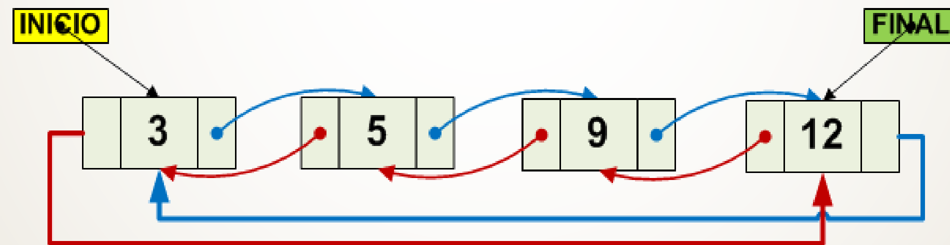
Analista Programador Universitario

Estructura de Datos

Unidad I: Listas Dobles



Facultad de Ingeniería
Universidad Nacional de Jujuy

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)

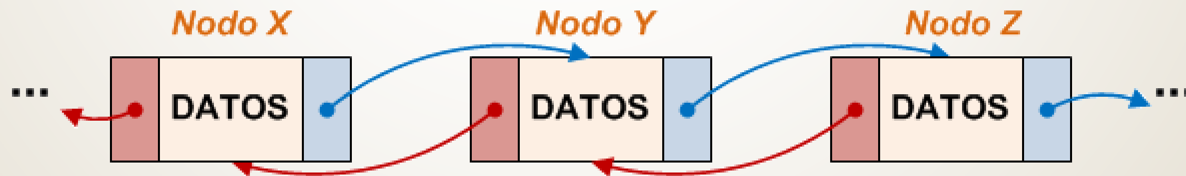
nodo=REGISTRO

datos: tipo_dato (simple, compuesto)

anterior: puntero a nodo

siguiente: puntero a nodo

FIN_REGISTRO

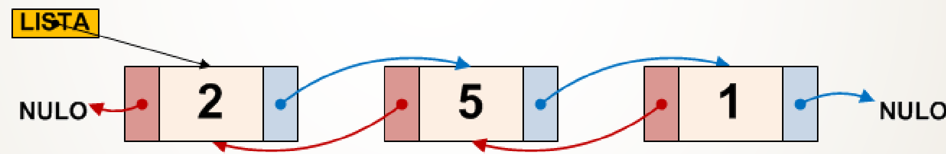


Operaciones Fundamentales

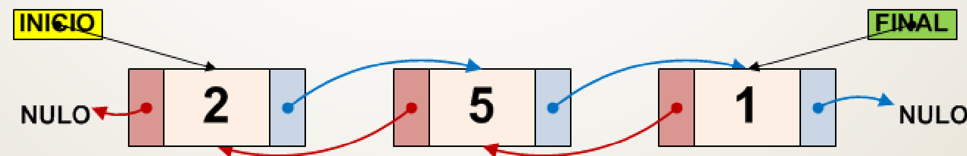
- Sobre una lista doble se definen las siguientes operaciones:
  - Iniciar lista
  - Crear nodo
  - Agregar nodo
    - ✓ agregar_inicio
    - ✓ agregar_final
    - ✓ agregar en orden
  - Quitar nodo
    - ✓ quitar_inicio
    - ✓ quitar_final
    - ✓ quitar_nodo_puntual
  - Mostrar (recorrido de la lista)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

Para crear un lista vacía se asigna NULO a los punteros de la lista

inicio

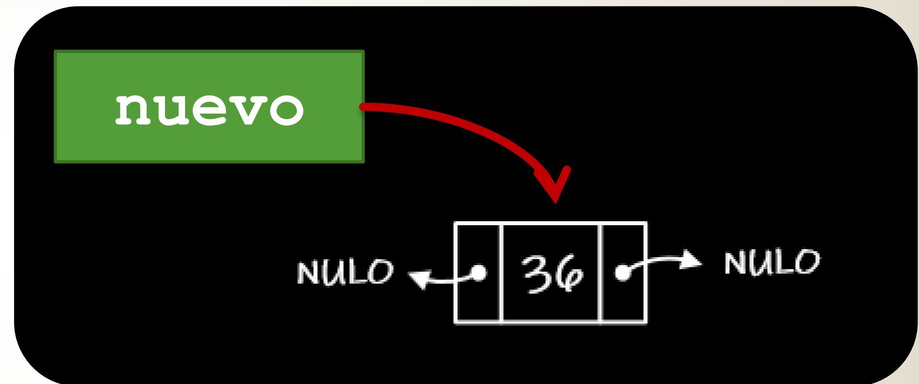
final

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

Implementación (3)

