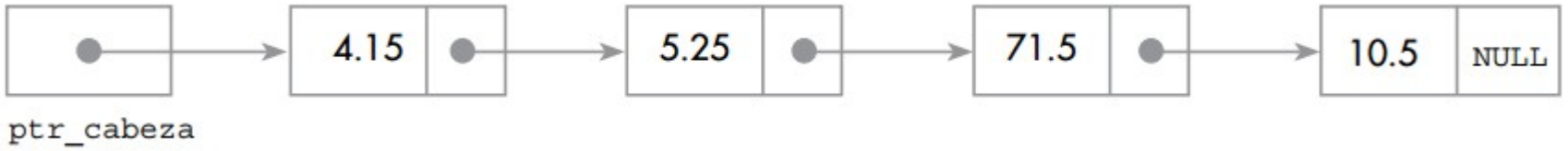
Lista vacía

Listas simplemente enlazadas

- Algunas operaciones:
  - Crear lista.
  - Insertar primero.
  - Insertar último.
  - Mostrar lista (imprimir).
  - Longitud.
  - Insertar en posición determinada.
  - Eliminar dato.
  - Eliminar datos posición determinada, etc.

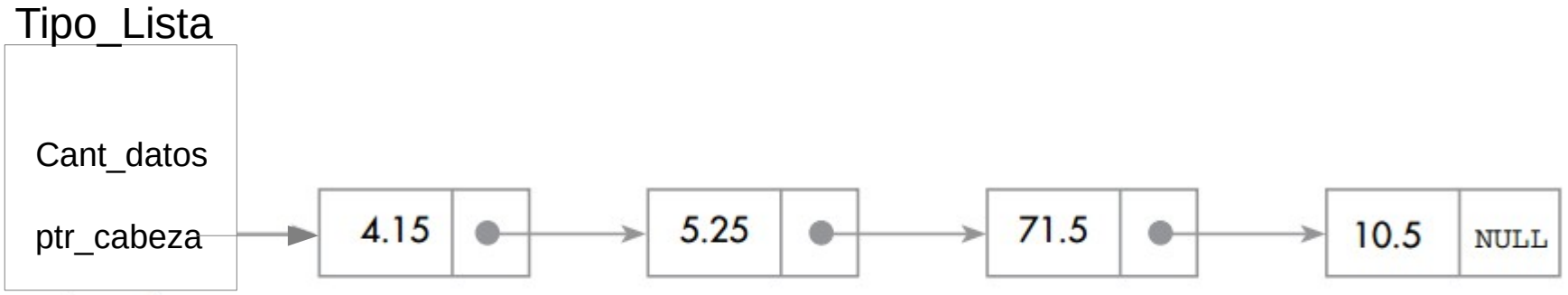
Listas simplemente enlazadas

- Existen distintas implementaciones.
  - Una es usando un puntero directamente al primer elemento de la lista (cabeza).



