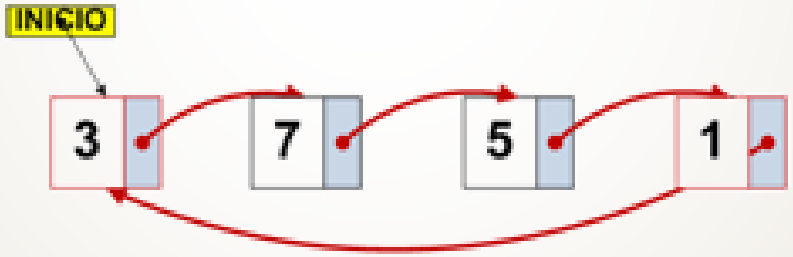
Analista Programador Universitario

Estructura de Datos

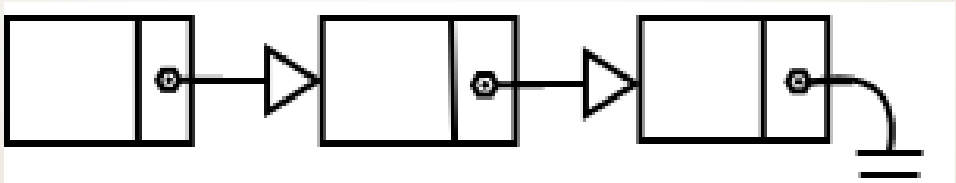
Unidad I: Listas Simples



Facultad de Ingeniería
Universidad Nacional de Jujuy

Listas Simples (1)

- Una lista simple es una colección de elementos (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.



Lista enlazada

Listas Simples (2)

- Un nodo es un registro con 2 campos esenciales:
  - Campo de datos (tipos de datos simples o compuestos)
  - Campo puntero (un puntero hacia otro nodo del mismo tipo.)

Listas Simples (3)

nodo=REGISTRO

datos: tipo_dato (simple, compuesto)

siguiente: puntero a nodo

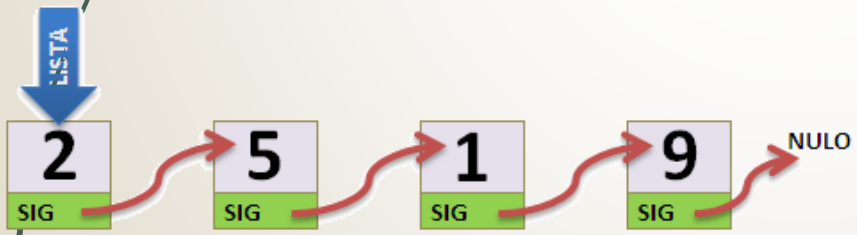
FIN_REGISTRO

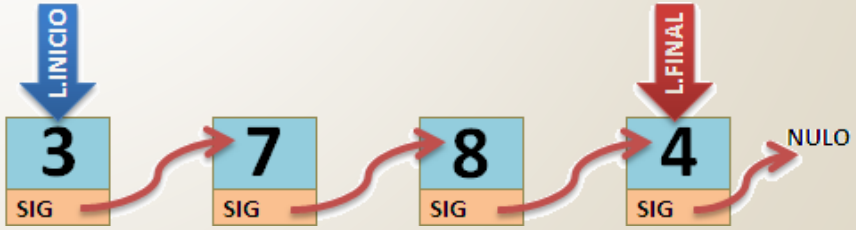
Operaciones Fundamentales

- Sobre una lista simple 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_especifico
  - Mostrar (recorrido de la lista)
  - Buscar un valor en la lista

Alternativas de Implementación

- Básicamente, la implementación del TDA lista requiere de la definición de un registro (datos y puntero a próximo elemento) y punteros que permitan acceder a la lista. La implementación puede presentar las siguientes variantes:
  - Un puntero al inicio de la lista
  - Un puntero al inicio y otro al final de la lista





Implementación (1)

- TDA lista: Implementación del tipo nodo y del puntero a éste.

```
typedef struct tnodo *pnodo;
typedef struct tnodo{
                        int dato;
                        pnodo sig;
                      };
```

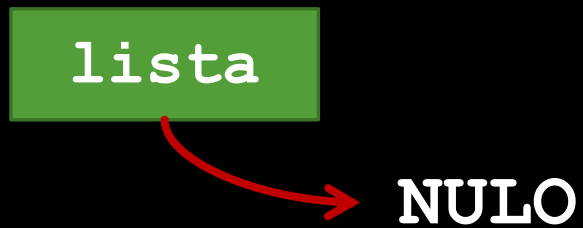
  - *pnodo*: es un tipo puntero que permite referenciar registros *tnodo*.
  - *sig*: es un campo tipo puntero (*pnodo*) que permite enlazar los nodos de una lista.

Implementación (2)

- Operación *iniciar lista*

- Iniciar lista
  - Propósito: inicializar la lista (esto genera una lista vacía).
  - Entrada: una lista (puntero de inicio de la lista).
  - Salida: una lista vacía (puntero de inicio de la lista en valor NULO).
  - Restricciones: ninguna.