- Operación *crear nodo*

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



¿Cómo se usa crear_nodo?

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

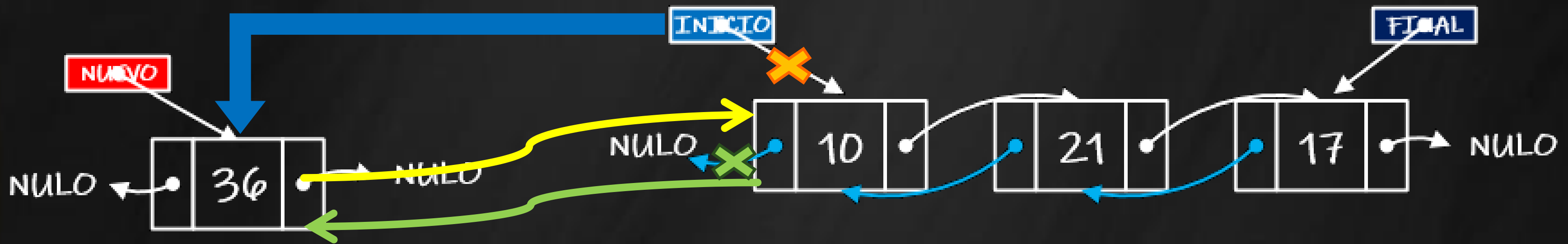
Operación agregar al inicio

Permite agregar un nodo al comienzo de la lista

Caso 1: Lista Vacía

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

Caso 2: Lista con elementos




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

Implementación (4)

