

# Estructuras de Datos

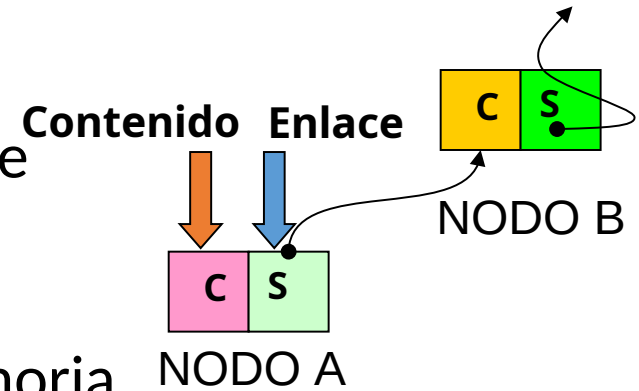
## Implementación Dinámicas de Listas

**Gonzalo Gabriel Méndez, Ph.D.**

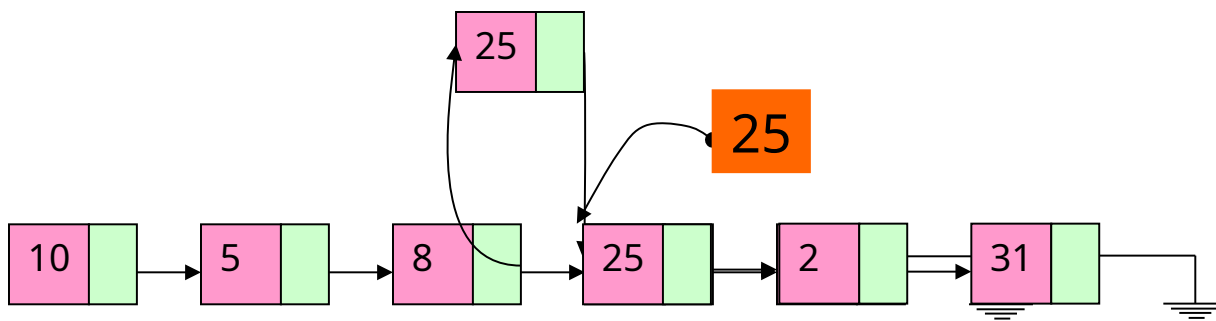
# TDA Lista

## Listas Simples Enlazadas (dinámico)

- Es una implementación flexible y potente



- Los nodos ya no son adyacentes en memoria
  - Un nodo A logra un enlace con otro B,
  - Almacenando dentro la dirección de memoria de B
- Al insertar o eliminar un nodo ya no hay que “mover” al resto de elemento, solo enlazarlo con la lista



# Lista Enlazada Simple

## Nivel de implementación

- Llevar control de las posiciones al primer y el último elemento
- Las posiciones son direcciones de memoria: **referencias**
  - *Header* y *Last* son referencias a Node en una lista enlazada
- La posición de un nodo estará dada por una referencia a dicho nodo
- Una lista enlazada no tiene datos predefinidos
  - Los elementos o Nodos van siendo creados y eliminados a medida que se va necesitando

# Lista Enlazada Simple

## Implementación

- Hay varias formas de definir una lista
  - Solo a través del primer elemento a la misma

```
public class SimpleLinkedList {  
    // reference to the first node.  
    private NodeList first;  
}
```

- O llevando control del primero y el último elemento

```
public class SimpleLinkedList {  
    // reference to the first node.  
    private NodeList first;  
    // reference to the last node.  
    private NodeList last;  
}
```

