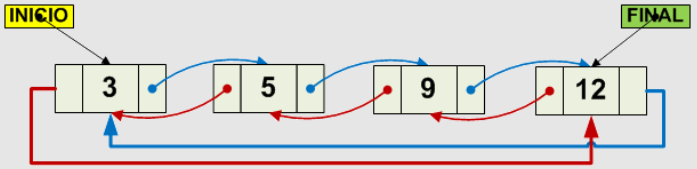
Técnico Informático

ESTRUCTURA DE DATOS

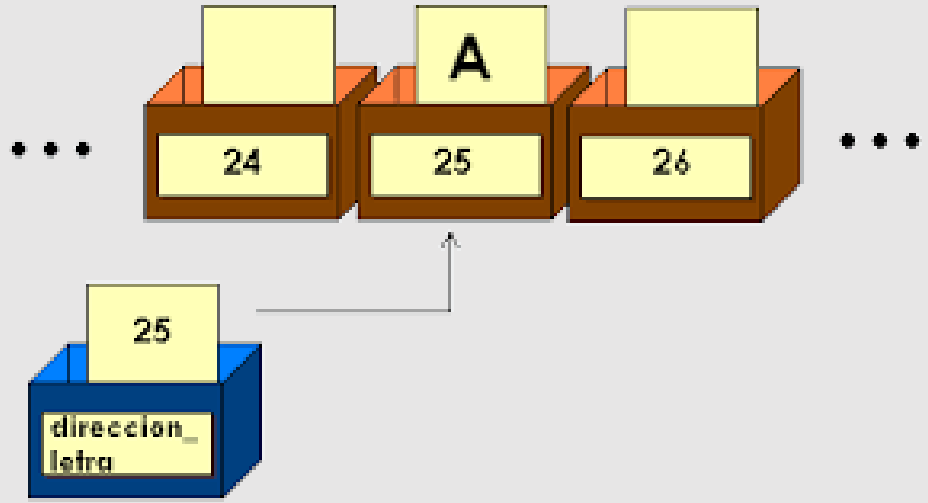
UNIDAD I: LISTAS DOBLES



Escuela de Minas "Dr. Horacio Carrillo"
Universidad Nacional de Jujuy

Índice

- Definición de Lista Doble
- Operaciones fundamentales
- Implementación
- Aplicaciones



Definición (1)

- Una lista doble es una colección de ***nodos*** ordenada según su posición, cuyo acceso/recorrido se realiza mediante ***punteros*** que enlazan los nodos.
- Una lista es una estructura lineal en la que los elementos (nodos) se disponen de tal forma que cada uno tiene un predecesor y un sucesor , salvo el primero y el último.

Definición (2)

- En una lista doble cada nodo se representa como un registro con 3 campos esenciales:
  - Campo de **datos** (tipos de datos simples o compuestos)
  - Campo puntero **siguiente** (un puntero hacia el siguiente nodo en la lista)
  - Campo puntero **anterior** (un puntero hacia el nodo predecente en la lista)

Definición (3)

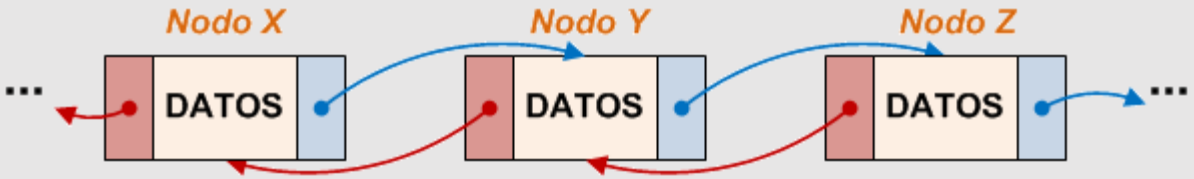
nodo=REGISTRO

datos: tipo_dato (simple, compuesto)

anterior: puntero a nodo

siguiente: puntero a nodo

FIN_REGISTRO

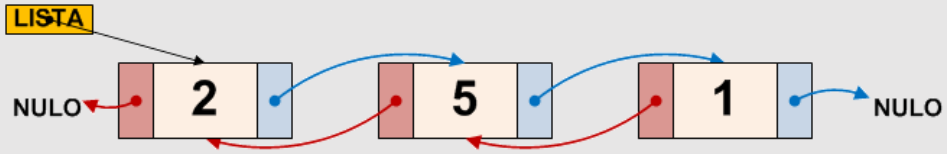


Operaciones Fundamentales

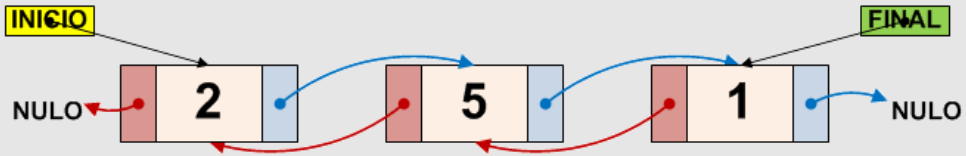
- Sobre una lista doble se definen las siguientes operaciones:
  - Iniciar lista
  - Crear nodo
  - Agregar nodo
    - ✓ agregar_inicio
    - ✓ agregar_final
    - ✓ agregar en orden
  - Quitar nodo
    - ✓ quitar_inicio
    - ✓ quitar_final
    - ✓ quitar_nodo_puntual
  - Mostrar (recorrido de la lista)
  - Buscar un valor en la lista