Para crear un lista vacía se asigna NULO al puntero de la lista



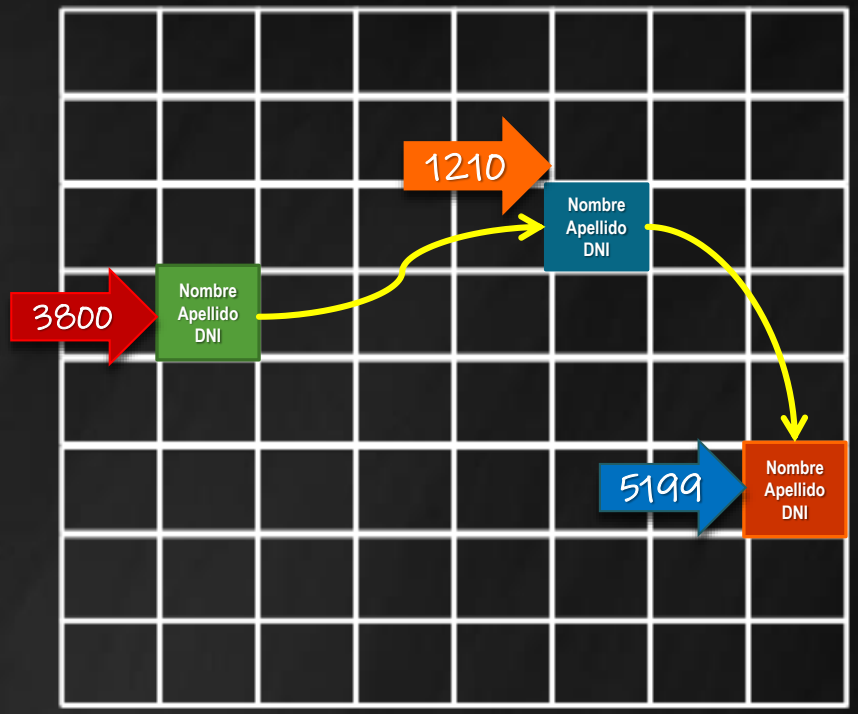

```
void inicia_lista(pnodo &lista)
{
      lista=NULL;
}
```

Operación Crear nodo

¿Cómo se crea un nodo?

Se debe solicitar un espacio de memoria usando una instrucción especial del lenguaje.

Ese espacio es accedido mediante un puntero.



¿Cómo se accede al contenido de un nodo?

Los datos de un nodo son accedidos mediante el puntero que tiene la dirección del nodo.

Por ejemplo:

```
p->nombre
P->apellido
P->DNI
```

¿Cómo conectamos estos elementos?

Implementación (3)

- Operación *crear nodo*

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

- Crear nodo
  - Propósito: crear un nuevo nodo (se reserva memoria para un nuevo elemento).
  - Entrada: un puntero a nodo.
  - Salida: un puntero con la dirección del nodo creado. Si el nodo no puede crearse, retorna NULO.
  - Restricciones: ninguna.

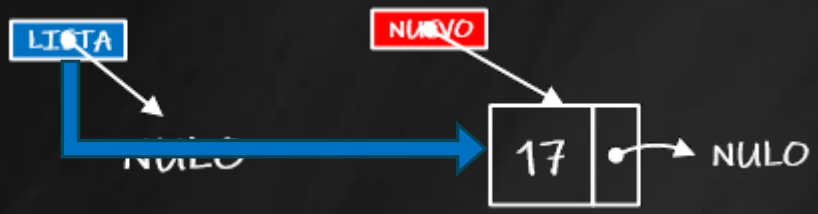
¿Cómo se usa *crear_nodo*?

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

Operación agregar al inicio

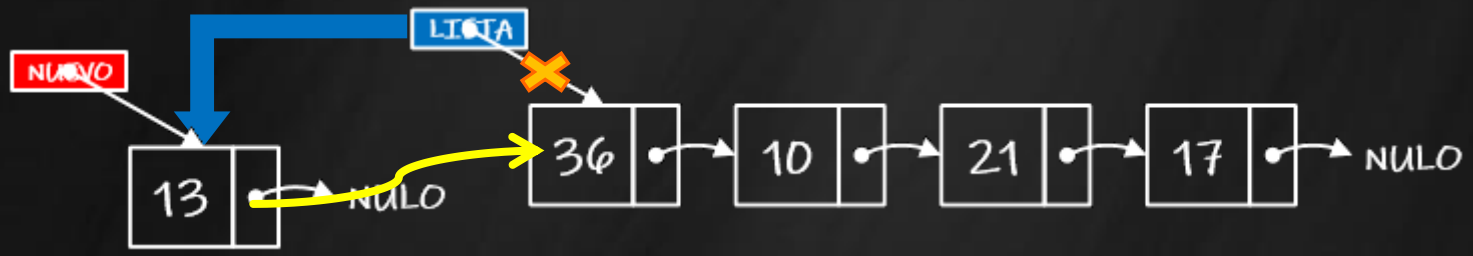
Permite agregar un nodo al comienzo de la lista