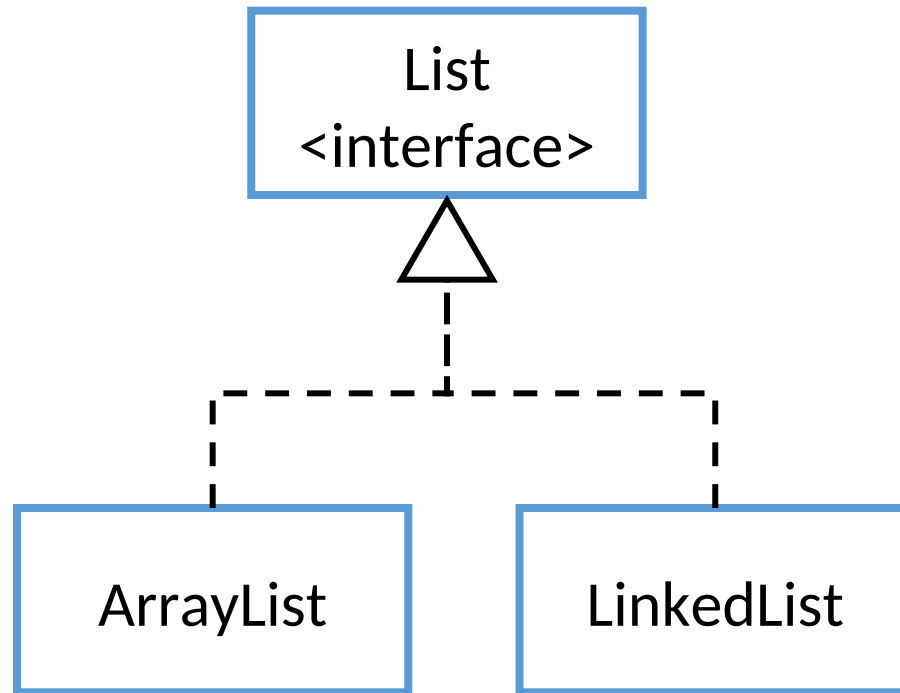
# Recap

# Implementación de Listas

- ArrayList y LinkedList son ambas estructuras de datos que implementan el TDA List
- La implementación determina la velocidad de las operaciones
- Elegir la estructura de datos para hacer la implementación de un TDA depende de qué operaciones se necesitan usar con el contenedor de datos

# ArrayList versus LinkedList

Ambas implementan la interface List



Se usan de manera similar, lo que varía es la implementación interna



# En Java

## ArrayList

- Arreglos con tamaño variable

- Crece 50% cada vez que cambia de tamaño

- Eficiente para get y set por posición

- No eficiente para añadir/remover al inicio/final

## LinkedList

t

- Eficientes para añadir y remover al inicio/final

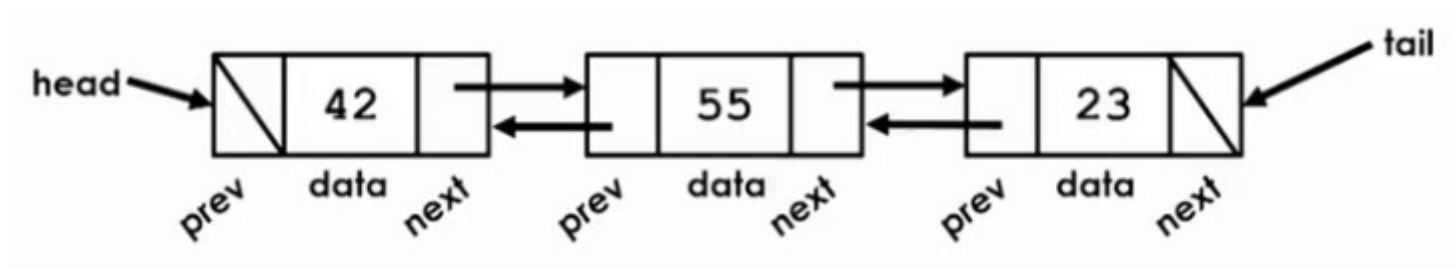
- Menos eficiente para get y set por posición