Alternativas de Implementación

- La implementación del TDA lista requiere de la definición de los nodos (registros) y punteros que permitan acceder a la lista. La implementación puede presentar 2 variantes:
  - Un puntero al inicio de la lista



  - Un puntero al inicio y otro al final de la lista



Implementación (1)

- Implementación del nodo y los punteros a la lista.

```
typedef struct tnodo *pnodo;
typedef struct tnodo{
                        int dato;
                        pnodo ant;
                        pnodo sig;
                      };
typedef struct tlista{
                        pnodo inicio;
                        pnodo final;
                      };
```

pnodo: tipo puntero que referencia registros *tnodo*.

ant: puntero (*pnodo*) que enlaza con el nodo anterior.

sig: puntero (*pnodo*) que enlaza con el nodo siguiente.

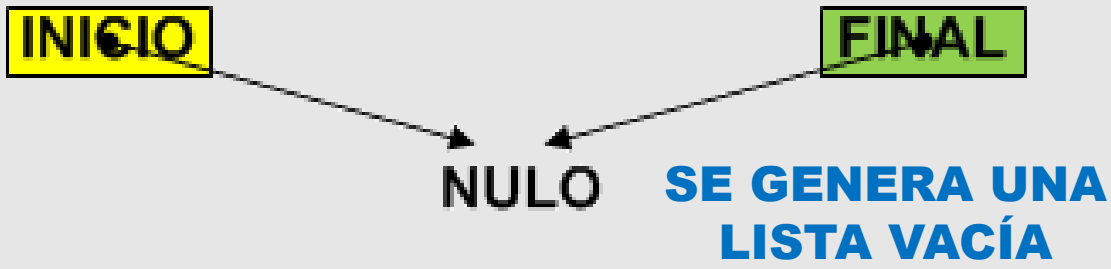
inicio: puntero al primer elemento de la lista.

final: puntero al último elmento de la lista.

Implementación (2)

- Operación *iniciar lista*

```
void inicia_lista(pnodo &lista)
{
      lista.inicio=NULL;
      lista.final=NULL;
}
```



Implementación (3)

- Operación *crear nodo*

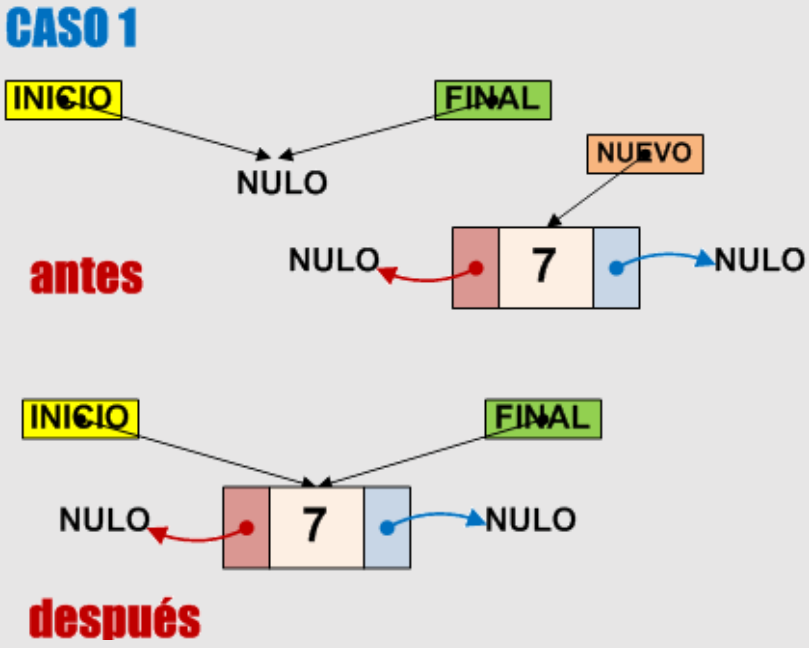
```
void crear(pnodo &nuevo)
{
  nuevo=new tnodo;
  if (nuevo!=NULL)
     { cout << "Ingrese valor: ";
       cin >> nuevo->dato;
      nuevo->ant=NULL;
      nuevo->sig=NULL;
     }
  else
     cout << "MEMORIA INSUFICIENTE" << endl;
}
```

¿Cómo se usa crear_nodo?

```
crear(nuevo);
if (nuevo!=NULL)
 agregar_inicio(milista,nuevo);
```

Implementación (4)

- Operación *agregar al inicio*
  - ***Caso 1:*** **agregar un nodo a una lista vacía**
  - *Caso 2:* agregar un nodo a una lista con elementos