Caso 1: Lista Vacía




```
lista=nuevo;
```

Caso 2: Lista con elementos




```
nuevo->sig=lista;
lista=nuevo;
```

Implementación (4)

- Operación *agregar al inicio*

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

Lista Vacía

Lista con elementos

- Agregar un nodo al inicio
  - Propósito: agregar un nodo al inicio de la lista.
  - Entrada: una lista y un nuevo dato.
  - Salida: una lista con un nuevo nodo al principio.
  - Restricciones: una lista inicializada y espacio en memoria para creación del nuevo nodo.

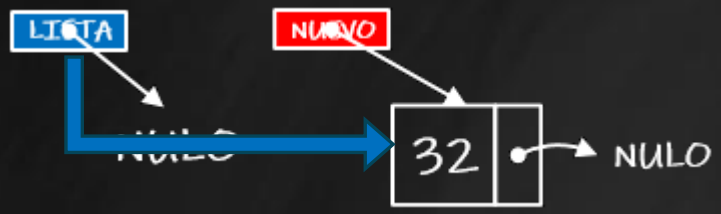
Resumiendo ambos casos

```
void agregar_inicio(pnodo &lista,pnodo nuevo)
{ nuevo->sig=lista;
  lista=nuevo;
}
```

Operación agregar al final

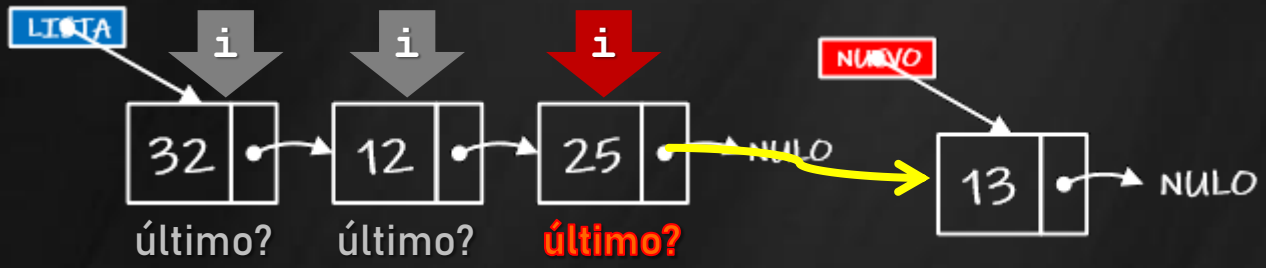
Permite agregar un nodo al final de la lista

Caso 1: Lista Vacía




```
lista=nuevo;
```

Caso 2: Lista con elementos




```
for(i=lista;i->sig!=NULL;i=i->sig);
i->sig=nuevo;
```

Implementación (5)

- Operación *agregar al final*

```
void agregar_final(pnodo &lista,pnodo nuevo)
{ pnodo i;
      if (lista==NULL)
         lista=nuevo;
      else
        { for(i=lista;i->sig!=NULL;i=i->sig);
          i->sig=nuevo;
        }
}
```

Lista vacía

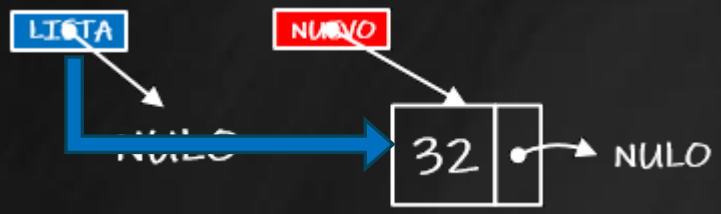
Lista con elementos

- Agregar un nodo al final
  - Propósito: agregar un nodo al final de la lista.
  - Entrada: una lista y un nuevo dato.
  - Salida: una lista con un nuevo nodo al final.
  - Restricciones: una lista inicializada y espacio en memoria para creación del nuevo nodo.

Operación agregar en orden

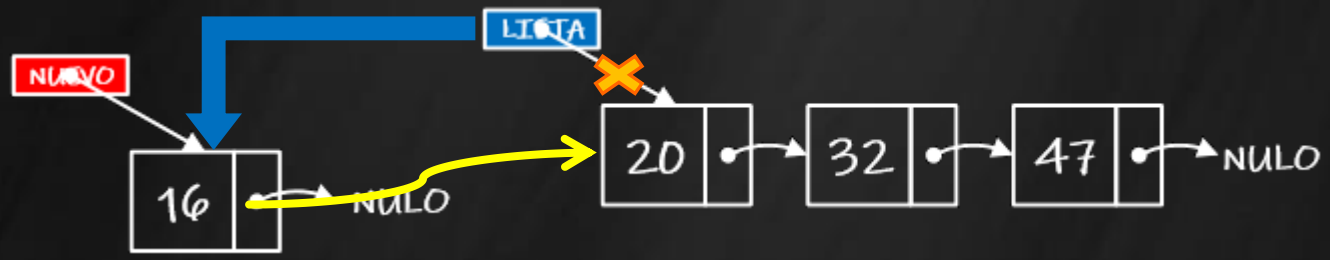
Permite agregar un nodo a la lista con un criterio de orden