- En realidad, son doblemente enlazadas

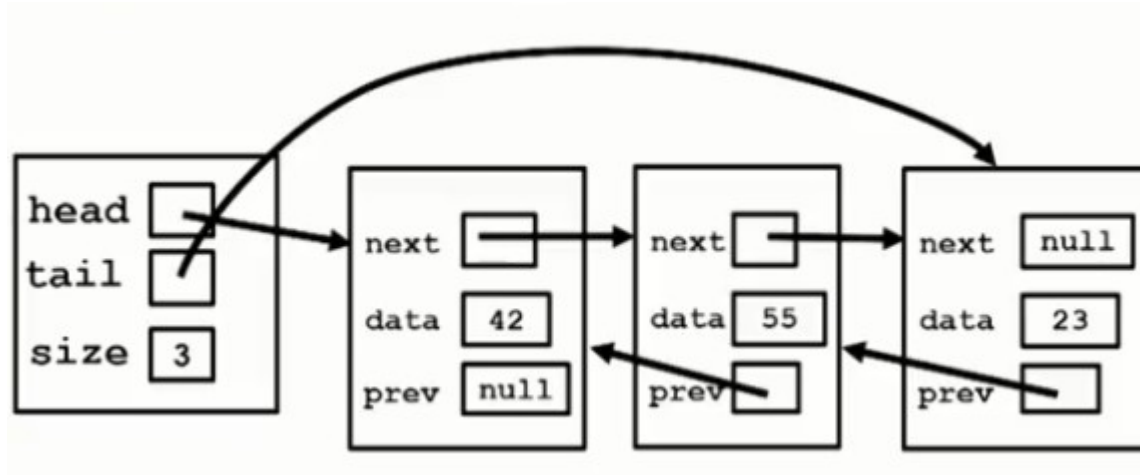
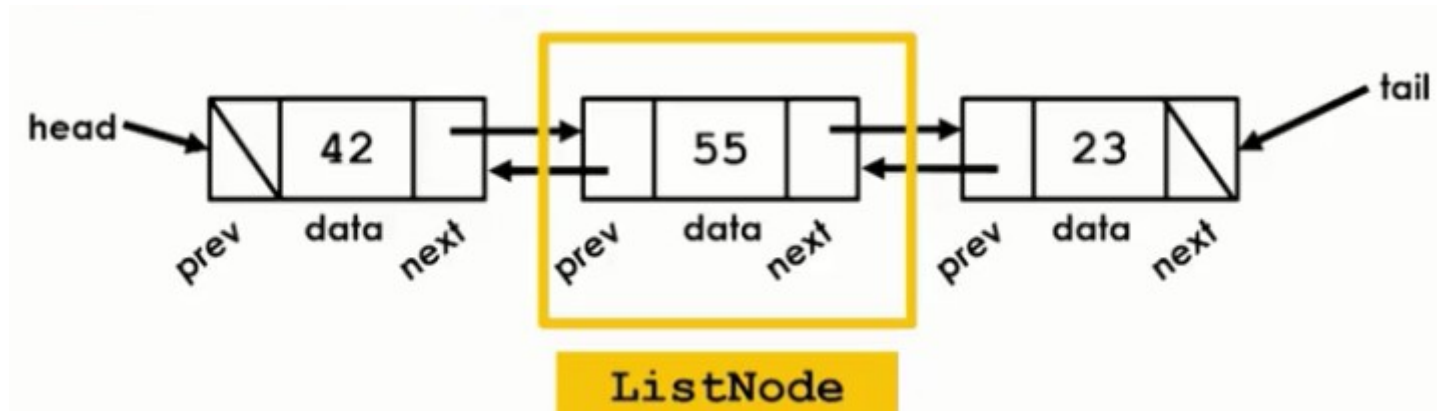
# **Listas Doblemente Enlazadas**

# Lista Doblemente Enlazada

- En las listas enlazadas solo se avanza en un sentido
- En las doblemente, se puede avanzar hacia la derecha o hacia la izq.
- En estas listas cada nodo tiene
  - Un predecesor, excepto el primero
  - Un sucesor, excepto el ultimo
- Cada nodo ya no tiene un solo enlace, tiene dos, hacia el siguiente y hacia el anterior