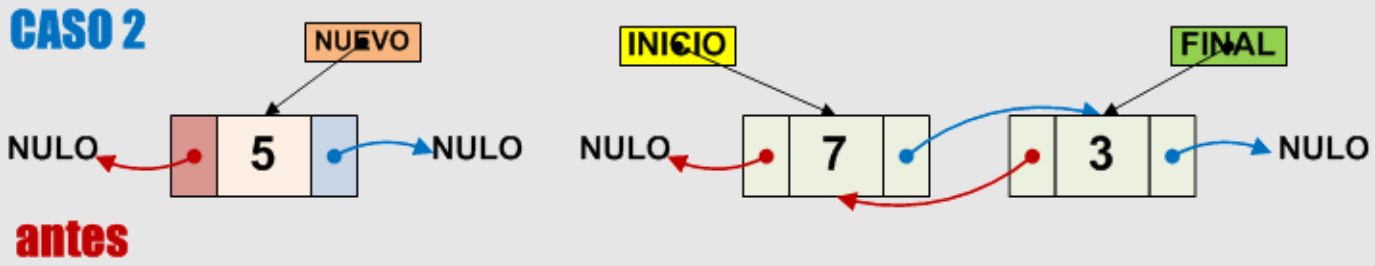


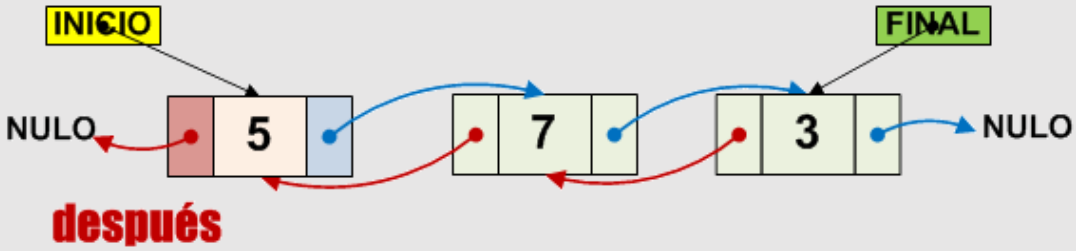
¿Cómo se agrega el nuevo nodo?

```
lista.inicio=nuevo;
lista.final=nuevo;
```

Implementación (5)

- Operación *agregar al inicio*
  - *Caso 1:* agregar un nodo a una lista vacía
  - *Caso 2:* agregar un nodo a una lista con elementos





¿Cómo se conecta el nuevo nodo?

```
nuevo->sig=lista.inicio;
lista.inicio->ant=nuevo;
lista.inicio=nuevo;
```

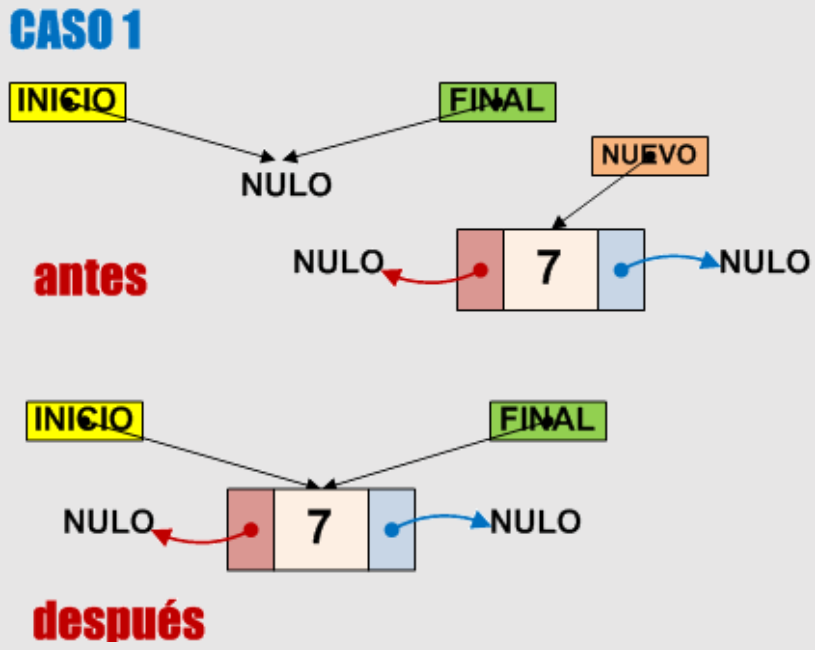
Implementación (6)

- Operación *agregar al inicio*

```
void agregar_inicio(tlista &lista,pnodo nuevo)
{ if (lista.inicio==NULL)
    { lista.inicio=nuevo;
      lista.final=nuevo; }
  else
    { nuevo->sig=lista.inicio;
      lista.inicio->ant=nuevo;
      lista.inicio=nuevo;
    }
}
```

Implementación (7)

- Operación *agregar al final*
  - ***Caso 1:*** **agregar un nodo a una lista vacía**
  - *Caso 2:* agregar un nodo a una lista con elementos

