

LISTAS DOBLES

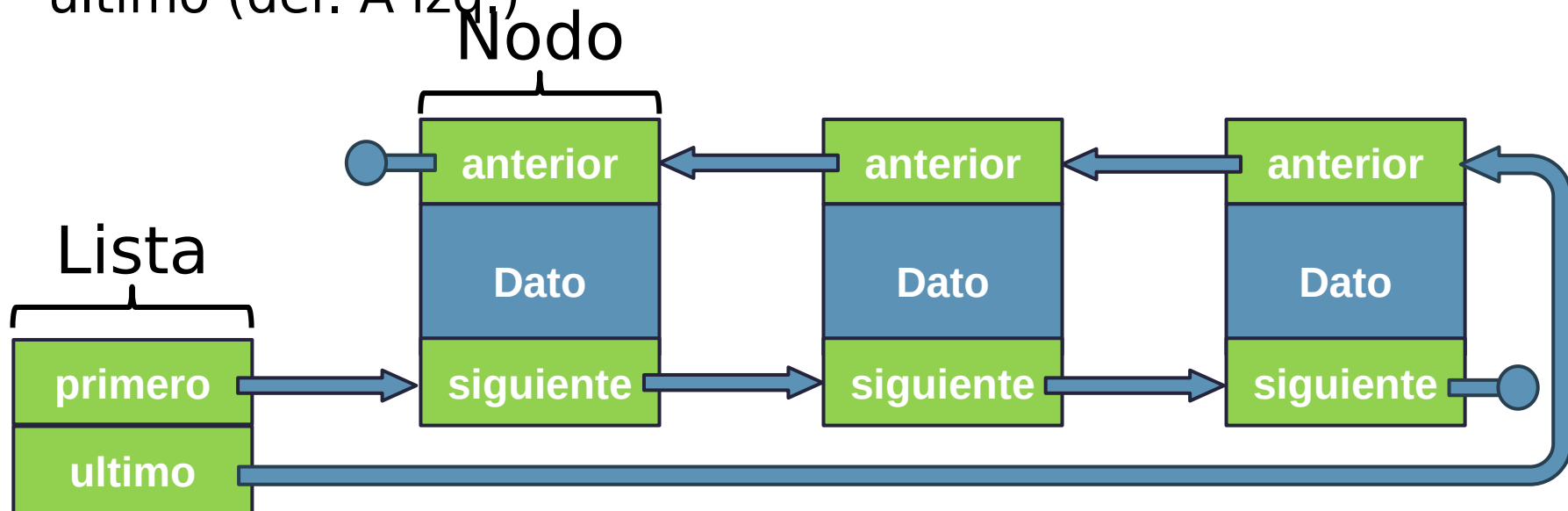
Explicación Práctica

Programación I - 2024

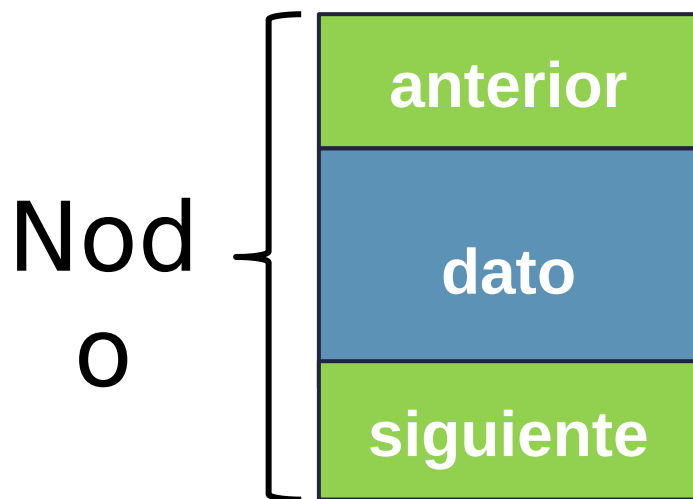
Facultad de Informática y Facultad de

Listas Dobles - Estructura

- Cada nodo tiene dos enlaces/punteros: uno al nodo anterior y otro al nodo siguiente
- Esto permite recorrer la lista en ambas direcciones (hacia adelante y hacia atrás)
- los dos enlaces siguen el mismo criterio de orden
- La lista se compone de 2 enlaces: primero (izq. a der.) y ultimo (der. A izq.)