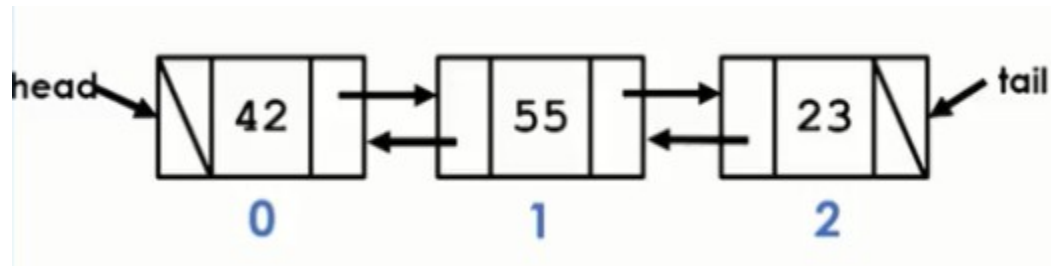
# Lista Doblemente Enlazada



# Lista Doblemente Enlazada

## Rendimiento de operaciones

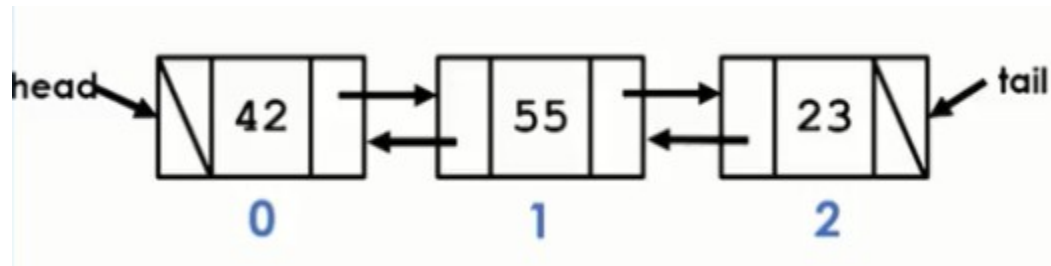
¿Cuánto tiempo toma acceder a un elemento en la implementación de LinkedList (en el peor caso)?



# Lista Doblemente Enlazada

## Rendimiento de operaciones

¿Cuánto tiempo toma acceder a un elemento en la implementación de LinkedList (en el peor caso)?



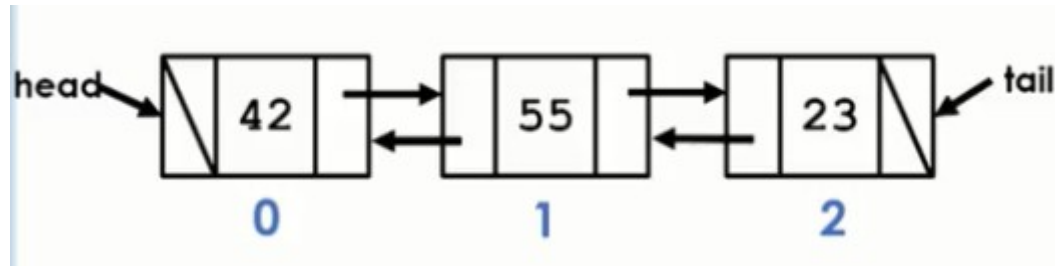
El peor de los casos

- El elemento está en el medio de la lista
- Moverse por a través de  $O(n/2)$  referencias desde la cabecera (o desde la cola) de la lista hasta el elemento que nos interesa
- $O(n)$

# Lista Doblemente Enlazada

## Rendimiento de operaciones

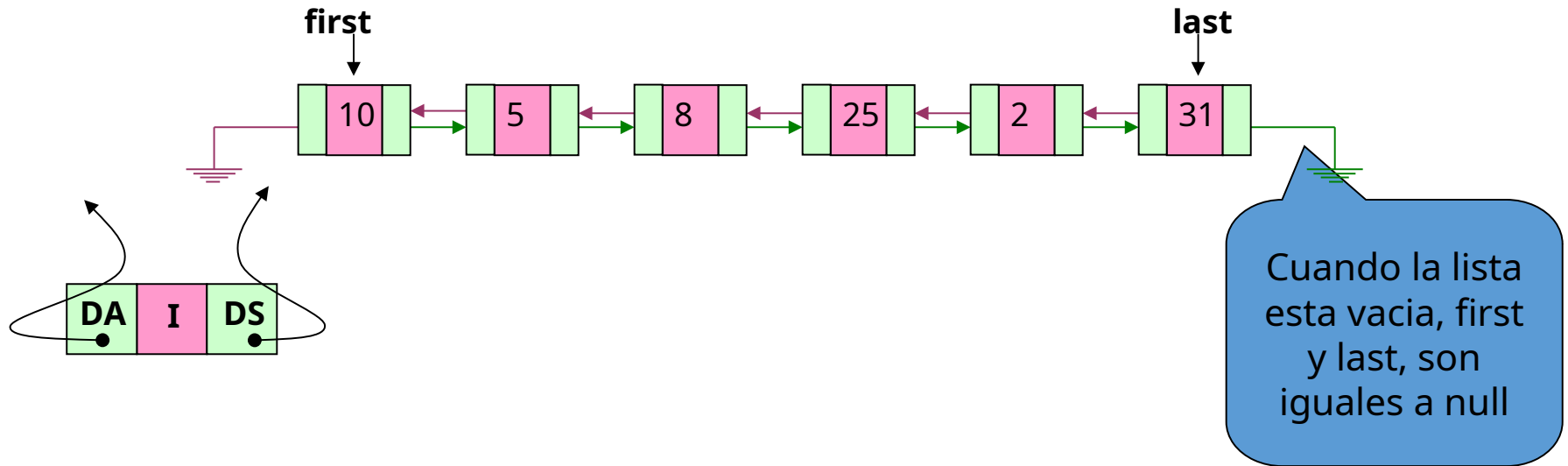
¿Cuánto tiempo toma insertar un elemento al inicio de la lista?