- 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;
    }
}
```

Lista Vacía

Lista con elementos

Operación agregar al final

Permite agregar un nodo al final de la lista

Caso 1: Lista Vacía

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

Caso 2: Lista con elementos

¿RECORRIDO?

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

Implementación (5)

- Operación *agregar al final*

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

Lista Vacía

Lista con elementos

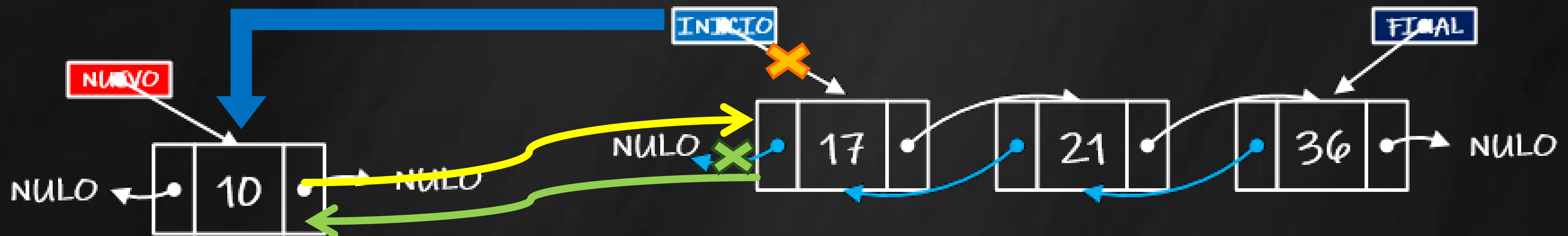
Operación agregar en orden

Permite agregar un nodo a la lista con un criterio de orden

Caso 1: Lista Vacía

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

Caso 2: el nuevo valor es menor o igual que el primer elemento de la lista



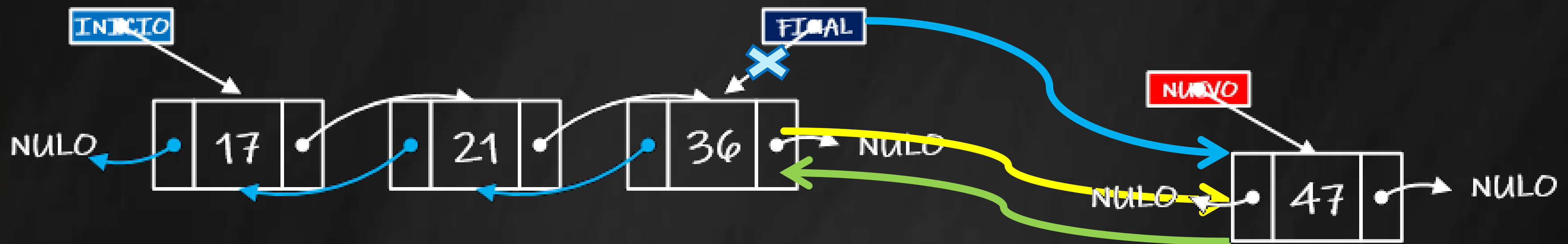
10<=17?

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

Operación agregar en orden

Caso 3: el nuevo valor es mayor o igual que el último elemento de la lista

Permite agregar un nodo a la lista con un criterio de orden



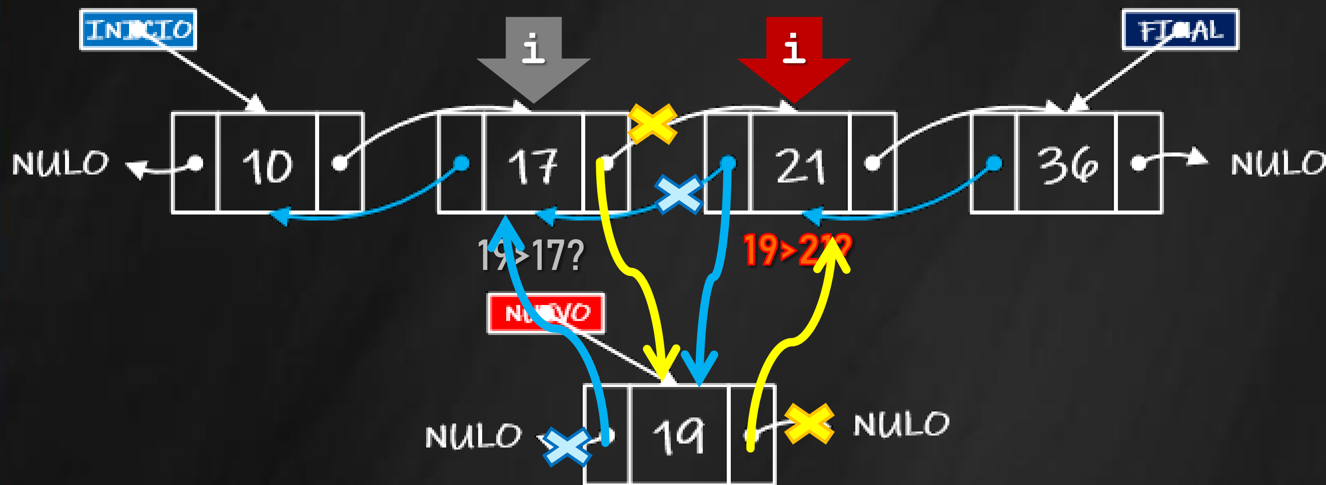
47>=36?

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

Operación agregar en orden

Caso 4: el nuevo valor debe ubicarse en el medio de la lista

Permite agregar un nodo a la lista con un criterio de orden




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

Implementación (6)

- Operación *agregar en orden*

```
void agregar_orden(tlista &lt, pnodo nuevo)
{ pnodo i;
  if (lt.inicio==NULL)
     // caso 1: lista vacía
  else
   if (nuevo->dato <= lt.inicio->dato)
       // caso 2: nuevo dato menor que el primer nodo (agregar_inicio)
   else
     if (nuevo->dato >= lt.final->dato)
          // caso 3: nuevo dato mayor que el último nodo (agregar final)
     else
     { for(i=lt.inicio->sig;i!=lt.final && 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.

