Listas

Introducción a la programación II

Estructuras de almacenamiento dinámicas

Surgen de la necesidad de utilizar espacio de almacenamiento que no se puede predecir a priori. Así, no es factible utilizar estructuras de datos estáticas debido a que puede sobrar espacio o puede no alcanzar.

Al emplear estructuras de almacenamiento dinámicas, las búsquedas, eliminaciones, e inserciones son más lentas y complejas.

Las estructuras dinámicas de datos más conocidas son: listas, arboles y grafos.

La lista enlazada o simple es una secuencia de nodos enlazados c/u con el siguiente mediante un puntero.

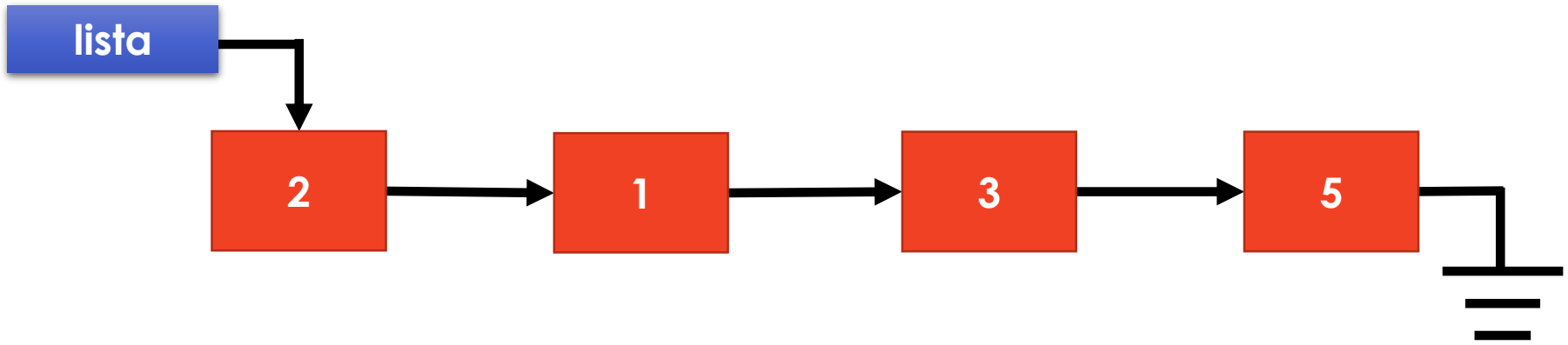
```
type
        punteronodo=^nodo;
        nodo=record
                variable1:tipo1;
                …
                variableN:tipoN;
                ste:punteronodo;
        end;
```

Lista simple

Ejemplo

```
type
        punterolista=^nodo;
        nodo=record
                dato:integer;
                ste:punterolista;
        end;
var     lista:punterolista;
```

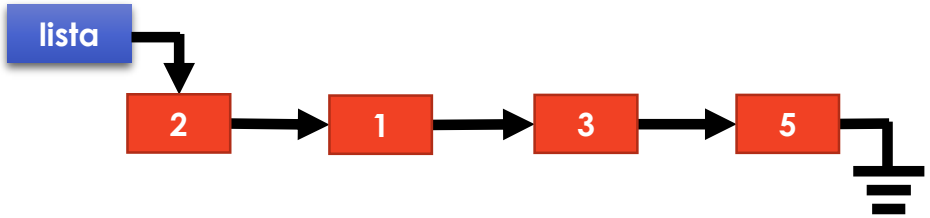
Característica: Mantiene puntero al primer nodo de la estructura y el ste del último nodo es NIL.