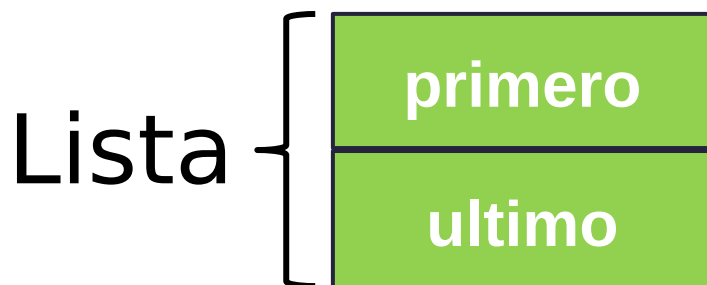


Listas Dobles – Definición de Estructura



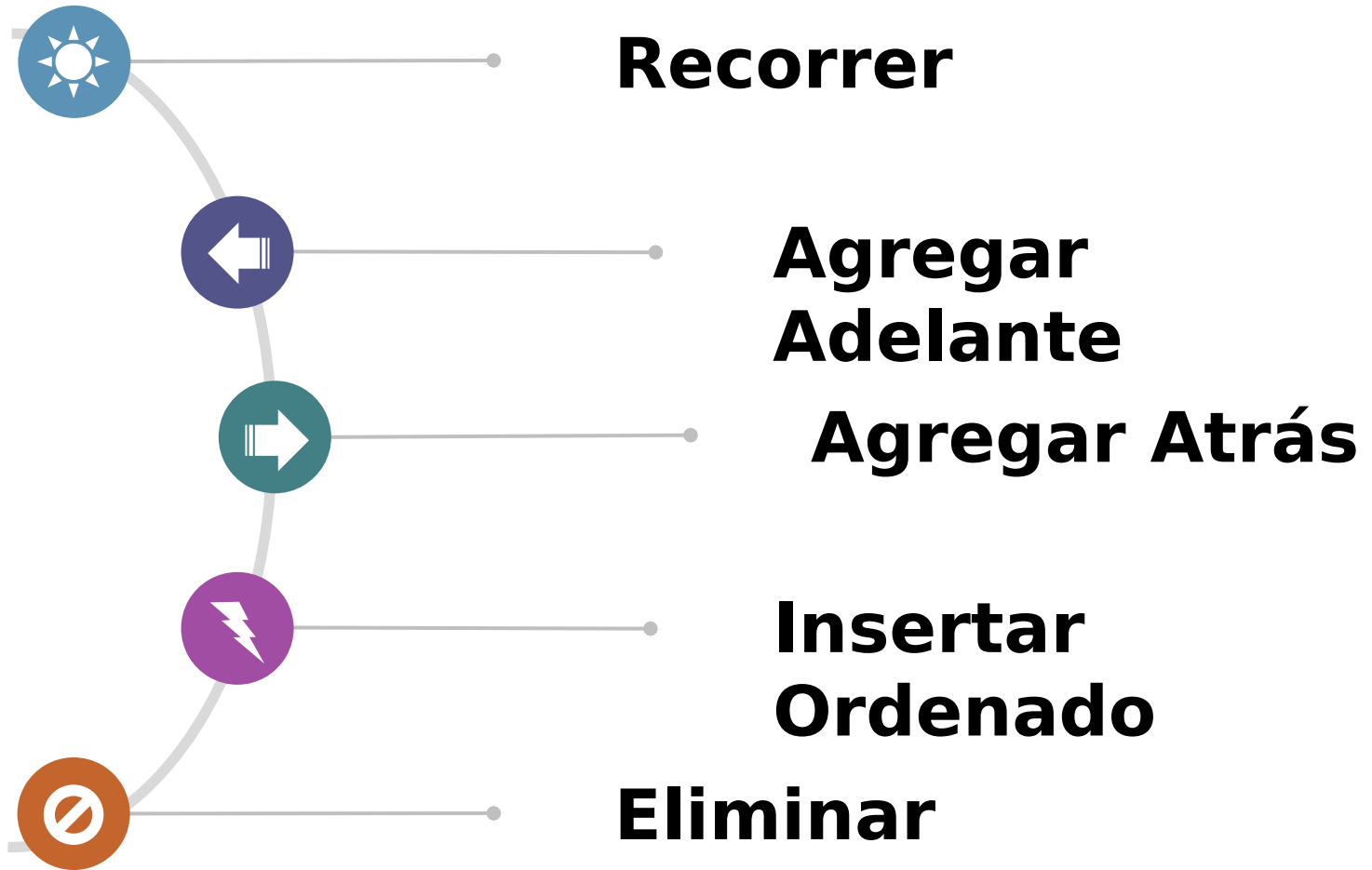
PtrNodo = ^nodo;