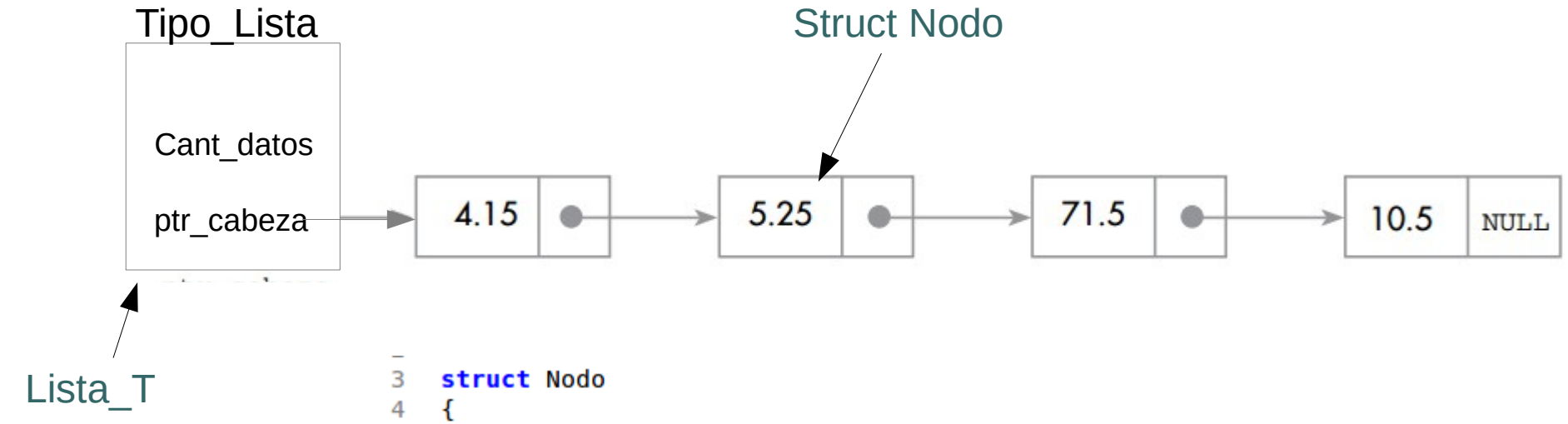
Listas simplemente enlazadas

- Existen distintas implementaciones.
  - Otra posible es usando una estructura previa que contiene más datos de la lista + enlace al primer elemento de la lista.



Listas simplemente enlazadas

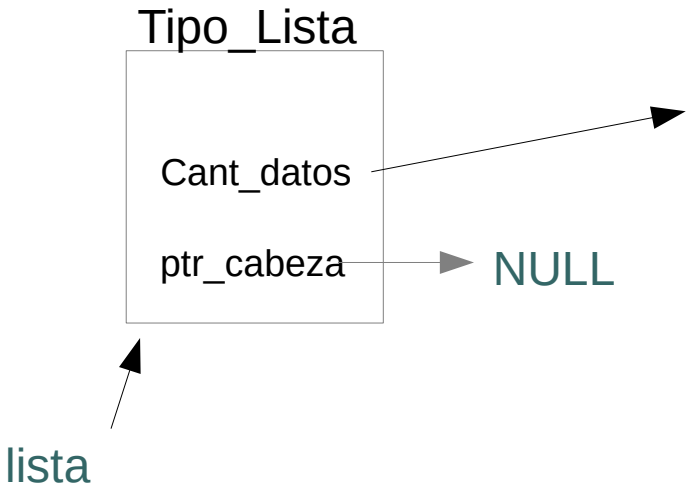
- Para los ejemplos se usará la segunda opción




```
3   struct Nodo
4   {
5     int dato;
6     struct Nodo *sig;
7   };
8
9   typedef struct{
10    int n;
11    struct Nodo *lista;
12  }Lista_T;
```

Listas simplemente enlazadas

- Crear lista = retorna una lista vacia.

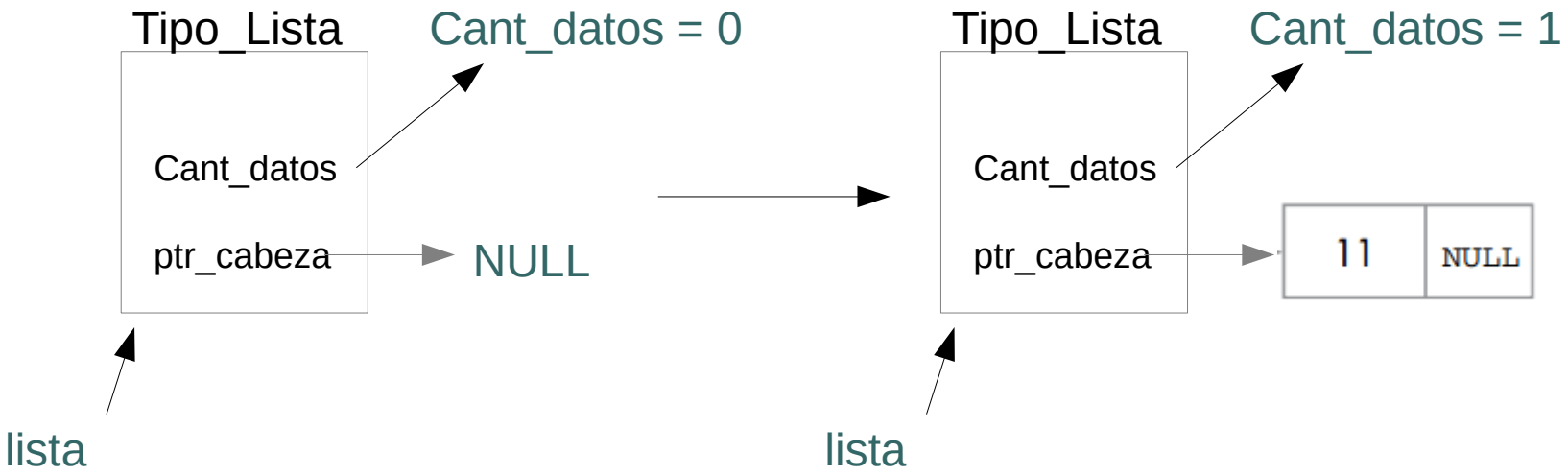



```
3  struct Nodo
4  {
5    int dato;
6    struct Nodo *sig;
7  };
8
9  typedef struct{
10   int n;
11   struct Nodo *lista;
12 }Lista_T;
```

```
6  Lista_T CrearLista()
7  {
8      Lista_T l;
9      l.lista = NULL;
10     l.n = 0;
11
12     return(l);
13 }
14
```