Caso 1: Lista Vacía




```
lista=nuevo;
```

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



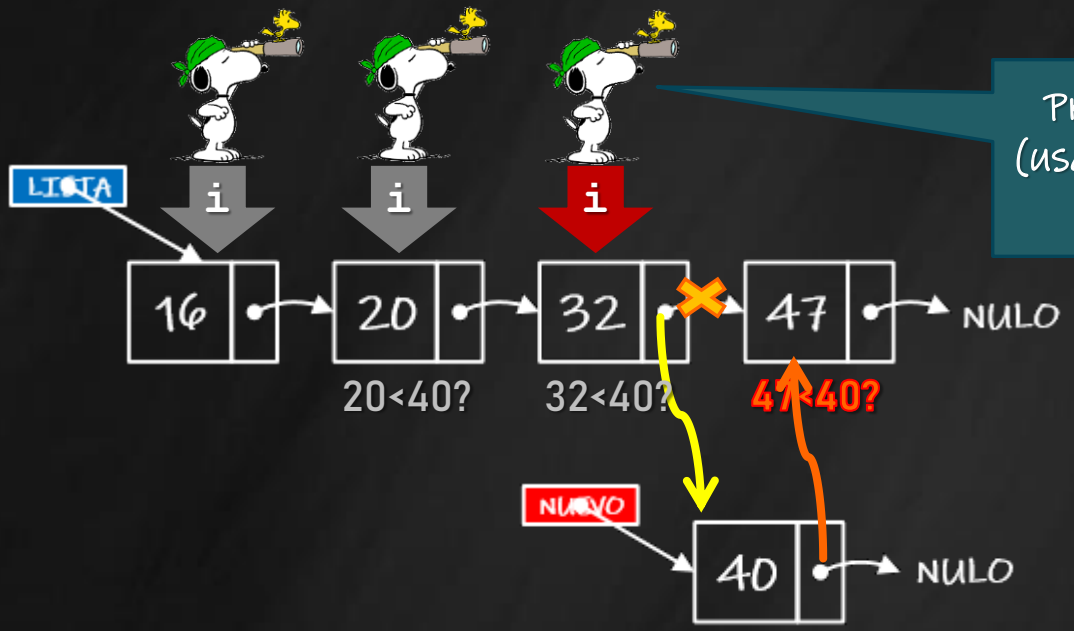
16<20?

```
nuevo->sig=lista;
lista=nuevo;
```

Operación agregar en orden

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

Caso 3: el nuevo valor debe ubicarse en el medio o final de la lista



Preguntamos de forma anticipada (usando el puntero sig) para conectar correctamente el nuevo nodo

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

Implementación (6)

- Operación *agregar en orden*

```
void agregarorden(pnodo &lista,pnodo nuevo)
{ pnodo i;
  if (lista==NULL)
   lista=nuevo;
  else
   {if (nuevo->dato < lista->dato)
     {nuevo->sig=lista;
      lista=nuevo;}
     else
     {for(i=lista;i->sig!=NULL && i->sig)->dato < nuevo->dato;i=i->sig);
        nuevo->sig=i->sig;
        i->sig=nuevo; }
   }
}
```

Lista vacía

Agregar al inicio (el nuevo valor es menor que el 1ro)

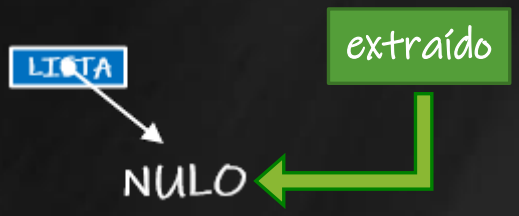
Agregar al medio o final

- Agregar un nodo en orden
  - Propósito: agregar, en orden, un nodo a la lista.
  - Entrada: una lista y un nuevo dato.
  - Salida: una lista, ordenada, con un nuevo nodo.
  - Restricciones: una lista inicializada y espacio en memoria para la creación del nuevo nodo.

Operación quitar inicio

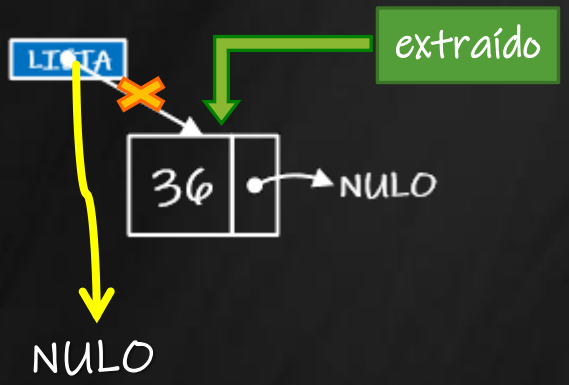
Permite extraer el primer nodo de la lista.