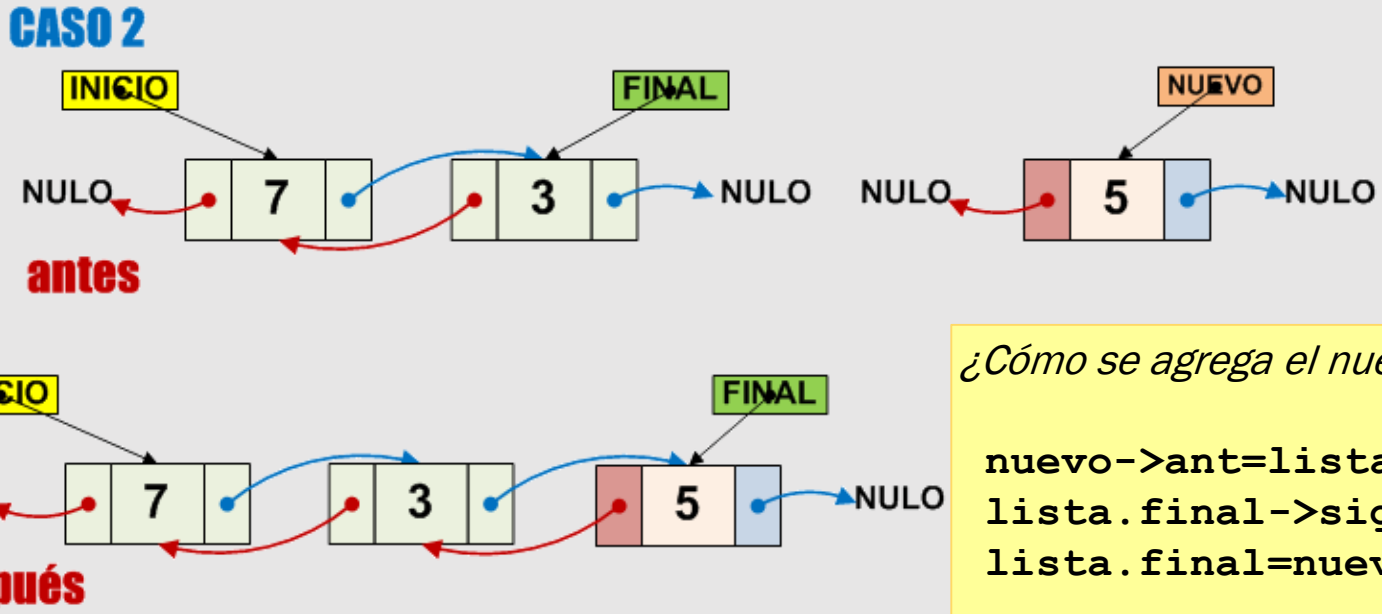


¿Cómo se agrega el nuevo nodo?

```
lista.inicio=nuevo;
lista.final=nuevo;
```

Implementación (8)

- Operación *agregar al final*
  - *Caso 1:* agregar un nodo a una lista vacía
  - *Caso 2:* agregar un nodo a una lista con elementos



¿Cómo se agrega el nuevo nodo?

```
nuevo->ant=lista.final;
lista.final->sig=nuevo;
lista.final=nuevo;
```

Implementación (9)

- Operación *agregar al final*

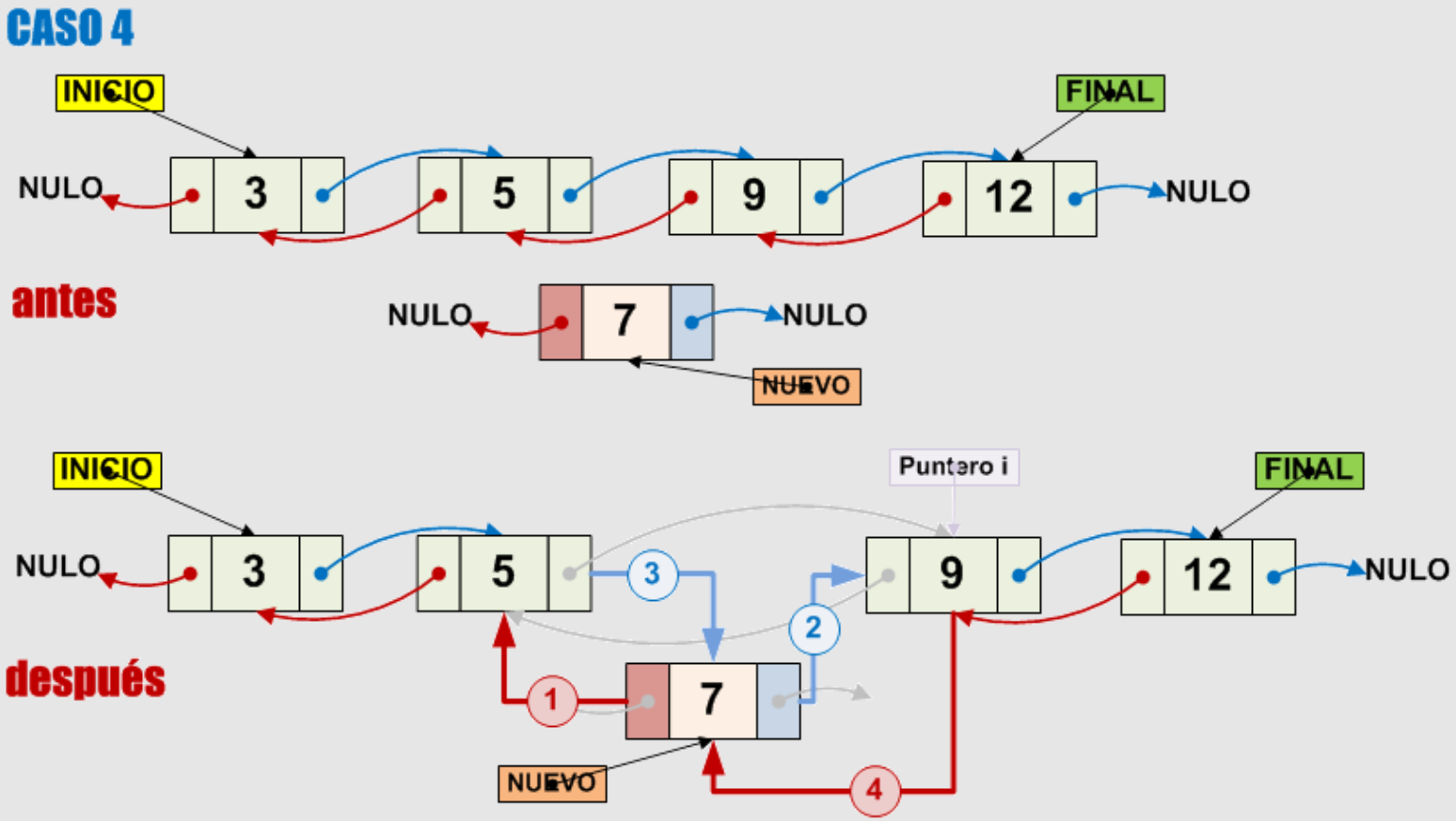
```
void agregar_final(tlista &lista,pnodo nuevo)
{ if (lista.inicio==NULL)
    { lista.inicio=nuevo;
      lista.final=nuevo; }
  else
    { nuevo->ant=lista.final;
      lista.final->sig=nuevo;
      lista.final=nuevo;
    }
}
```