Nodo = **record**
 anterior: PtrNodo;
 dato: (*tipo de dato*);
 siguiente: PtrNodo;
end;



ListaDoble = **record**
 primero: PtrNodo;
 ultimo: PtrNodo;
end;

Listas Dobles - Operaciones



Listas Dobles – Estructura de Datos

Estructura de datos para lista doble

type

PtrNodo = ^nodo;

TipoDato = Integer;

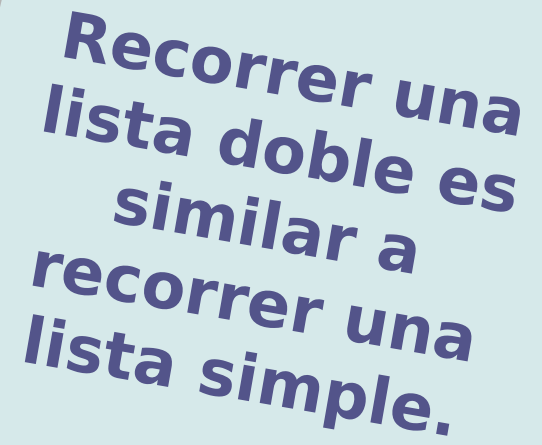
Nodo = **record**
 ant: PtrNodo;
 dato: TipoDato;
 sig: PtrNodo;
end;

ListaDoble = **record**
 primero: PtrNodo;
 ultimo: PtrNodo;
end;

Listas Dobles – Recorrido: imprimir

```
procedure ImprimirIzqDer(lista: ListaDoble);  
var actual: PtrNodo;  
begin  
  actual := lista.primeros;  
  while actual <> nil do begin  
    Write(actual^.dato, ' ');  
    actual := actual^.sig;  
  end;  
  WriteLn(); // Salto de línea  
end;
```

```
procedure ImprimirDerIzq(lista: ListaDoble);  
var actual: PtrNodo;  
begin  
  actual := lista.ultimo;  
  while actual <> nil do begin  
    Write(actual^.dato, ' ');  
    actual := actual^.ant;  
  end;  
  WriteLn; // Salto de línea  
end;
```

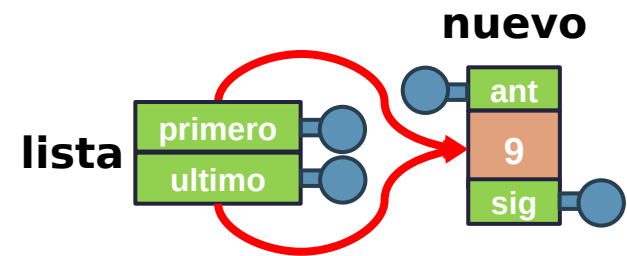


Recorrer una
lista doble es
similar a
recorrer una
lista simple.