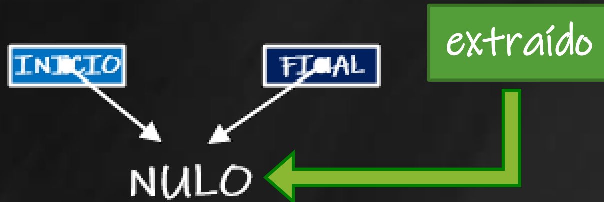
Se conecta el nuevo nodo al antecesor y sucesor en la lista.

Los nodos antecesor y sucesor se conectan al nuevo nodo.

Operación quitar inicio

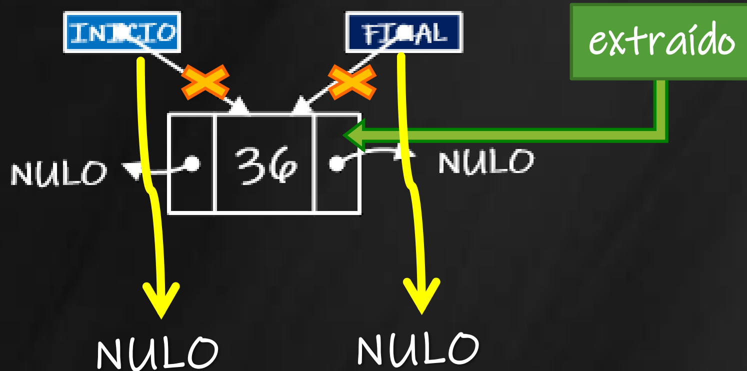
Permite extraer el primer nodo de la lista.

Caso 1: Lista Vacía




```
extraido=NULL;
```

Caso 2: Lista con un único elemento

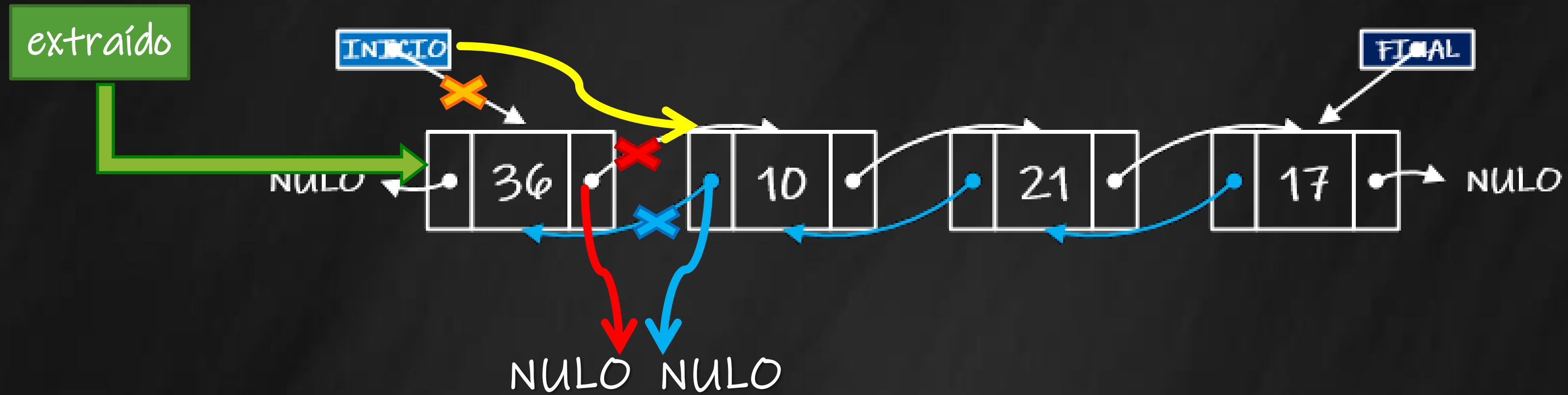



```
extraido=lista.inicio;
lista.inicio=NULL;
lista.final=NULL;
```

Operación quitar inicio

Permite extraer el primer nodo de la lista.