Implementación (10)

- Operación *agregar en orden*
  - *Caso 1:* agregar un nodo a una lista vacía
  - *Caso 2:* agregar un elemento menor que el primero de la lista (*agregar_inicio*)
  - *Caso 3*: agregar un elemento mayor que el último de la lista (*agregar_final*)
  - *Caso 4*: agregar un elemento en el medio de la lista.

Implementación (11)

- Operación *agregar en orden* (caso 4)



Implementación (12)

- Operación *agregar en orden* (caso 4)

```
for(i=lista.inicio;i->sig!=NULL && nuevo->dato > i->dato;
i=i->sig);
```

```
nuevo->ant=i->ant;
nuevo->sig=i;
```

```
(i->ant)->sig=nuevo;
i->ant=nuevo;
```

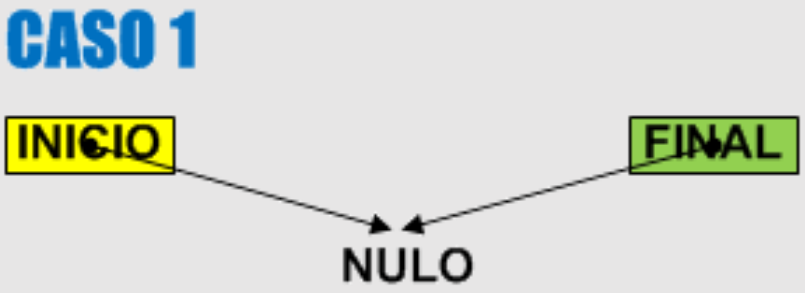
Se recorre la lista hasta encontrar el lugar de inserción.

Los nodos antecesor y sucesor se conectan al nuevo nodo.

Se conecta el nuevo nodo a los nodos antecesor y sucesor en la lista.

Implementación (13)

- Operación *quitar del inicio*
  - ***Caso 1:*** **Quitar un nodo de una lista vacía**
  - *Caso 2*: Quitar un nodo de una lista con 1 elemento
  - *Caso 3*: Quitar un nodo de una lista con 2 o más elementos

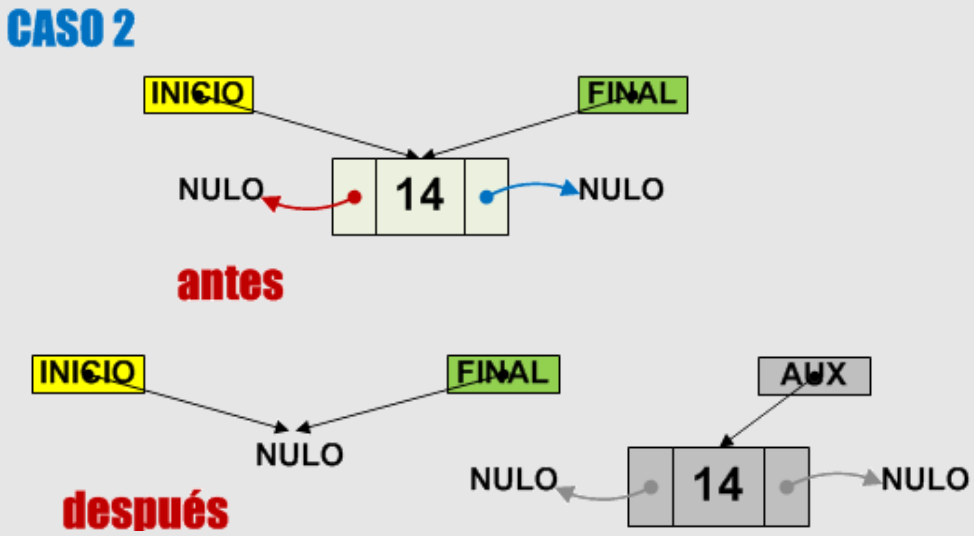


¿Cuál es el resultado de quitar_inicio si la lista está vacía?

```
return NULL;
```