Caso 1: Lista Vacía




```
extraido=NULL;
```

Caso 2: Lista con un único elemento




```
extraido=lista;
lista=NULL;
```

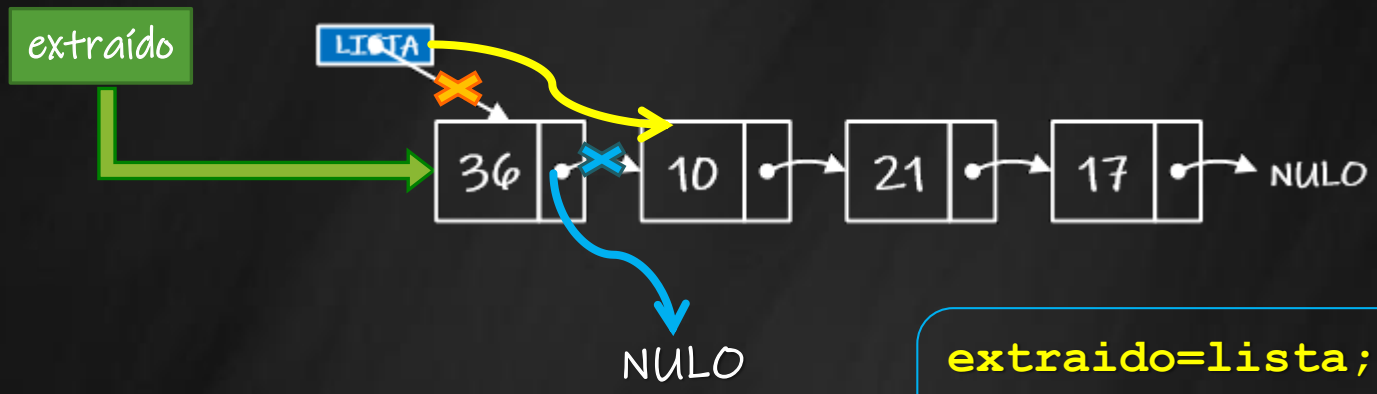
o

```
extraido=lista;
lista=lista->sig;
```

Operación quitar inicio

Caso 3: Lista con 2 o más elementos

Permite extraer el primer nodo de la lista.




```
extraido=lista;
lista=lista->sig;
extraido->sig=NULL;
```

¿Los casos 2 y 3 pueden unificarse?

Caso 2

```
extraido=lista;
lista=lista->sig;
```

Caso 3

```
extraido=lista;
lista=lista->sig;
extraido->sig=NULL;
```

Implementación (7)

- Operación *quitar del inicio*

```
pnodo quitar_inicio(pnodo &lista)
{ pnodo extraido;
  if (lista==NULL)
     extraido=NULL;
  else
   { extraido=lista;
     lista=lista->sig;
     extraido->sig=NULL;
   }
  return extraido;
}
```

Extracción de lista vacía

Extracción del primer nodo

- Quitar un nodo del inicio
  - Propósito: quitar el primer nodo de la lista.
  - Entrada: una lista.
  - Salida: una lista con un nodo menos (extraído del inicio) y la dirección del elemento extraído.
  - Restricciones: una lista inicializada y no vacía.

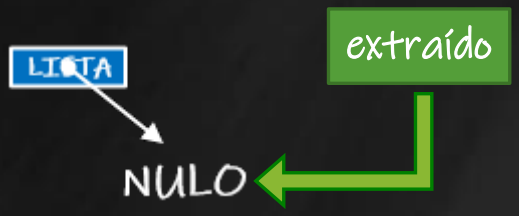
¿Cómo se usa *quitar_inicio*?

```
eliminado=quitar_inicio(milista);
 if (eliminado!=NULL)
   { cout << "Eliminado: " << eliminado->dato;
     delete(eliminado); }
 else
   cout << "NO PUEDE ELIMINAR, LISTA VACIA“;
```

Operación quitar final

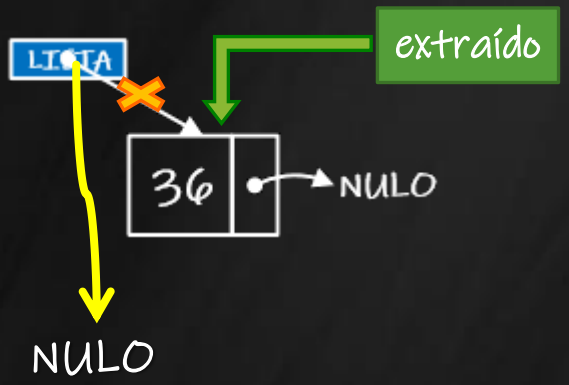
Permite extraer el último nodo de la lista.