Caso 3: Lista con 2 o más elementos




```
extraido=lista.inicio;
lista.inicio=lista.inicio->sig;
lista.inicio->ant=NULL;
extraido->sig=NULL;
```

Implementación (7)

- Operación *quitar del inicio*

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

Extracción de lista vacía

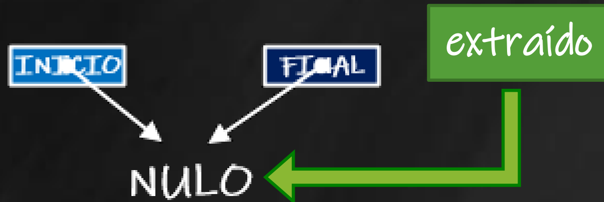
Extracción del único nodo

Extracción del primer nodo

Operación quitar final

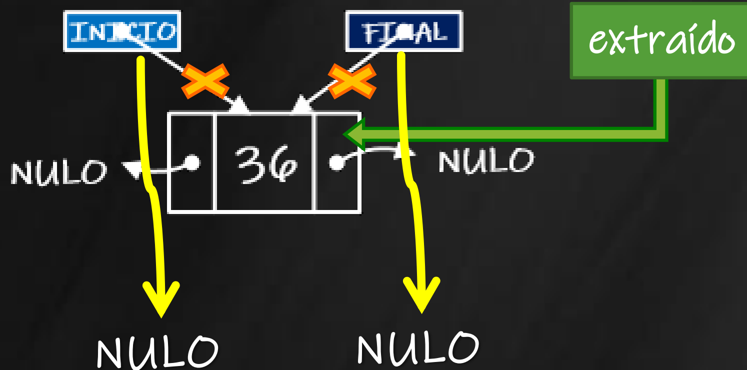
Permite extraer el último nodo de la lista.

Caso 1: Lista Vacía




```
extraido=NULL;
```

Caso 2: Lista con un único elemento

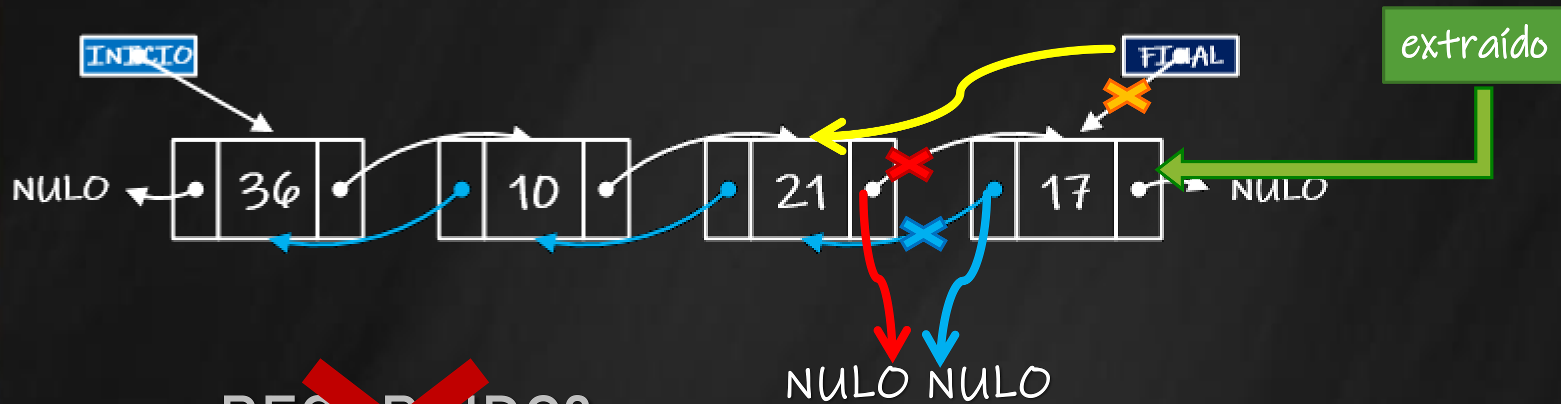



```
extraido=lista.inicio;
lista.inicio=NULL;
lista.final=NULL;
```

Operación quitar final

Permite extraer el último nodo de la lista.