Implementación (14)

- Operación *quitar del inicio*
  - *Caso 1:* Quitar un nodo de una lista vacía
  - ***Caso 2*: Quitar un nodo de una lista con 1 elemento**
  - *Caso 3*: Quitar un nodo de una lista con 2 o más elementos

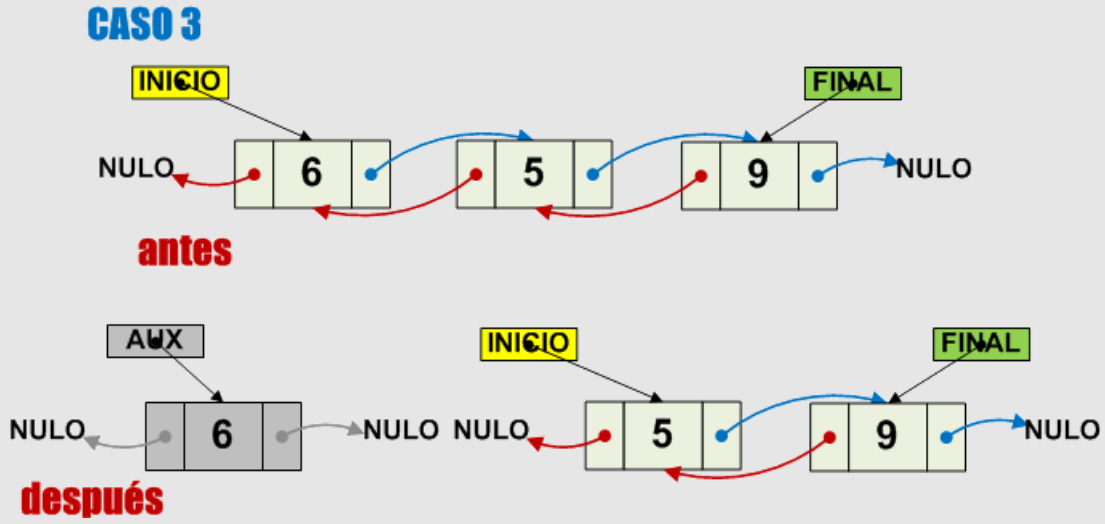


¿Cuál es el resultado de quitar_inicio si la lista tiene un solo elemento?

```
aux=lista.inicio;
lista.inicio=NULL;
lista.final=NULL;
return aux;
```

Implementación (15)

- Operación *quitar del inicio*
  - *Caso 1:* Quitar un nodo de una lista vacía
  - *Caso 2*: Quitar un nodo de una lista con 1 elemento
  - ***Caso 3*: Quitar un nodo de una lista con 2 o más elementos**



¿Cuál es el resultado de quitar_inicio si la lista tiene varios elementos?

```
aux=lista.inicio;
lista.inicio=aux->sig;
aux->sig=NULL;
lista.inicio->ant=NULL;
return aux;
```

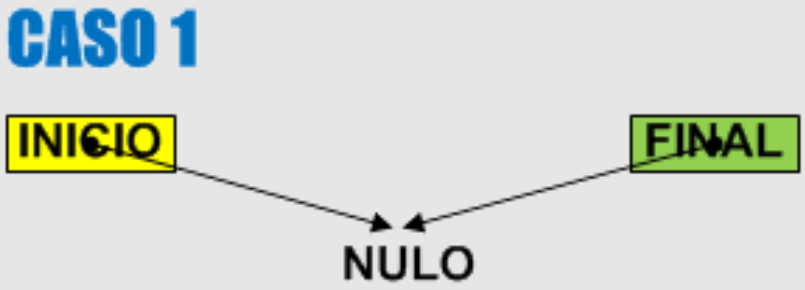
Implementación (16)

- Operación *quitar del inicio*

```
pnodo quitar_inicio(tlista &lista)
{pnodo aux;
 if (lista.inicio==NULL)
   aux=NULL;
 else
  if (lista.inicio==lista.final)
   {aux=lista.inicio;
    lista.inicio=NULL;
    lista.final=NULL; }
else
 {aux=lista.inicio;
  lista.inicio=aux->sig;
  aux->sig=NULL;
  lista.inicio->ant=NULL;
 }
 return aux;
}
```

Implementación (17)

- Operación *quitar del final*
  - ***Caso 1:*** **Quitar un nodo de una lista vacía**
  - Caso 2: Quitar un nodo de una lista con 1 elemento
  - *Caso 3*: Quitar un nodo de una lista con 2 o más elementos