Lista simple

¿Cómo construir la lista simple?

```
type
        punterolista=^nodo;
        nodo=record
                dato:integer;
                ste:punterolista;
        end;
var
        lista:punterolista;
```

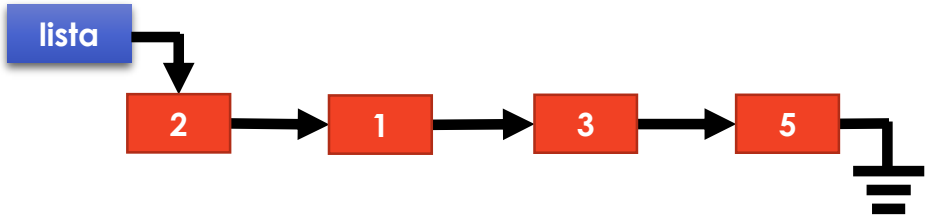


Representación de lista en memoria

Variable	Dirección de memoria	Contenido
Lista	200	250
Lista^.dato	250	2
Lista^.ste		261
Lista^.ste^.dato	261	1
Lista^. ste^.ste		265
Lista^.ste^.ste^.dato	265	3
Lista^.ste^.ste^.ste		268
Lista^.ste^.ste^.ste^.dato	268	5
Lista^.ste^.ste^.ste^.ste		NIL

Lista simple

```
type
        punterolista=^nodo;
        nodo=record
                dato:integer;
                ste:punterolista;
        end;
var
        lista,nodo:punterolista;
Begin
        new(nodo);
        nodo^.dato:=2;
        lista:=nodo;
        new(nodo);
        nodo^.dato:=1;
        lista^.ste:=nodo;
        new(nodo);
        nodo^.dato:=3;
        lista^.ste^.ste:=nodo;
        new(nodo);
        nodo^.dato:=5;
        nodo^.ste:=NIL;
        lista^.ste^.ste^.ste:=nodo;
        …
End.
```

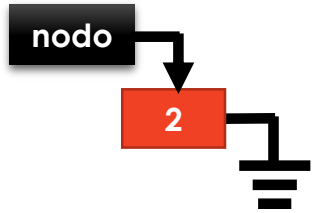


Siempre agrega un nodo al final

Observación: esta forma de agregar nuevos nodos a una lista es solo demostrativa. Para agregar nuevos nodo a una lista debemos utilizar métodos.

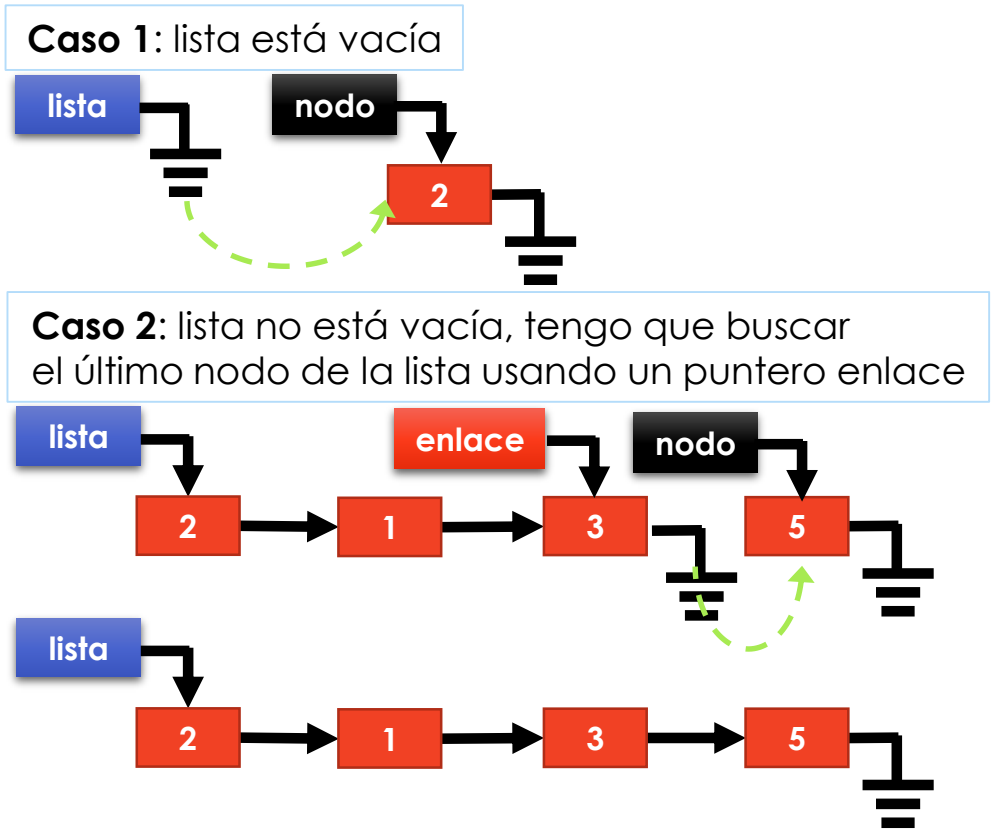
Crear un nuevo nodo

```
Procedure crearnodo(var nodo:punterolista;dato:integer);
//La creación del nuevo nodo se puede realizar mediante un procedimiento como en este
//ejemplo o una función (con otro nombre) que retorno un puntero del nodo creado
Begin
        new(nodo);
        nodo^.dato:=dato;
        nodo^.ste:=NIL;
End;
```