Listas Dobles – Agregar Adelante

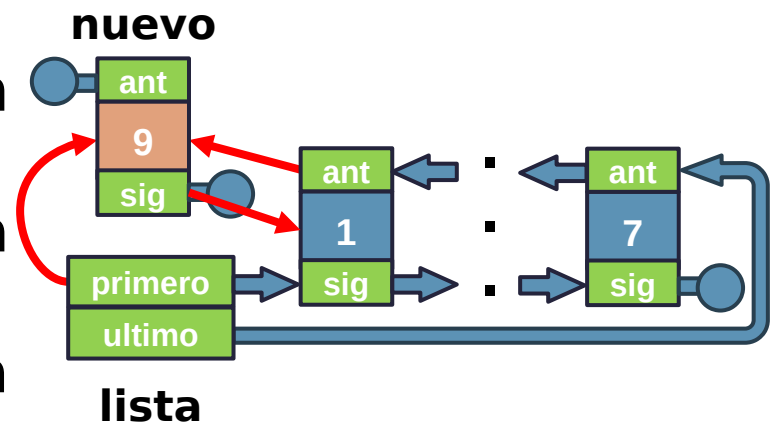
Lista vacía:

- ***lista.primer*** y ***lista.ultimo*** deben apuntar al nuevo
- ***nuevo^.ant*** y ***nuevo^.sig*** del nuevo elemento deben apuntar a Nil



Lista no vacía:

- ***nuevo*** debe quedar primero en ***lista***
- ***lista.primer*** debe apuntar a ***nuevo***
- no hay nodos anteriores a ***nuevo***: ***nuevo^.ant*** debe apuntar a Nil
- ***nuevo^.sig*** debe apuntar al



Listas Dobles – Agregar Adelante

```
procedure AgregarAdelante(var lista: ListaDoble; valor:  
TipoDato);
```

```
var nuevo: PtrNodo;
```

```
begin
```

```
  New(nuevo);
```

```
  nuevo^.dato := valor;
```

```
  nuevo^.ant := nil;
```

```
  nuevo^.sig := nil;
```

```
if lista.primeros = nil then begin
```

```
  // caso de lista vacía
```

```
  lista.primeros := nuevo;
```

```
  lista.ultimos := nuevo;
```

```
end
```

```
else begin
```

```
  // caso de lista NO vacía
```

```
  nuevo^.sig := lista.primeros;
```

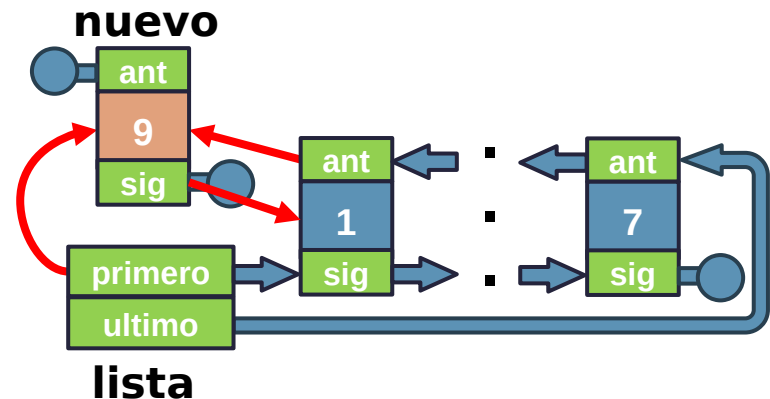
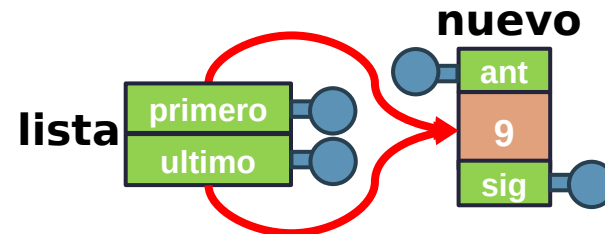
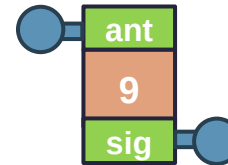
```
  lista.primeros^.ant := nuevo;
```

```
  lista.primeros := nuevo;
```

```
end;
```

```
end;
```