Caso 1: Lista Vacía




```
extraido=NULL;
```

Caso 2: Lista con un único elemento




```
extraido=lista;
lista=NULL;
```

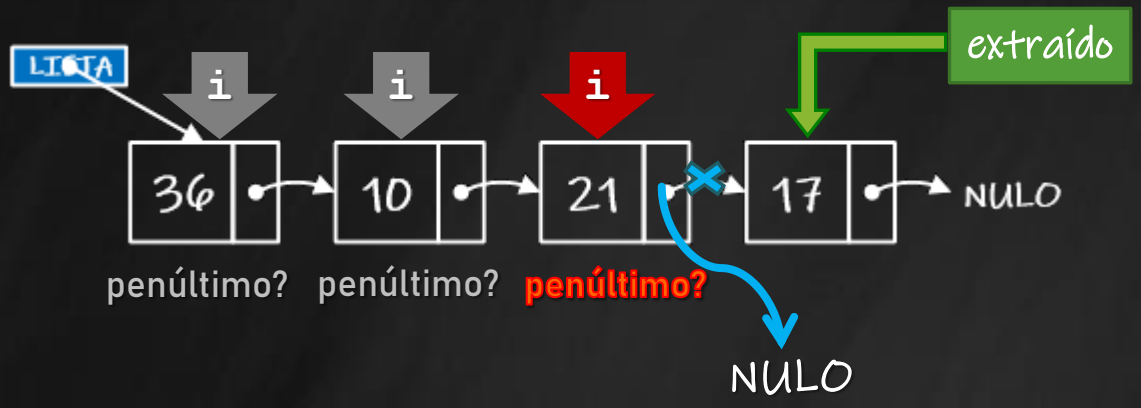
o

```
extraido=lista;
lista=lista->sig;
```

Operación quitar final

Caso 3: Lista con 2 o más elementos

Permite extraer el último nodo de la lista.




```
for( i=lista ;(i->sig)->sig!=NULL ; i=i->sig );
extraido=i->sig;
i->sig=NULL;
```

Implementación (8)

- Operación *quitar del final*

```
pnodo quitar_final(pnodo &lista)
{ pnodo extraido,i;
  if (lista==NULL)
     extraido=NULL;
  else
   {if (lista->sig==NULL)
      {extraido=lista;
        lista=NULL; }
    else
      { for(i=lista;(i->sig)->sig!=NULL;i=i->sig);
        extraido=i->sig;
        i->sig=NULL; }
   }
  return extraido;
}
```

Extracción de lista vacía

Extracción del único nodo

Extracción del último nodo

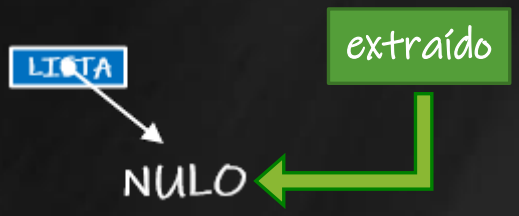
- Quitar un nodo del final
  - Propósito: quitar el último nodo de la lista.
  - Entrada: una lista.
  - Salida: una lista con un nodo menos (extraído del final) y la dirección del elemento extraído.
  - Restricciones: una lista inicializada y no vacía.

Operación quitar nodo específico

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

Caso 1: Lista Vacía

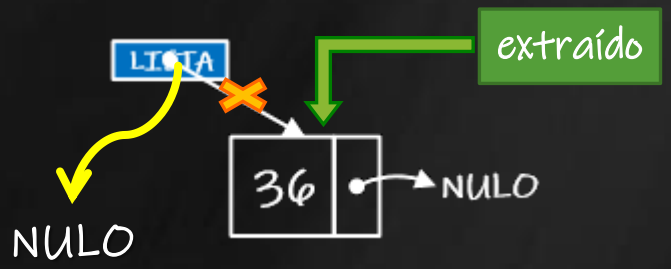
Extraer 36



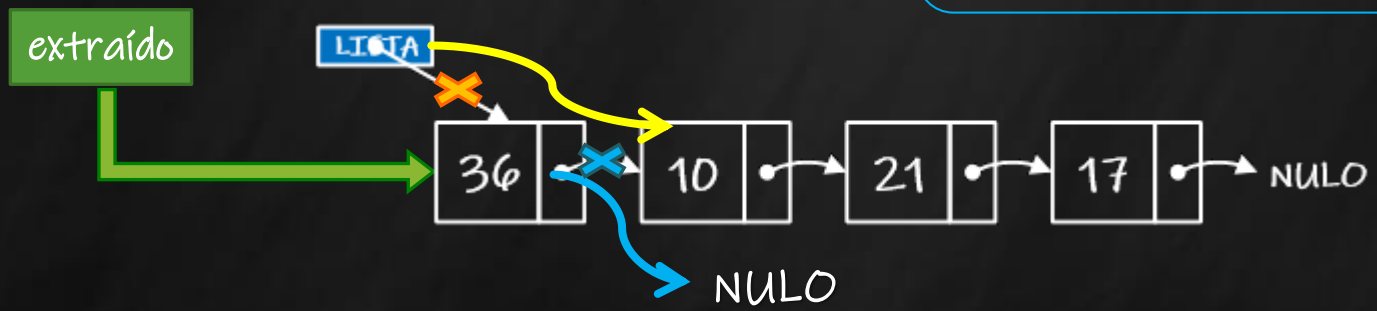

```
extraido=NULL;
```

Caso 2: Extracción del primer elemento (único/varios elementos)