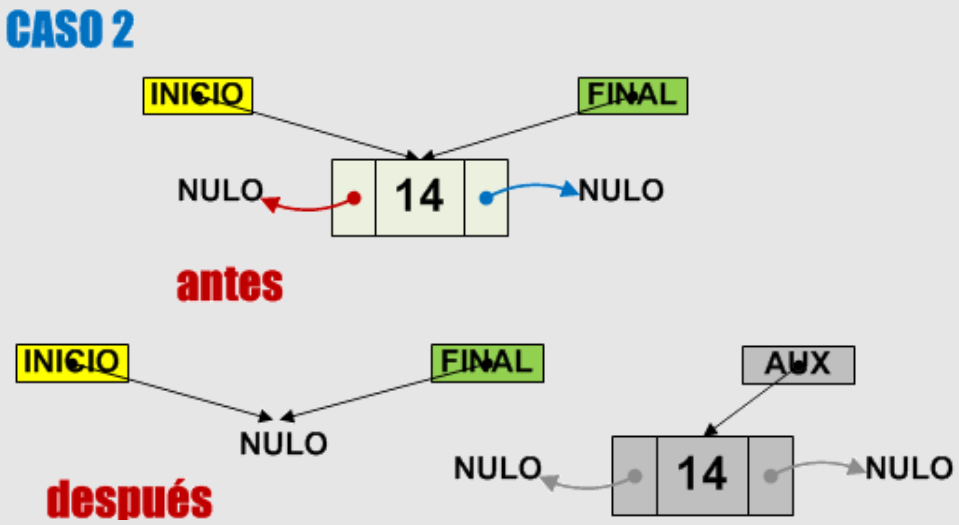


¿Cuál es el resultado de quitar_final si la lista está vacía?

```
return NULL;
```

Implementación (18)

- Operación *quitar del final*
  - *Caso 1:* Quitar un nodo de una lista vacía
  - ***Caso 2*: Quitar un nodo de una lista con 1 elemento**
  - *Caso 3*: Quitar un nodo de una lista con 2 o más elementos

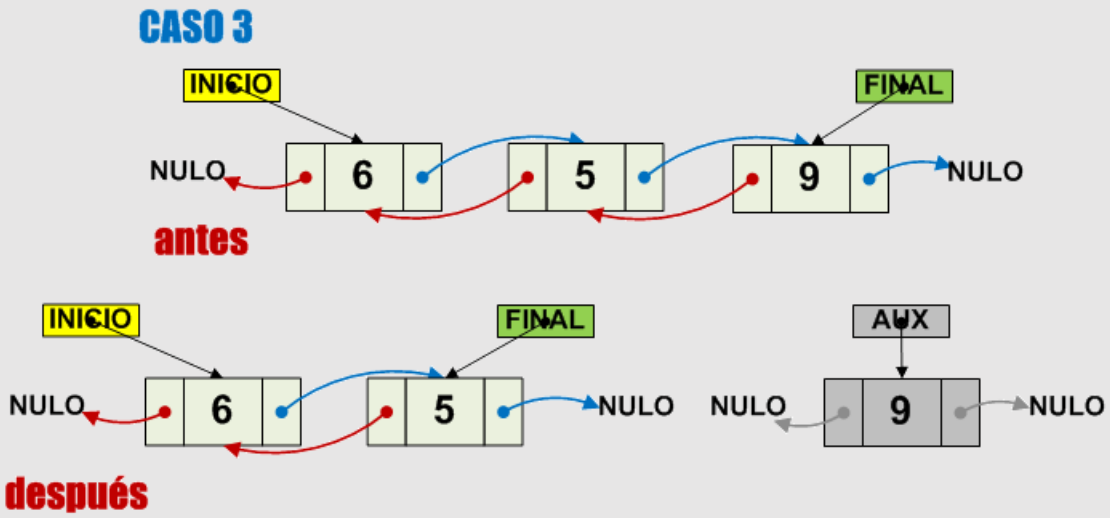


¿Cuál es el resultado de quitar_final si la lista tiene un solo elemento?

```
aux=lista.inicio;
lista.inicio=NULL;
lista.final=NULL;
return aux;
```

Implementación (19)

- Operación *quitar del final*
  - *Caso 1:* Quitar un nodo de una lista vacía
  - *Caso 2*: Quitar un nodo de una lista con 1 elemento
  - ***Caso 3*: Quitar un nodo de una lista con 2 o más elementos**



¿Cuál es el resultado de quitar_final si la lista tiene varios elementos?

```
aux=lista.final;
lista.final=aux->ant;
aux->ant=NULL;
lista.final->sig=NULL;
return aux;
```

Implementación (20)

- Operación *quitar del final*

```
pnodo quitar_final(tlista &lista)
{pnodo aux;
 if (lista.inicio==NULL)
   aux=NULL;
 else
  if (lista.inicio==lista.final)
    {aux=lista.inicio;
     lista.inicio=NULL;
     lista.final=NULL;
    }
else
   {aux=lista.final;
    lista.final=aux->ant;
    aux->ant=NULL;
    lista.final->sig=NULL;
   }
   return aux;
}
```

Implementación (21)