Lista simple

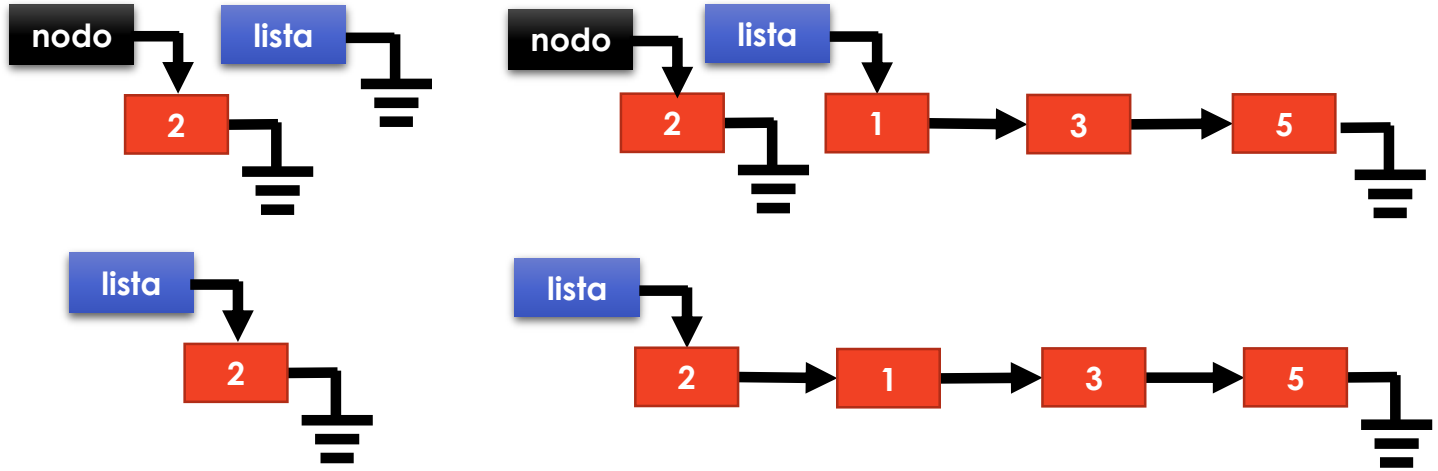
```
Procedure agregarnodoalfinal(var lista:punterolista;nodo:punterolista);
//el nuevo nodo ya esta creado previamente a llamar este método
//defino un punto enlace para que recorra la lista sin modificar y perder el puntero al primer nodo
var         enlace:punterolista;
Begin
            if lista=NIL then
                        lista:=nodo
            else
            begin
                        enlace:=lista;
                        while enlace^.ste<>NIL do
                                    enlace:=enlace^.ste;
                        enlace^.ste:=nodo;
            end;
End;
```



Lista simple

```
Procedure agregardatoalprincipio(var lista:punterolista;dato:integer);
var         nodo:punterolista;
Begin
            crearnodo(nodo,dato);
            nodo^.ste:=lista;
            lista:=nodo;
End;
```

Caso único: no importa si la lista está vacía o no, siempre se enlaza de la misma forma



Lista simple

```
function existedatolista(lista:punterolista;dato:integer):boolean;
//busco un dato, NUNCA tengo que crear memoria en un puntero para buscar un dato
Var     aux:punterolista;
Begin
        aux:=lista;
        while (aux<>NIL)and(aux^.dato<>dato) do
                aux:=aux^.ste;
        existenodolista:=(aux<>NIL);
End;
```