Caso 3: Lista con 2 o más elementos



¿RECORRIDO?

```
extraido=lista.final;
lista.final=lista.final->ant;
lista.final->sig=NULL;
extraido->ant=NULL;
```

Implementación (8)

- Operación *quitar del final*

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

Extracción de lista vacía

Extracción del único nodo

Extracción del último nodo

Operación quitar nodo específico

Permite extraer un nodo de la lista que contiene un valor específico.

Caso 1: Extracción de lista vacía

```
extraido=NULL;
```

Caso 2: Extracción del único elemento de la lista (es el buscado)

```
extraido=lista.inicio;
lista.inicio=NULL;
lista.final=NULL;
```

Caso 3: El valor a extraer coincide con el dato del primer nodo (quitar inicio)

```
extraido=lista.inicio;
lista.inicio=lista.inicio->sig;
lista.inicio->ant=NULL;
extraido->sig=NULL;
```

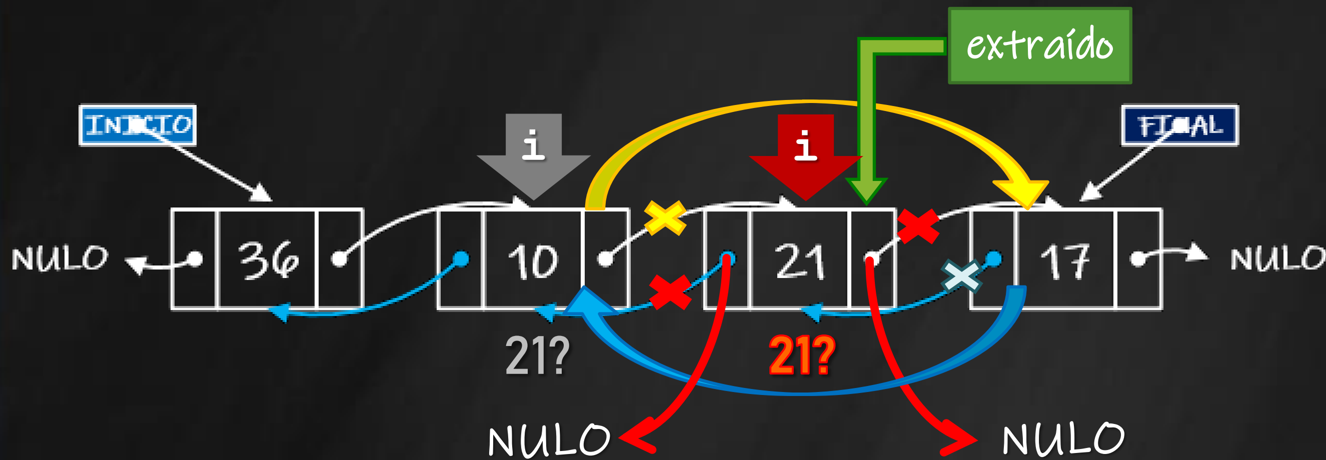
Caso 4: El valor a extraer coincide con el dato del último nodo (quitar final)

```
extraido=lista.final;
lista.final=lista.final->ant;
lista.final->sig=NULL;
extraido->ant=NULL;
```

Operación quitar nodo específico

Permite extraer un nodo de la lista que contiene un valor específico.

Caso 5: Extracción de un valor del medio de la lista




```
for( i=lista.inicio->sig ;i != lista.final && i->dato != buscado ;  i=i->sig );
 extraido=i;
 (i->ant)->sig=extraido->sig;
 (i->sig)->ant=extraido->ant;
 extraido->sig=NULL;
 extraido->ant=NULL;
```

Sólo si i != lista.final

¿Qué ocurre si el valor no pertenece a la lista?