El número de operaciones es constante y no depende de cuantos elementos existen en la lista

$O(1)$

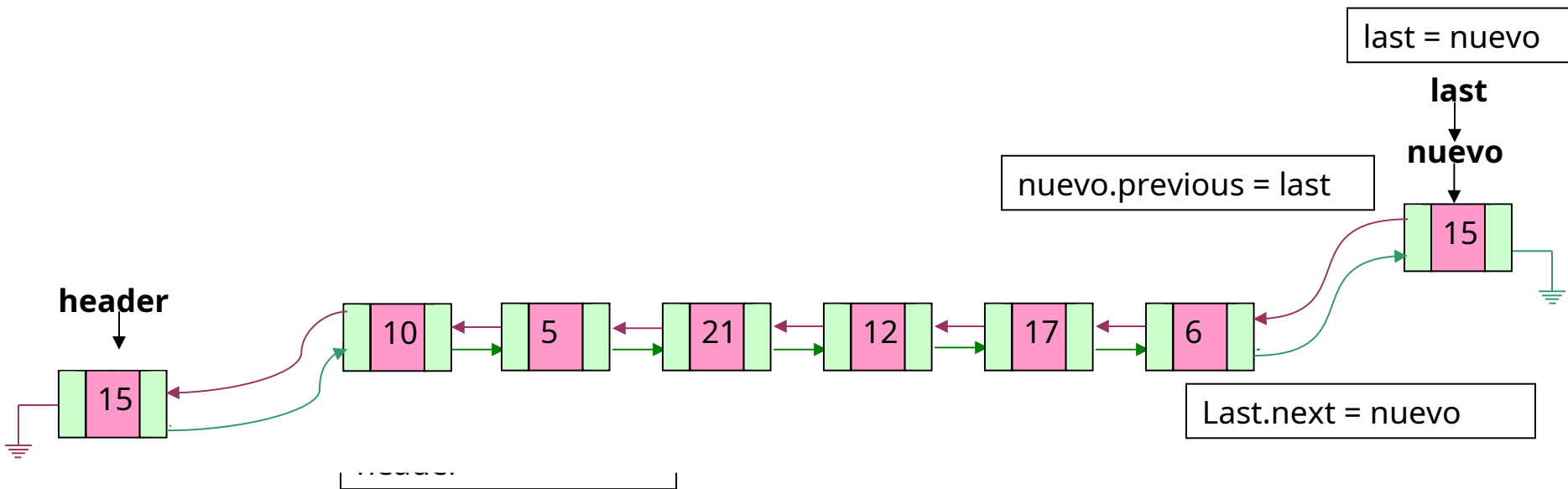
# Listas Doblemente Enlazadas Implementación