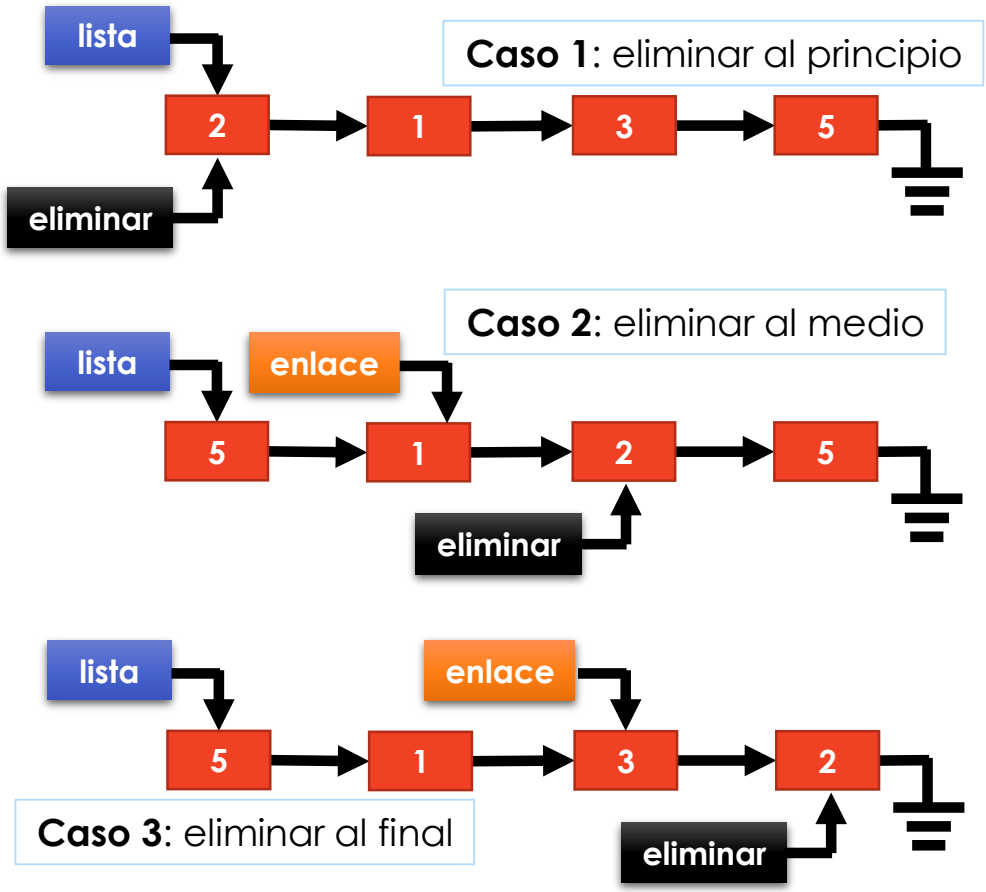
El resultado solo permite conocer si el dato existe

```
function obtenerpunteronodolista(lista:punterolista;dato:integer):punterolista;
//busco un puntero a partir de dato, NUNCA tengo que crear memoria en un puntero para buscar un dato
Var     aux:punterolista;
Begin
        aux:=lista;
        while (aux<>NIL)and(aux^.dato<>dato) do
                aux:=aux^.ste;
        obtenerpunteronodolista:=aux;
End;
```

El puntero permite tener acceso al contenido y además saber si existe

Lista simple

```
Procedure eliminarnododelista(var lista:punterolista;dato:integer);
//NUNCA crear memoria en un puntero donde esta dato para luego eliminarlo
Var             eliminar,enlace:punterolista;
Begin
If (lista<>NIL) then begin
     //Caso 1
     if (lista^.dato=dato) then begin
                    eliminar:=lista;
                    lista:=lista^.ste;
                    dispose(eliminar);
     end
     else begin
                     //Caso 2 y 3
                     //Siempre se consulta desde el nodo anterior
                    enlace:=lista;
                    while (enlace^.ste<>NIL)and
                                   (enlace^.ste^.dato<>dato) do
                                   enlace:=enlace^.ste;
                    if (enlace^.ste<>NIL) then
                    begin
                                   eliminar:=enlace^.ste;
                                   enlace^.ste:= enlace^.ste^.ste;
                                   dispose(eliminar);
                    end
     end;
end;
End;
```