Listas simplemente enlazadas

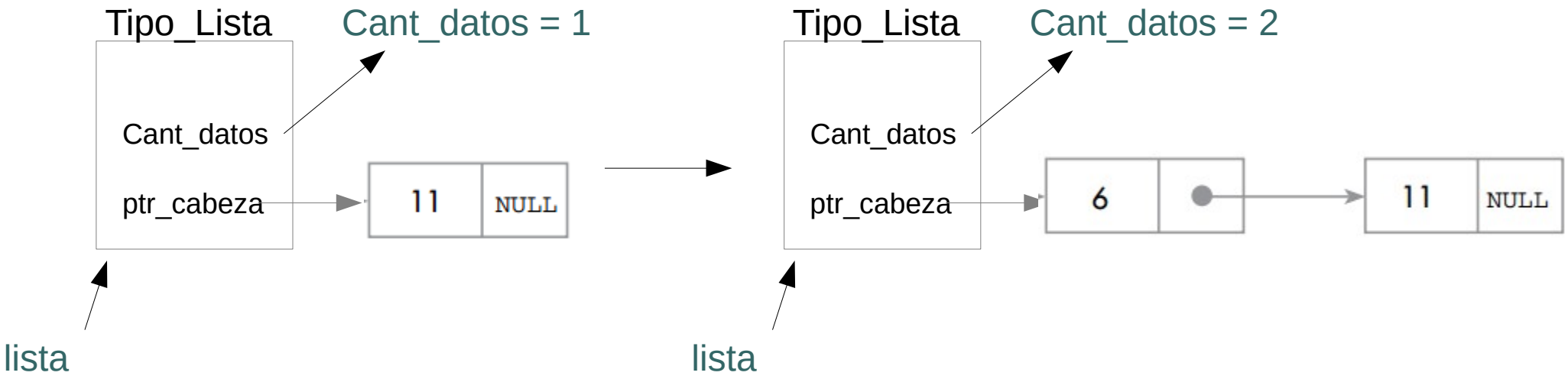
- Insertar_primero (Tipo_Lista * lista, int dato)



1) Crear el nodo
2) Agregar nuevo nodo a la lista

Listas simplemente enlazadas

- Insertar_primero (Tipo_Lista * lista, int dato)



Quiero insertar el 4 en esta lista:

Pasos 1 y 2

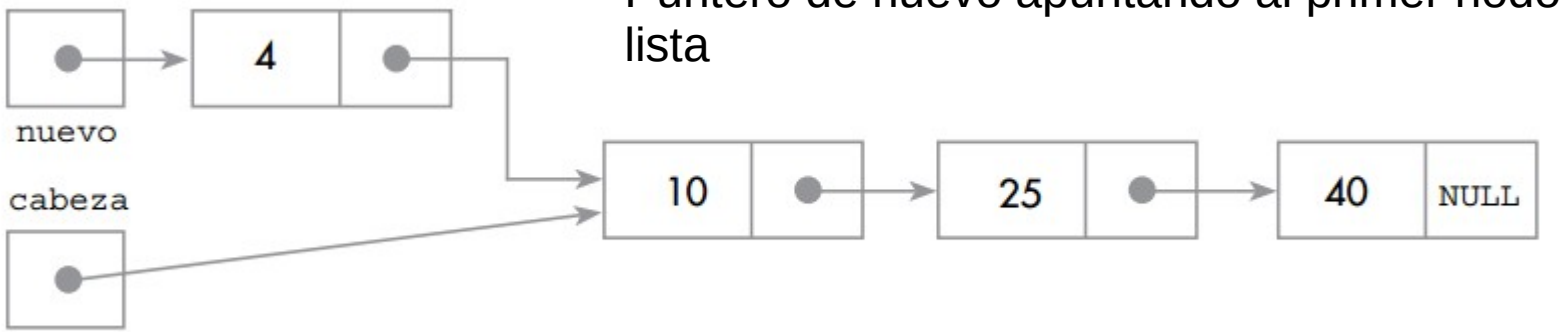
Malloc del nuevo nodo

nuevo

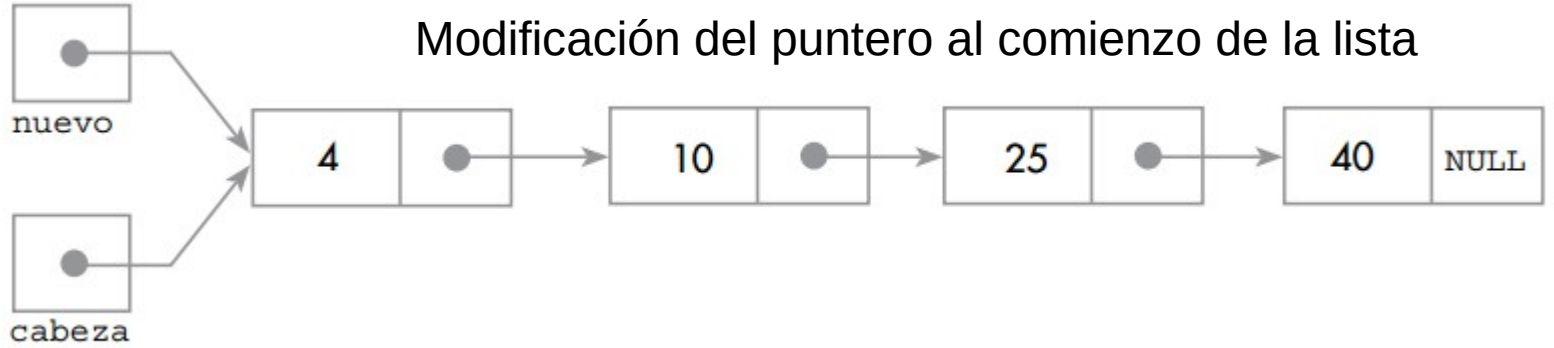
cabeza

Lista antes de la inserción

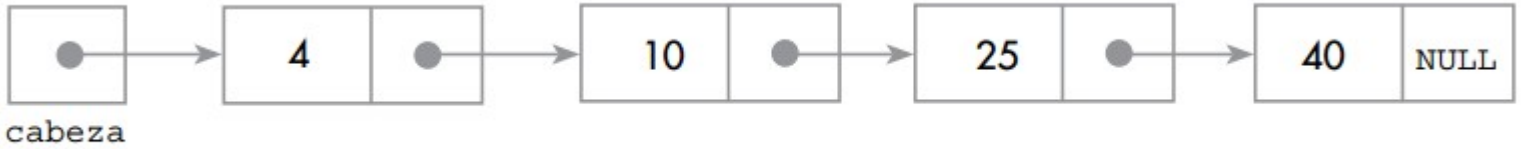
Puntero de nuevo apuntando al primer nodo de la lista



Modificación del puntero al comienzo de la lista



Lista resultante



Listas simplemente enlazadas

- Insertar_primero (Tipo_Lista * lista, int dato)

```
16  int InsertarPrimero(Lista_T *l, int x)
17  {
18      struct Nodo *nuevo;
19
20      if ((nuevo = (struct Nodo*)malloc(sizeof(struct Nodo))) == NULL)
21      {
22          printf("No se pudo alocar memoria \n");
23          exit(-1);
24      }
25
26      nuevo->dato = x;
27      nuevo->sig = l->lista;
28
29      l->lista = nuevo;
30      l->n = l->n+1;
31
32      return 0;
33  }
```

1) Crear el nodo
2) Agregar nuevo nodo a la lista

Listas simplemente enlazadas

- Insertar_primero (Tipo_Lista * lista, int dato)

```c
int InsertarPrimero(Lista_T *l, int x)
{
    // creacion del nodo con el nuevo elemento
    struct Nodo *nuevo;
    nuevo = CrearNodo(x);

    // actualizacion de punteros
    nuevo->sig = l->lista;
    l->lista = nuevo;

    //actualización de la cantidad de elementos de la lista
    l->n = l->n+1;

    return 0;
}
```