Extraer 36



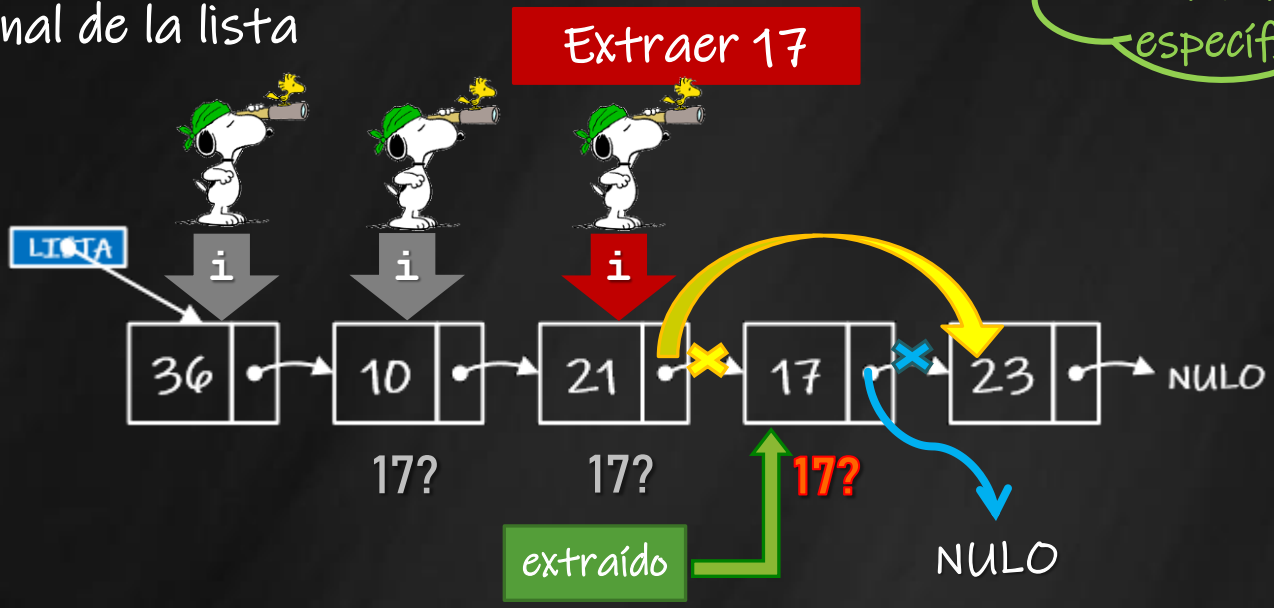

```
extraido=lista;
lista=lista->sig;
extraido->sig=NULL;
```



Operación quitar nodo específico

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

Caso 3: Extracción de un valor del medio o final de la lista




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

¿Qué ocurre si el valor a extraer no se encuentra en la lista?

Implementación (9)

- Quitar un nodo según un valor especificado
  - Propósito: quitar un nodo con valor específico.
  - Entrada: una lista y el valor a extraer.
  - Salida: una lista con un nodo menos (extraído el valor solicitado) y la dirección del nodo extraído.
  - Restricciones: una lista inicializada y no vacía.

- Operación *quitar un nodo según un valor solicitado*

```
pnodo quitar_nodo(pnodo &lista,int valor)
{ pnodo extraido,i;
  if (lista==NULL)
    extraido=NULL;
  else
    if (lista->dato==valor)
     {extraido=lista;
       lista=extraido->sig;
      extraido->sig=NULL; }
    else
     {for(i=lista;i->sig!=NULL && valor!=(i->sig)->dato;i=i->sig);
      if (i->sig!=NULL)
       {extraido=i->sig;
        i->sig=extraido->sig;
        extraido->sig=NULL; }
      else
        extraido=NULL; }
  return extraido;
}
```