Resolución de problemas con lista simple

Para poder resolver problemas con lista simple implica poder resolver previamente los siguientes métodos:

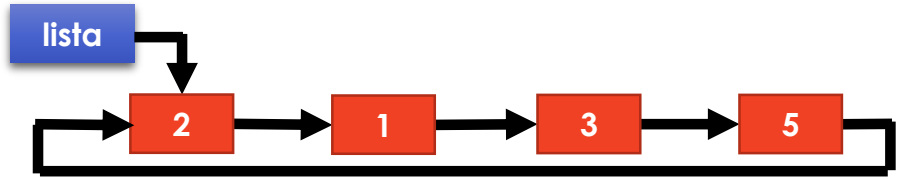
- Agregar nodo al principio
- Agregar nodo al final
- Eliminar nodo
- Obtener puntero a nodo
- Existe valor en lista
- Hacer cálculos sobre lista:
  - **Obtener suma de lista**
  - **Obtener promedio de lista**
- **Eliminar lista completa**

Hay que recorrer toda la lista dependiendo de su orden

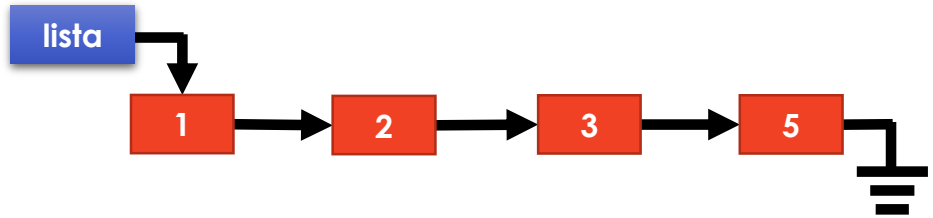
Casos especiales de lista simple

Hay casos espaciales de lista simple que también son utilizados en la materia:

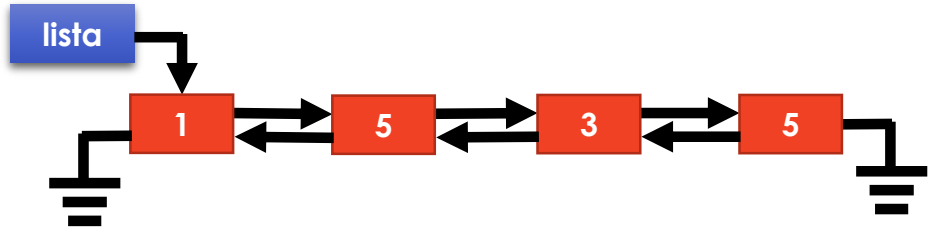
- **Lista simple circular**: el puntero siguiente del último nodo apunta al primero de la lista.



- **Lista simple ordenada**: los elementos mantienen un criterio de orden, que debe mantenerse mientras se realizan las distintas operaciones.



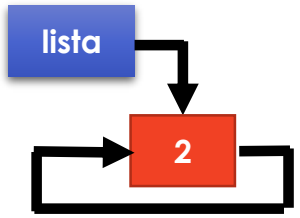
- **Lista doblemente vinculada**: los nodos mantienen un puntero al siguiente y al anterior.



Lista simple circular

Durante las búsquedas o recorridos no se puede basar en la existencia de NIL. Así, hay que considerar a veces casos que no están o difieren de la lista simple común.

Por ejemplo, para eliminar el primer nodo aparecen estos casos.



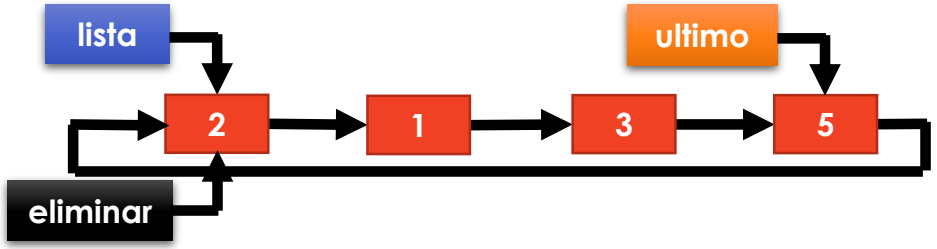
If (lista<>nil) and (lista^.dato=dato) and (lista^.ste=lista) then…