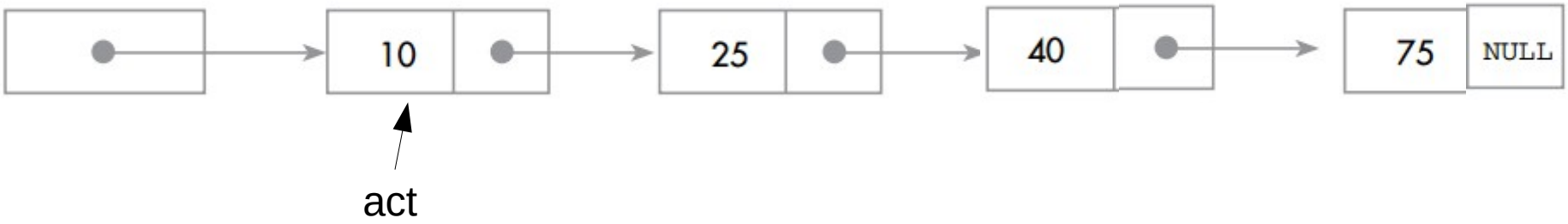
Otra implementación

```c
struct Nodo* CrearNodo(int dato){

    struct Nodo *nuevo;

    if ((nuevo = (struct Nodo*)malloc(sizeof(struct Nodo))) == NULL)
    {
        printf("No se pudo alocar memoria \n");
        exit(-1);
    }

    nuevo->dato = dato;
    nuevo->sig = NULL;
    return nuevo;
}
```

1) Crear el nodo
2) Agregar nuevo nodo a la lista

Listas simplemente enlazadas

- Insertar_ultimo (Tipo_Lista * lista, int dato)

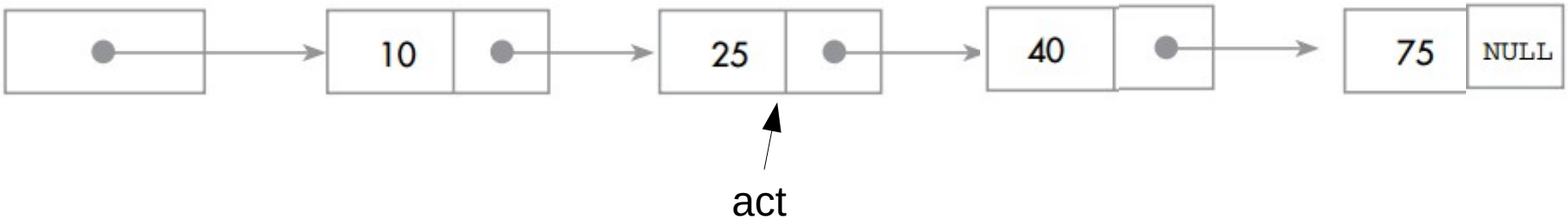


act → sig = nuevo;

1) Crear el nodo
2) Agregar nuevo nodo a la lista

Listas simplemente enlazadas

- Insertar_ultimo (Tipo_Lista * lista, int dato)

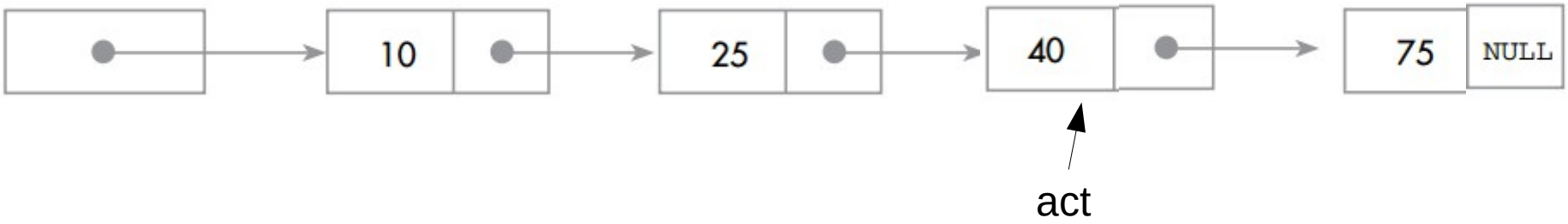


act → sig = nuevo;

1) Crear el nodo
2) Agregar nuevo nodo a la lista

Listas simplemente enlazadas

- Insertar_ultimo (Tipo_Lista * lista, int dato)



act → sig = nuevo;