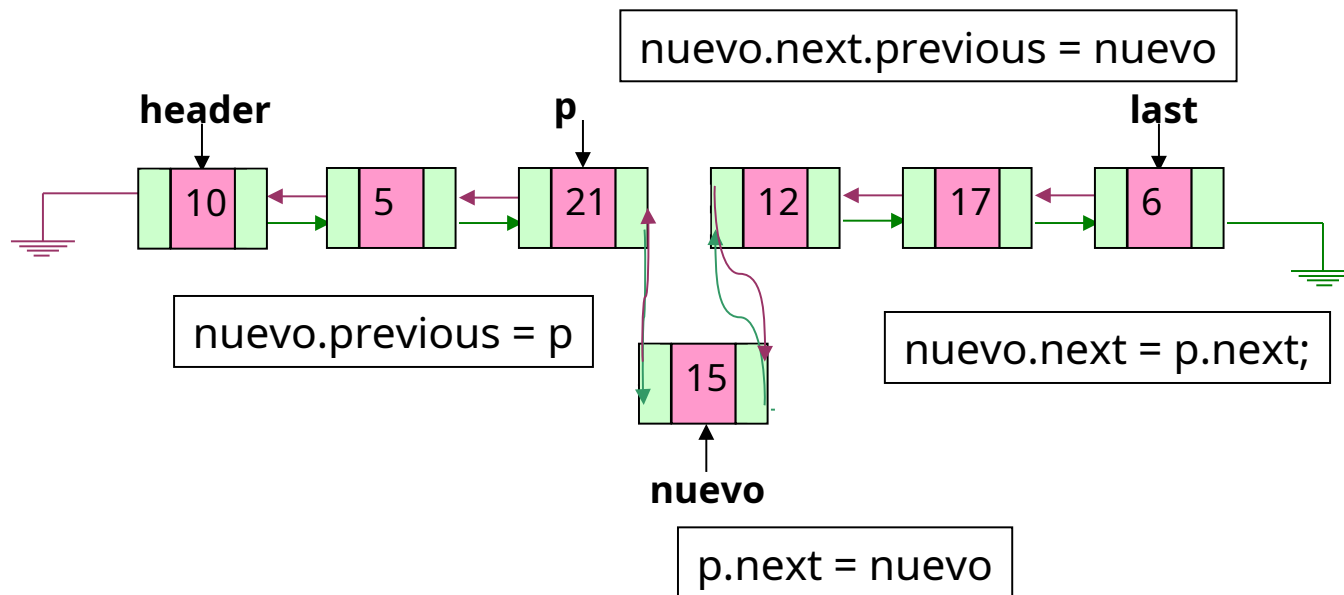
# Listas Doblemente Enlazadas

## Insertar nodos



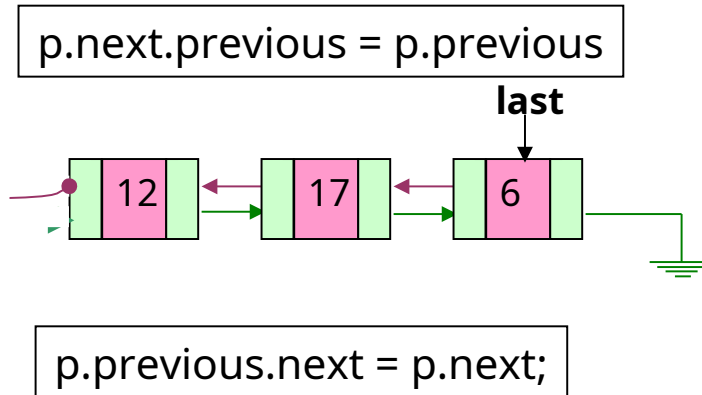
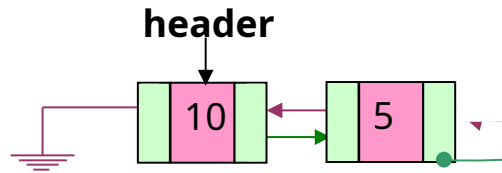
# Listas Doblemente Enlazadas