Directamente se elimina **lista** y se le asigna NIL (no requiere un puntero auxiliar).

else If (lista<>nil) and (lista^.dato=dato) and (lista^.ste<>lista) then…

Hay que buscar el puntero al **último** nodo.

```
…
enlace:=lista;
while (enlace^.ste<>lista) do
        enlace:=enlace^.ste;
…
```

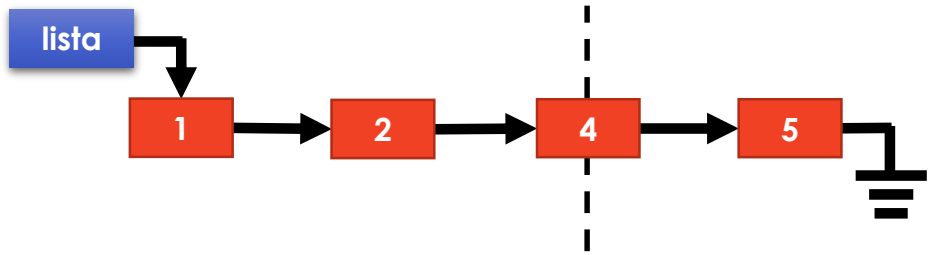


Luego se actualiza el puntero lista, se actualiza el puntero ultimo, y finalmente utilizar un puntero auxiliar para **eliminar** el nodo.

Lista simple ordenada

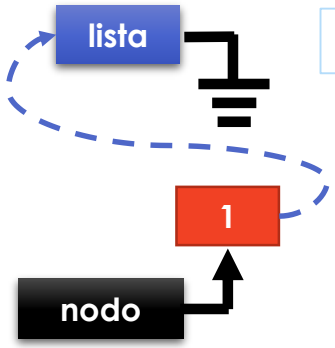
Para este tipo de lista también hay casos espaciales que no están en la lista simple común.

No realizar una búsqueda innecesaria si por el criterio de orden el elemento no existe. Ejemplo, buscar o eliminar el nodo con contenido 3.



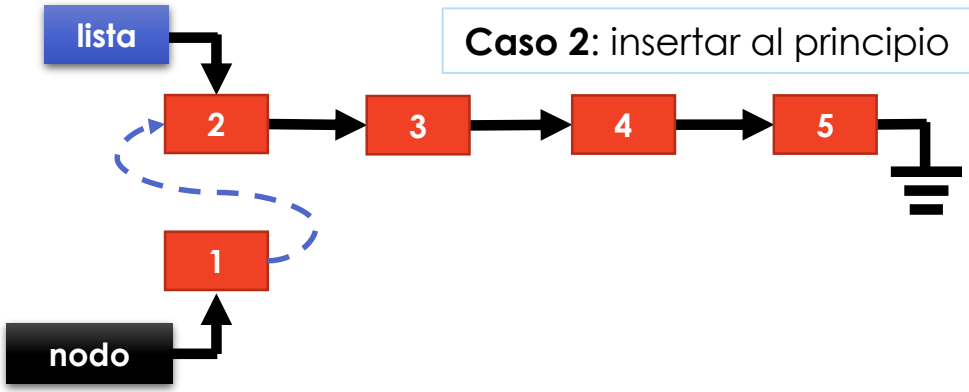
Cuando se agrega un elemento se debe insertar ordenado. Agregarlo al principio o al final y luego reordenar está **MAL**.