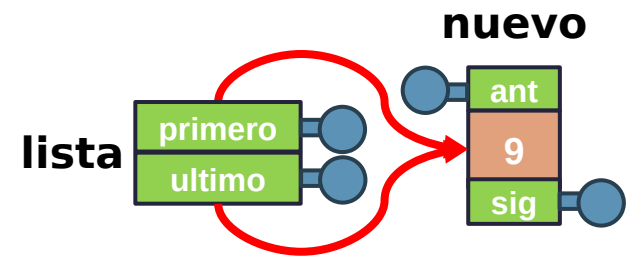
nuevo



Listas Dobles – Agregar al Final

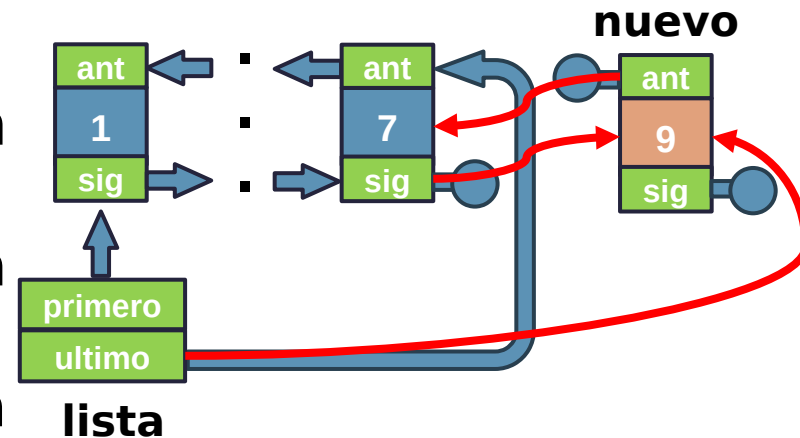
Lista vacía:

- ***lista.primer*** y ***lista.ultimo*** deben apuntar al nuevo
- ***nuevo^.ant*** y ***nuevo^.sig*** del nuevo elemento deben apuntar a Nil



Lista no vacía:

- ***nuevo*** debe quedar último en ***lista***
- ***lista.ultimo*** debe apuntar a ***nuevo***
- no hay nodos siguientes a ***nuevo***: ***nuevo^.ant*** debe apuntar al viejo último
- ***nuevo^.sig*** debe apuntar a Nil



Listas Dobles – Agregar al Final

```
procedure AgregarAlFinal(var Lista: ListaDoble; valor:  
TipoDato);
```

```
var nuevo: PtrNodo;
```

```
begin
```

```
  New(nuevo);
```

```
  nuevo^.dato := valor;
```

```
  nuevo^.sig := nil;
```

```
  nuevo^.ant := nil;
```

```
if Lista.primeros = nil then begin
```

```
  // caso de lista vacía
```

```
  Lista.primeros := nuevo;
```

```
  Lista.ultimos := nuevo;
```

```
end
```

```
else begin
```

```
  // caso de lista NO vacía
```

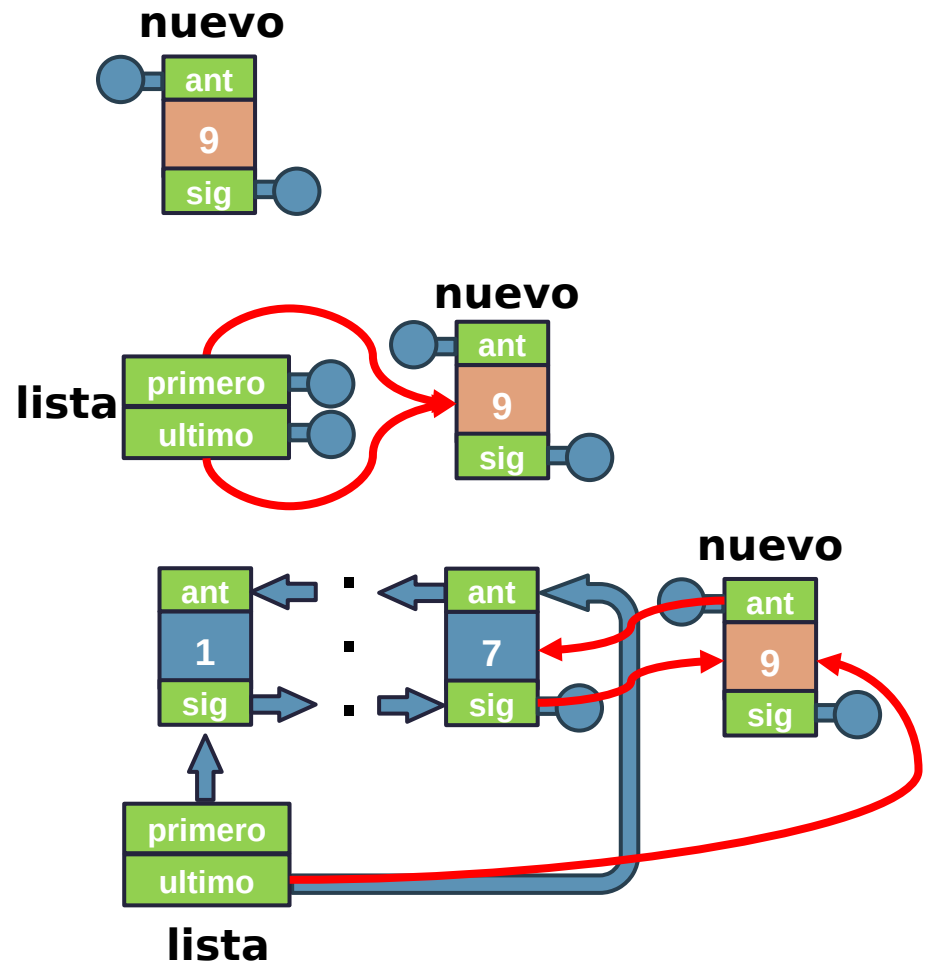
```
  Lista.ultimos^.sig := nuevo;
```

```
  nuevo^.ant := Lista.ultimos;
```

```
  Lista.ultimos := nuevo;
```

```
end;
```