## Insertar entre



# Listas Doblemente Enlazadas

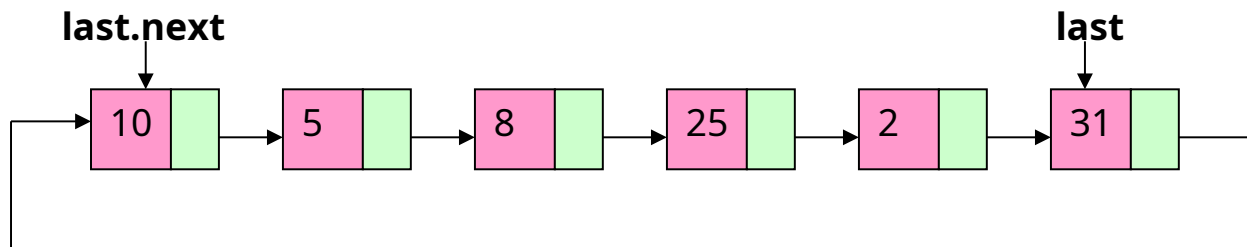
## Eliminar nodos



# Listas Circulares

# LISTAS CIRCULARES

- El avance en las listas enlazadas es
  - Solo a la derecha(siguiendo)
  - Limitado hasta el ultimo nodo
- Hay ocasiones en las que se desearía,
  - Poder avanzar ilimitadamente
  - Del ultimo nodo, pasar al primero
    - **last.next = first**



# EL TDA LISTA CIRCULAR

No es necesario mantener un control de primero y ultimo

Solo con el **último** ya se tiene todo, pues el siguiente al ultimo es el primero

```
class LCENodo<E> {  
    E content;  
    LCENodo<E> next;  
}
```

```
class LCE<E> {  
    LCENodo<E> last;  
}
```

# CREAR NODO

- Para crear un nodo valido

```
public LCENodo(T data){  
    this.data = data;  
    this.next = this;  
}
```



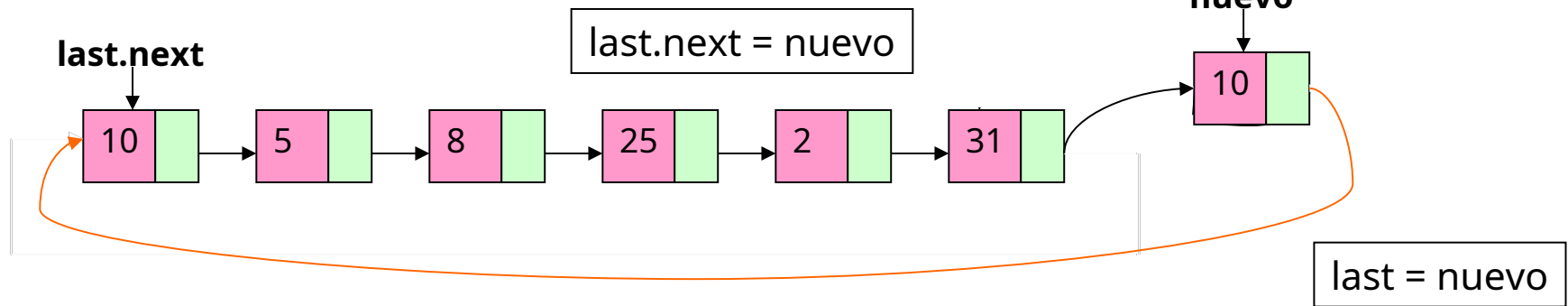
# Método: getPrevious

Busca el anterior a un nodo dado

```
private LCENodo<E> getPrevious (LCENodo<E> p){  
    if (isEmpty() || p == null)  
        return null;  
    LCENodo<E> q;  
    q = this.last;  
    do {  
        if (q.next == p) return q;  
        q = q.next;  
    } while (q != this.last);  
    return null;  
}
```



# Inserción al Inicio y al Final



```
boolean insertarNodeInicio( LCENodo<E>  
nuevo ){
```

```
    if (nuevo == null) return false;
```

```
    if (isEmpty())
```

```
        this.last = nuevo;
```

```
    else {
```

```
        nuevo.next = this.last.next;
```

```
        this.last.next = nuevo;
```

```
    }
```

```
    return true;
```

```
}
```

```
boolean insertarNodeFin(LCENodo<E>  
nuevo){
```

```
    if (nuevo == null) return false;
```

```
    if (isEmpty()) this.last = nuevo;
```

```
    else {
```

```
        nuevo.next = this.last.next;
```

```
        this.last.next = nuevo;
```

```
    }
```

```
    this.last = nuevo;
```

```
    return true;
```

# LISTAS CIRCULARES DOBLEMENTE ENLAZADAS (LCDE)

- Es una implementacion de listas circulares
  - Con nodos que tienen dos punteros
- Asi se recorre la lista en el
  - Sentido del avance del reloj y
  - En sentido contrario
- Seguimos llevando un control solo del ultimo nodo de la lista