Extracción de lista vacía

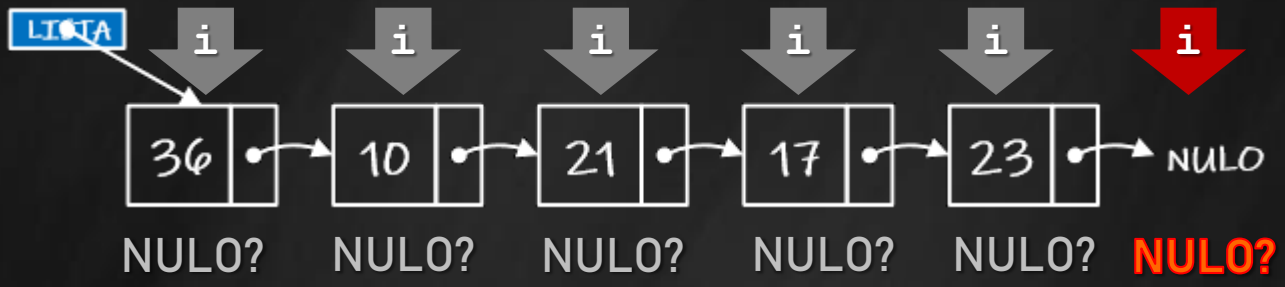
Extracción del primer nodo

Extracción del medio o final

Valor no encontrado

Operación mostrar lista

Permite mostrar nodo a nodo el contenido de una lista.




```
36   10   21   17   23
```

Implementación (10)

- Operación *mostrar datos de la lista*

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

- **Mostrar lista**
  - **Propósito: mostrar el contenido de la lista.**
  - **Entrada: una lista (puntero de inicio de la lista).**
  - **Salida: se muestran por pantalla los datos almacenados en los nodos.**
  - **Restricciones: una lista inicializada y no vacía.**

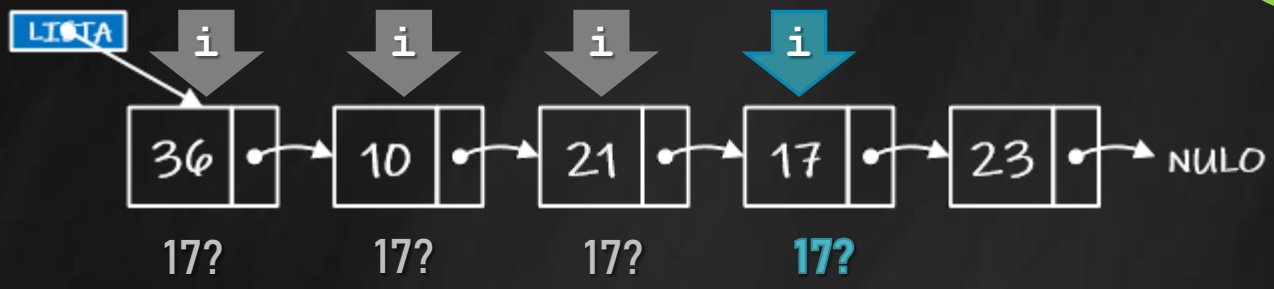
Se recorre la lista, nodo a nodo, hasta que el puntero *i* sea NULO

Operación buscar dato

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

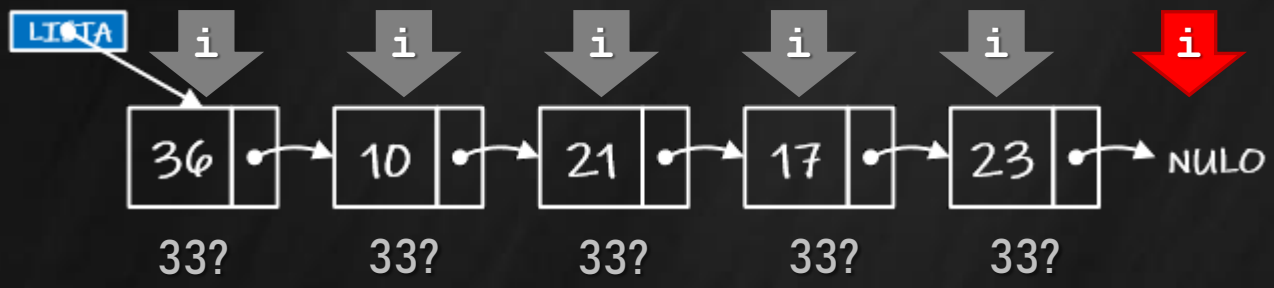
Caso Positivo: buscando un valor existente

Buscar 17



Caso Negativo: buscando un valor inexistente

Buscar 33



Implementación (11)

- Buscar un dato en la lista
  - Propósito: buscar un valor específico en la lista
  - Entrada: una lista y el valor a buscar.
  - Salida: valor true si el dato buscado se encuentra en la lista, caso contrario, false.
  - Restricciones: una lista inicializada y no vacía.

- Operación *buscar un dato en la lista*

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

Se recorre la lista, nodo a nodo, hasta que el puntero *i* sea NULO o se detecte el valor buscado.

Aplicaciones

- El TDA lista puede aplicarse para:
  - implementación de cualquier colección homogénea de elementos (por ejemplo, conjuntos).
  - implementación del TDA pila
  - implementación del TDA cola
- El TDA lista permite realizar una implementación dinámica que resulta útil cuando la variabilidad en la cantidad de elementos del problema es grande.
- El TDA lista permite implementar muchas de las estructuras utilizadas por los Sistemas Operativos.

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.