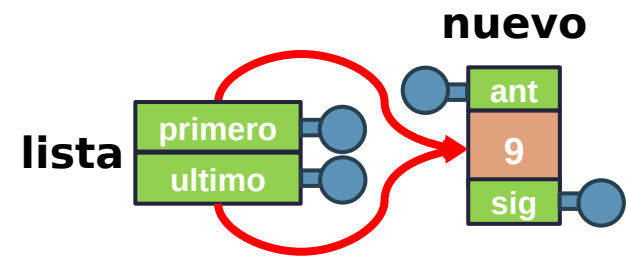
```
end;
```



Listas Dobles – Insertar Ordenado

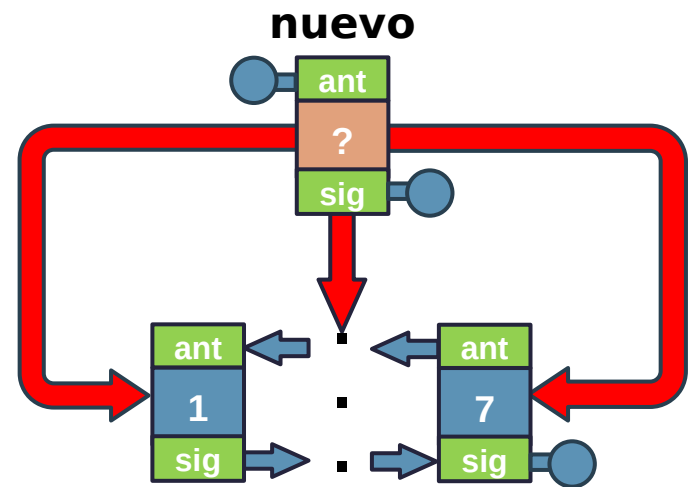
Lista vacía:

- ***lista.primer*** y ***lista.ultimo*** deben apuntar al nuevo
- ***nuevo^.ant*** y ***nuevo^.sig*** del nuevo elemento deben apuntar a Nil



Lista no vacía, se busca la posición y hay 3 casos posibles:

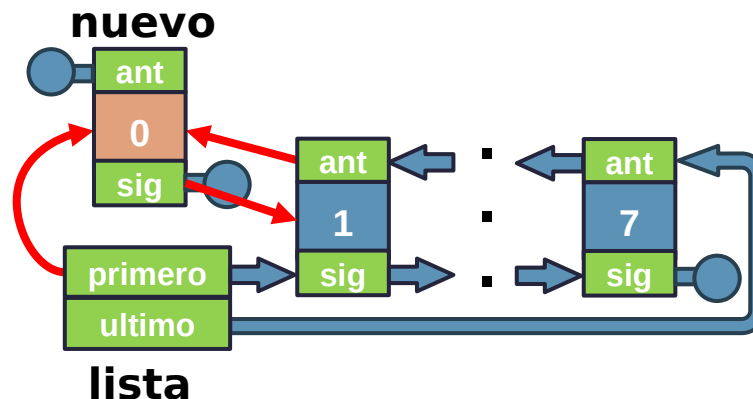
- ***Insertar al principio***
- ***Insertar en “medio”***
- ***Insertar al final***