Implementación (9)

- Operación *quitar un nodo específico*

```
pnodo quitar_nodo(tlista &lt, int buscado)
{ pnodo i, extraido;
  if (lt.inicio==NULL)
      // caso 1: extracción de lista vacía
  else
    if (lt.inicio->dato==buscado)
     {  if (lt.inicio==lt.final)
           // caso 2: extracción del único elemento
        else
           // caso 3: extracción del primer elemento
     }
    else
      if (lt.final->dato==buscado)
          // caso 4: extracción del último elemento
      else
        // caso 5: extracción de un elemento del medio de la lista
  return extraido;
}
```

Implementación (10)

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

```
                              . . .
for(i=lt.inicio->sig;i!=lt.final && buscado!=i->dato;i=i->sig);
if (i!=lista.final)
{   extraido=i;
    (i->ant)->sig=extraido->sig; // i->sig
    (i->sig)->ant=extraido->ant; // i->ant
     aux->ant=NULL;
     aux->sig=NULL;
}
else
     extraido=NULL;
```

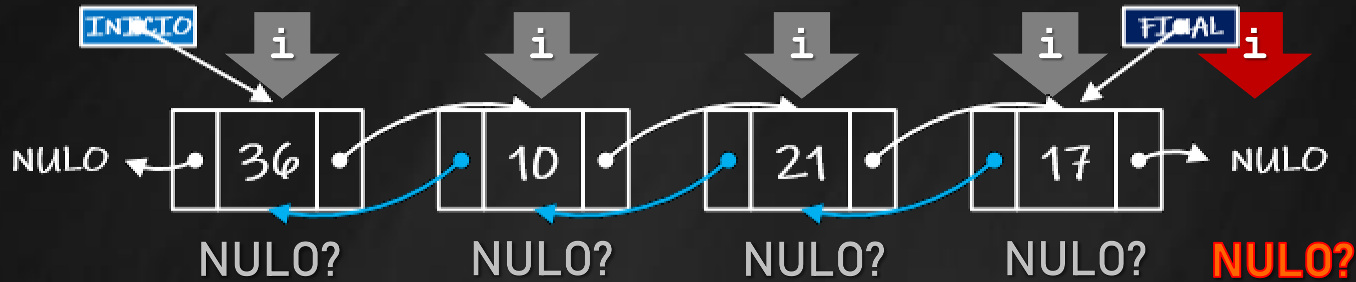
Se recorre la lista comparando cada nodo con el valor buscado

Se desconecta el nodo que tiene el valor buscado

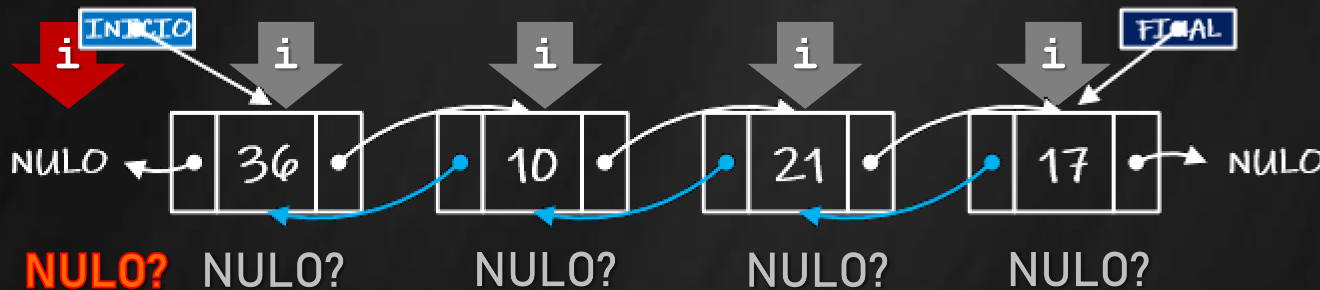
Resultado NULO si el valor no está en la lista

Operación mostrar lista

Permite mostrar nodo a nodo el contenido de una lista.