1) Crear el nodo
2) Agregar nuevo nodo a la lista

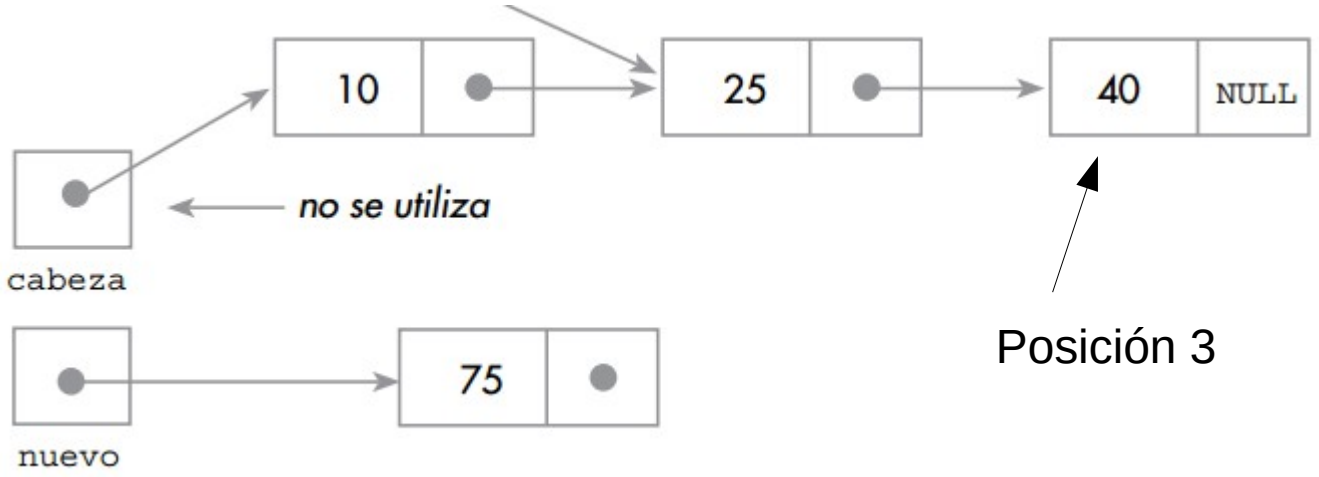
Listas simplemente enlazadas

Insertar_ultimo

```
70  int InsertarUltimo(Lista_T *l, int x)
71  {
72      // puntero auxiliar
73      struct Nodo *p;
74
75      // creacion del nodo con el nuevo elemento
76      struct Nodo *nuevo;
77      nuevo = CrearNodo(x);
78
79      // como es el ultimo nodo, el campo sig = NULL
80      nuevo->sig = NULL;
81
82      if (! EstaVacia(*l))  {
83
84          // puntero auxiliar apunta a la cabeza de la lista
85          p = l->lista;
86
87          // recorrido de la lista hasta el ultimo nodo
88          while (p->sig != NULL)  {
89              p = p->sig;
90          }
91
92          // actualizacion del puntero del último nodo
93          p->sig = nuevo;
94      }
95      else {
96          // si la lista esta vacia el nuevo nodo es el primero
97          l->lista = nuevo;
98      }
99      // actualizacion de la cantidad de elementos de la lista
100     l->n = l->n +1;
101
102     return 0;
103
104 }
105
```

Listas simplemente enlazadas

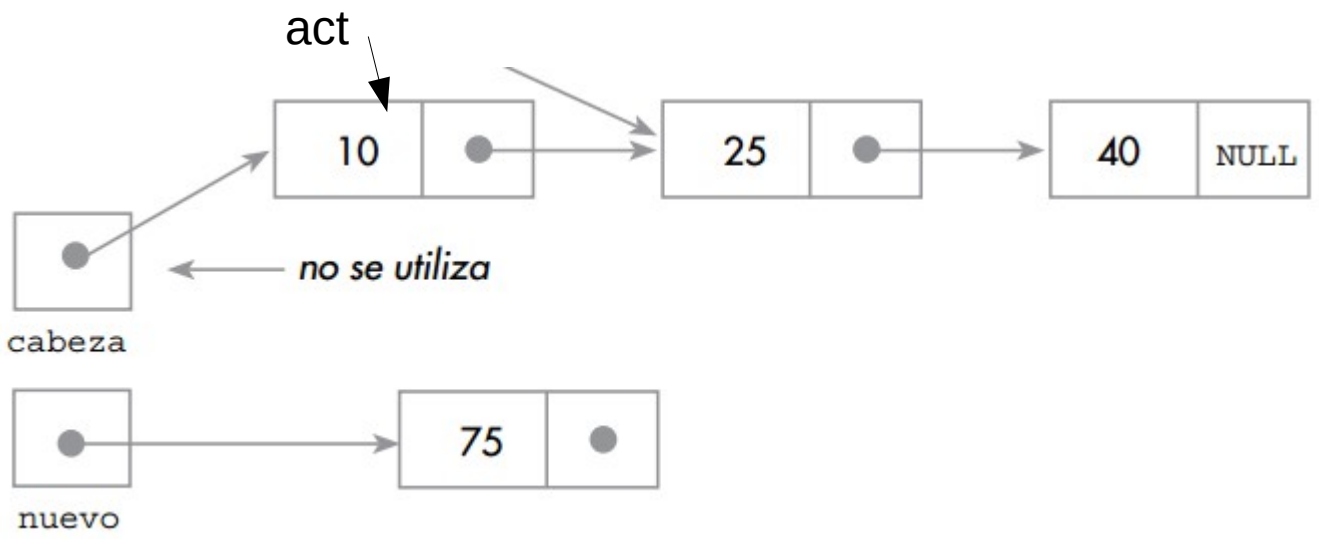
- Insertar_posicion (Tipo_Lista * lista, int dato, int posicion)