Insertar nodo en lista simple ordenada



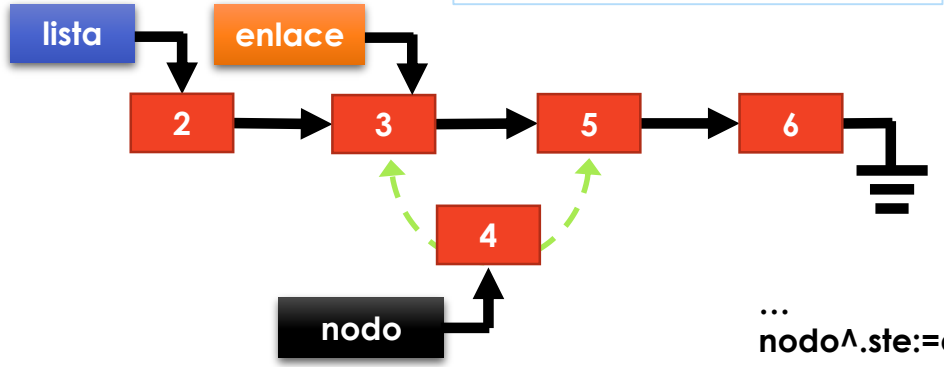
Caso 1: lista vacia

```
...
nodo^.ste:=lista;
lista:=nodo;
...
```

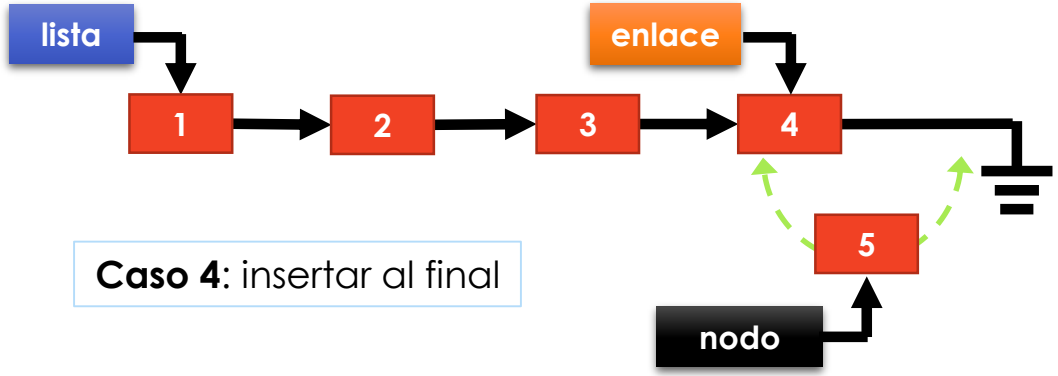


Caso 2: insertar al principio

Caso 3: insertar al medio



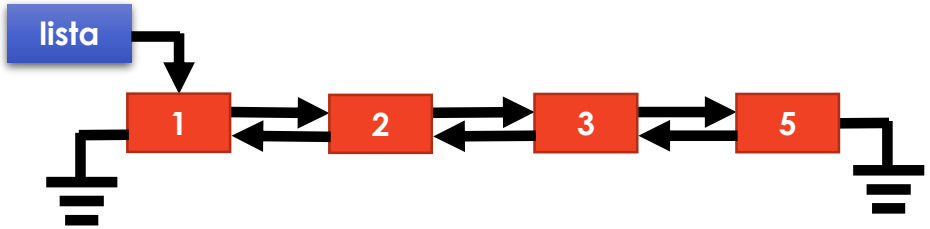

```
...
nodo^.ste:=enlace^.ste;
enlace^.ste:=nodo;
...
```



Caso 4: insertar al final

Lista doblemente vinculada

Este tipo de lista mantiene por cada nodo un puntero al siguiente y al anterior nodo.




```
type
        punterolista=^nodo;
        nodo=record
                dato:integer;
                ste,ant:punterolista;
        end;
```

Observaciones:

- Las operaciones de recorrido para eliminar o insertar no requieren ubicar el lugar desde el nodo anterior.
- Se requieren de más pasos ante una modificación para mantener la propiedad de doblemente vinculada.
- Según el problema puede indicar la presencia de un puntero al último nodo.

Tipos de listas descriptas hasta ahora

Para cada tipo de lista mencionada se deberá tener en cuenta los siguientes métodos (procedimientos y funciones).

	Búsqueda, recorrido, etc.	Agregar nodo	Insertar nodo	Eliminar nodo	Eliminar lista	Ordenar lista
Lista simple	X	X		X	X	X
Lista simple ordenada	X		X	X	X	
Lista circular	X	X		X	X	X
Lista circular ordenada	X		X	X	X	
Lista doblemente vinculada	X	X		X	X	X
Listas doblemente vinculada ordenada	X		X	X	X	