36 10 21 17



17 21 10 36

Implementación (11)

- Operación *mostrar datos de la lista*

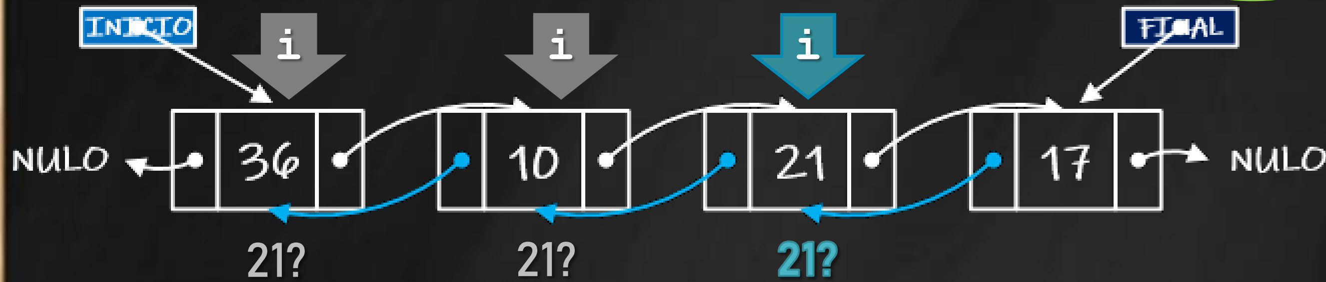
```
void mostrar_1(tlista lista)
{ pnodo i;
  if (lista.inicio!=NULL)
    for(i=lista.inicio;i!=NULL;i=i->sig)
    cout << "Nodo: " << i->dato << endl;
  else
    cout << "LISTA VACIA";
}

void mostrar_2(tlista lista)
{ pnodo i;
  if (lista.inicio!=NULL)
    for(i=lista.final;i!=NULL;i=i->ant)
    cout << "Nodo: " << i->dato << endl;
  else
    cout << "LISTA VACIA";
}
```

Operación buscar dato

Determina si un valor se encuentra o no almacenado en una lista.

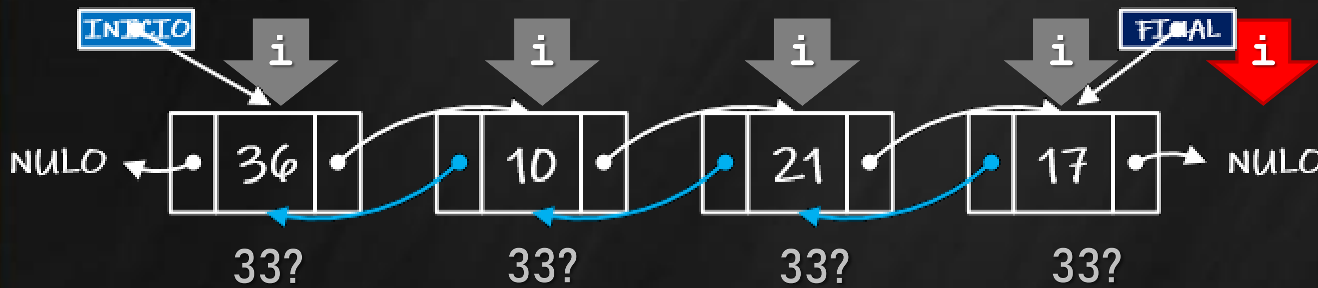
Caso Positivo: buscando un valor existente

Buscar 21



Caso Negativo: buscando un valor inexistente

Buscar 33



Implementación (12)

- Operación *buscar un dato en la lista*

```
bool buscar_nodo(tlista lt, int buscado)
{ pnodo k=NULL;
  if (lt.inicio!=NULL)
    for(k=lt.inicio;k!=NULL && k->dato!=buscado;k=k->sig);
  return k!=NULL;
}
```

Se recorre la lista comparando cada nodo con el valor buscado

Si el valor pertenece a la lista el puntero i tendrá su dirección

Si el recorrido se completa sin hallar el valor, k será NULO

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.