1) Crear el nodo
2) Agregar nuevo nodo a la lista

Listas simplemente enlazadas

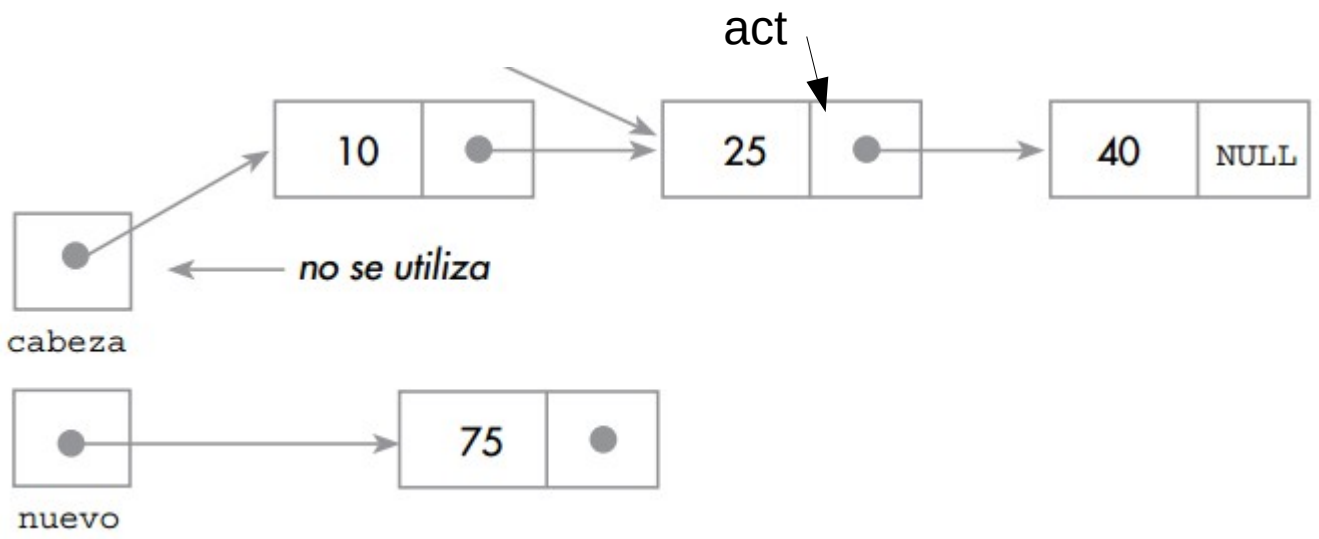
- Insertar_posicion (Tipo_Lista * lista, int dato, int posicion)