Listas Dobles – Insertar Ordenado



Insertar al principio:

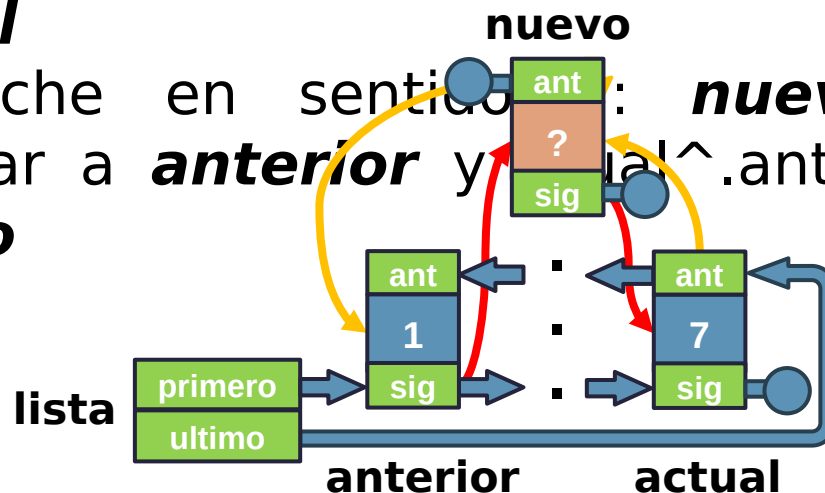
- **nuevo** debe quedar primero en **lista**
- **lista.primer** debe apuntar a **nuevo**
- no hay nodos anteriores a **nuevo**: **nuevo^.ant** debe apuntar a Nil
- **nuevo^.sig** debe apuntar al viejo primero
- Como la lista no está vacía, al insertar **nuevo** como primero, no hay que actualizar **lista.ultimo**



Listas Dobles – Insertar Ordenado

Insertar en “medio”:

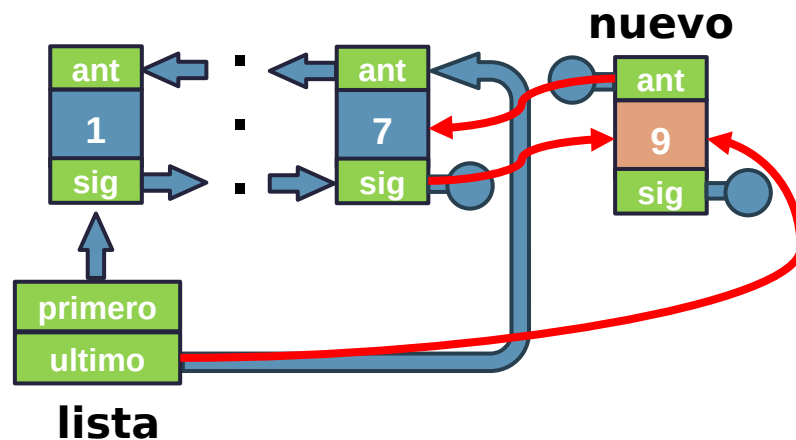
- El **nuevo** nodo, al quedar en el “medio” de la lista NO requiere la actualización de **lista.primer**o y **lista.ultimo**
- Hay un nodo **anterior** a **nuevo** y otro siguiente (**actual**):
 - Enganche en sentido  : **anterior^.sig** debe apuntar a **nuevo** y **nuevo^.sig** debe apuntar **actual**
 - Enganche en sentido  : **nuevo^.ant** debe apuntar a **anterior** y **actual^.ant** debe apuntar **nuevo**



Listas Dobles – Insertar Ordenado

Insertar al final:

- **nuevo** debe quedar ultimo en **lista**
- **lista.ultimo** debe apuntar a **nuevo**
- no hay nodos siguientes a **nuevo**: **nuevo^.sig** debe apuntar a Nil
- **nuevo^.ant** debe apuntar al viejo último
- Como la lista no está vacía, al insertar **nuevo** como último, no hay que actualizar **lista.primer**



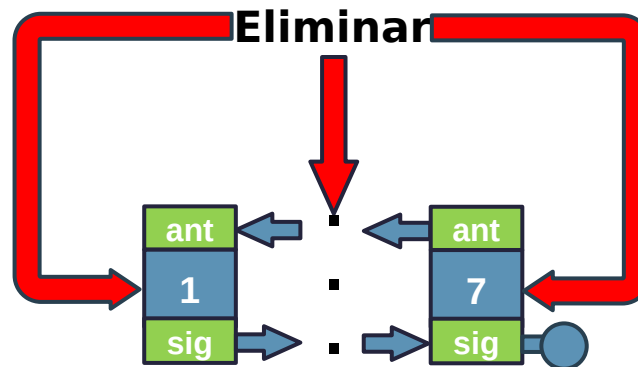
Listas Dobles – Insertar Ordenado

```
procedure InsertarOrdenado(var Lista: ListaDoble; valor: TipoDato);  
var nuevo, actual, anterior: PtrNodo;  
begin  
    New(nuevo); // Crear un nuevo nodo  
    nuevo^.dato := valor; nuevo^.ant := nil; nuevo^.sig := nil;  
    if Lista.primeros = nil then begin // caso de una lista vacía  
        Lista.primeros := nuevo;  
        Lista.ultimo := nuevo;  
    end  
    else begin // buscar posición para insertar  
        actual := Lista.primeros;  
        anterior := nil;  
        while (actual <> nil) and (actual^.dato < valor) do begin  
            anterior := actual;  
            actual := actual^.sig;  
        end;  
        if anterior = Nil then begin // Insertar al principio  
            nuevo^.sig := Lista.primeros;  
            Lista.primeros^.ant := nuevo;  
            Lista.primeros := nuevo;  
        end  
        else begin // Insertar en medio o al final  
            nuevo^.ant := anterior; nuevo^.sig := actual; anterior^.sig :=  
nuevo;  
        if actual <> nil then // diferencia de actualización si es medio o  
final  
            actual^.ant := nuevo  
        else  
            Lista.ultimo := nuevo;  
        end;  
    end;  
end;
```

Listas Dobles – Eliminar

Eliminar:

- Primero debe buscarse el elemento a eliminar
- Luego hay 3 casos posibles:
 - *Eliminar el nodo del principio*
 - *Eliminar un nodo intermedio*
 - *Eliminar el nodo final*



Listas Dobles – Eliminar

```
procedure EliminarPorValor(var Lista: ListaDoble; valor: TipoDato);  
Var actual: PtrNodo;  
begin  
    // Buscar el nodo con el valor dado  
    actual := Lista.primeros;  
    while (actual <> nil) and (actual^.dato <> valor) do  
        actual := actual^.sig;  
  
    // si encontró elemento => actualiza enganches en nodos y lista  
    if actual <> nil then begin  
        if actual^.ant <> nil then // existe nodo anterior => enganche  
            actual^.ant^.sig := actual^.sig;  
        if actual^.sig <> nil then  
            actual^.sig^.ant := actual^.ant;  
        if lista.primeros = actual then  
            lista.primeros := actual^.sig;  
        if lista.ultimo = actual then  
            lista.ultimo := actual^.ant;  
        Dispose(actual);  
    end;  
  
end;
```

Listas Dobles vs. Listas Simples

Ventajas

- Permiten recorrer la lista en ambas direcciones de forma eficiente
- Facilitan la inserción y eliminación de nodos, ya que cada nodo está enlazado con sus vecinos
- Útiles en situaciones donde se requiere acceso bidireccional

Desventajas

- Ocupan más memoria por nodo que las listas simplemente enlazadas (necesitan almacenar dos enlaces en lugar de uno)
- Las operaciones de inserción y eliminación son más complejas, ya que se deben actualizar los enlaces en ambas direcciones