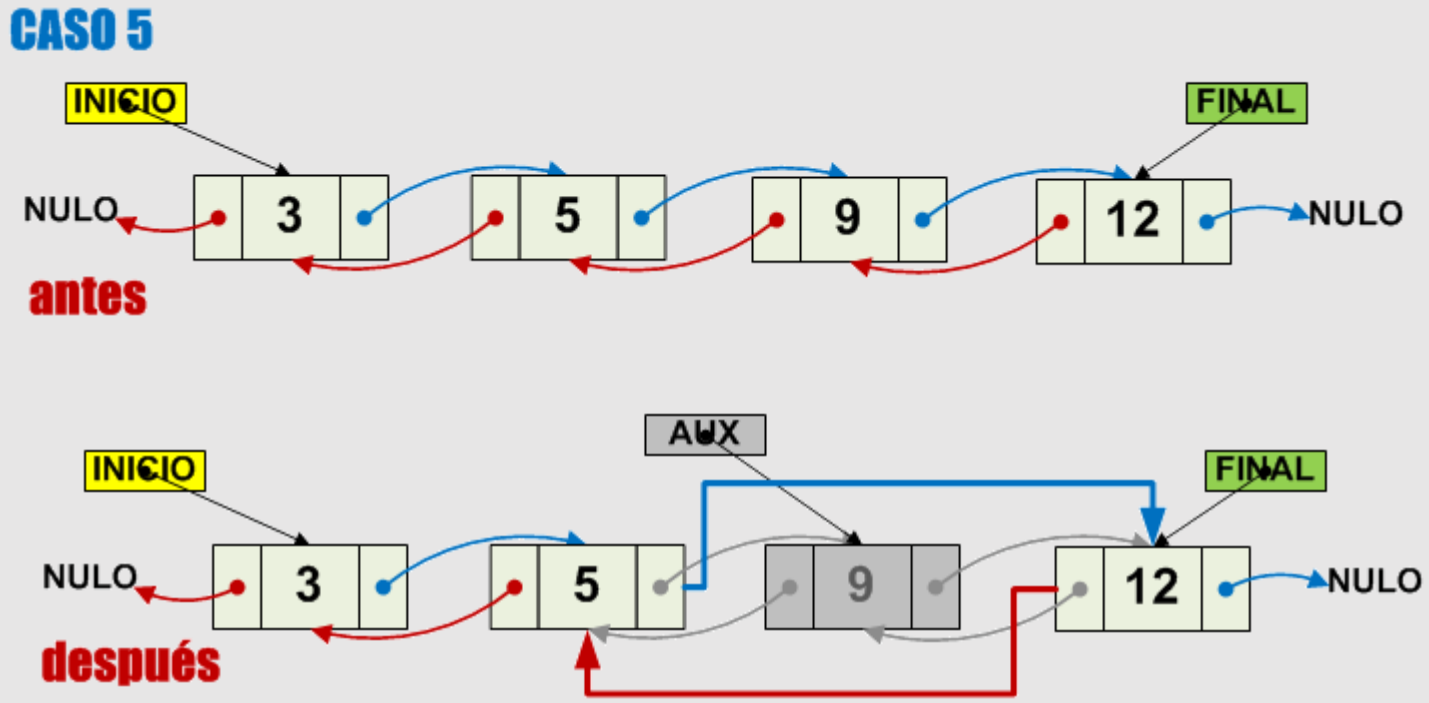
- Operación *quitar un nodo específico*
  - *Caso 1:* Quitar un nodo de una lista vacía
  - *Caso 2*: Quitar el único nodo de la lista

  } *quitar_inicio* y *quitar_final*

  - *Caso 3*: Quitar el primer nodo (*quitar_inicio*)
  - *Caso 4*: Quitar el último nodo (*quitar_final*)
  - ***Caso 5*: Quitar un nodo del medio de la lista**

Implementación (22)

- Operación *quitar un nodo específico* (caso 5)



Implementación (23)

- Operación *quitar un nodo específico* (caso 5)

```
for(i=lista.inicio;i->sig!=NULL && valor!=i->dato;i=i->sig);
```

```
if (i->sig!=NULL)
  {aux=i;
   (i->ant)->sig=i->sig;
   (i->sig)->ant=i->ant;
   aux->ant=NULL;
   aux->sig=NULL;
  }
else
  aux=NULL;
```

Se recorre la lista en busca del valor

Se desconecta el nodo que tiene el valor buscado

Resultado NULO si el valor no está en la lista

Implementación (24)

- Operación *mostrar datos de la lista*

```
void mostrar(tlista lista)
{ pnodo i;
  if (lista.inicio!=NULL)
    for(i=lista.inicio;i!=NULL;i=i->sig)
    cout << "Nodo: " << i->dato << endl;
  else
    cout << "LISTA VACIA";
}
```

Implementación (25)

- Operación *buscar un dato en la lista*

```
bool buscar_nodo(tlista L, int valor)
{pnodo i;
 bool encontrado=false;
 if (L.inicio!=NULL)
  for(i=L.inicio;i!=NULL && encontrado==false;i=i->sig)
    if (i->dato==valor)
      encontrado=true;
  return encontrado;
}
```

Bibliografía

- Joyanes Aguilar *et al*. Estructuras de Datos en C++. Mc Graw Hill. 2007.
- De Giusti, Armando *et al.* Algoritmos, datos y programas, conceptos básicos. Editorial Exacta. 1998.
- Joyanes Aguilar, Luis. Fundamentos de Programación. Mc Graw Hill. 1996.
- Hernández, Roberto *et al*. Estructuras de Datos y Algoritmos. Prentice Hall. 2001.