1) Crear el nodo
2) Agregar nuevo nodo a la lista

Listas simplemente enlazadas

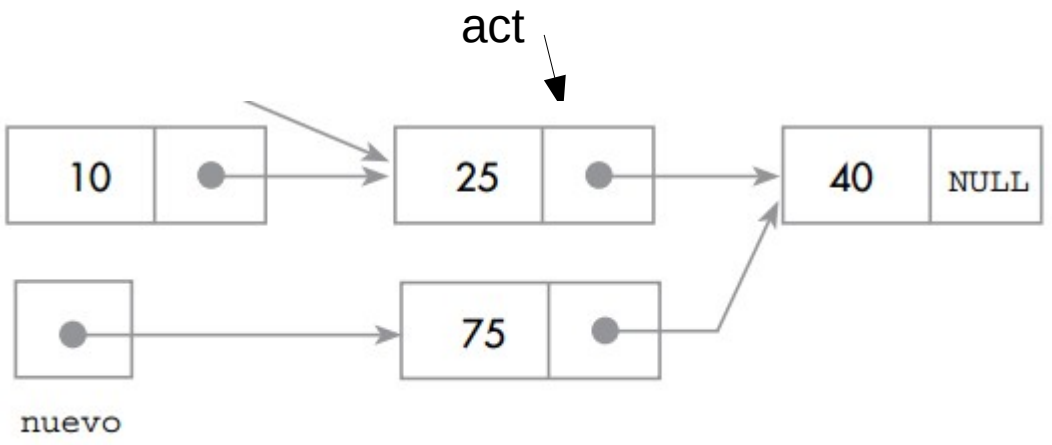
- Insertar_posicion (Tipo_Lista * lista, int dato, int posicion)



1) Crear el nodo
2) Agregar nuevo nodo a la lista

Listas simplemente enlazadas

- Insertar_posicion (Tipo_Lista * lista, int dato, int posicion)

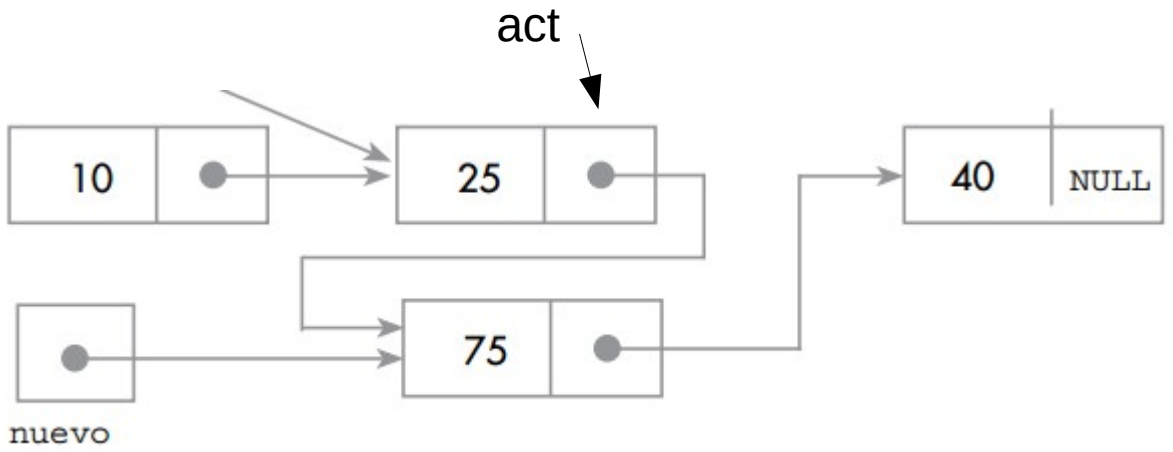


nuevo → sig = act → sig;

1) Crear el nodo
2) Agregar nuevo nodo a la lista

Listas simplemente enlazadas

- Insertar_posicion (Tipo_Lista * lista, int dato, int posicion)



Act → sig = nuevo;

1) Crear el nodo
2) Agregar nuevo nodo a la lista

Listas simplemente enlazadas

Insertar_posicion

```
178  // se asume como rango valido desde 1 a la cantidad de elementos
179  int InsertarPosicion(Lista_T *l, int pos, int x)
180  {
181      struct Nodo *act, *nuevo;
182      int i;
183
184      // verificacion que la posicion sea correcta
185      if ((pos >= 1) && (pos<= (LongitudLista(*l))))
186      {
187          // distingo la insercion al principio de la lista
188          if (pos == 1) {
189             InsertarPrimero(l, x);
190          }
191          else {
192             // creo el nuevo nodo
193             nuevo = CrearNodo(x);
194
195             // recorro la lista hasta pos - 1
196             act = l->lista;
197             i = 1;
198             while(i < pos-1) {
199               act = act->sig;
200               i++;
201             }
202
203             // actualizo punteros
204             nuevo->sig = act->sig;
205             act->sig = nuevo;
206             // actualizacion de la cantidad de elementos
207             l->n = l->n+1;
208          }
209       }
210       else // la posicion esta fuera de rango
211          printf("Posicion fuera de rango \n");
212
213      return 0;
214  }
215
```