



# Conceptos de Algoritmos Datos y Programas

# CADP – Temas de la clase de hoy

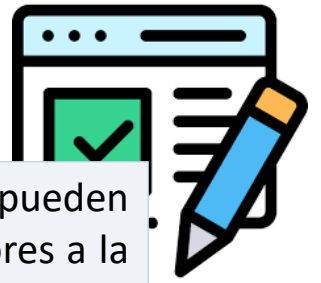


- Tipo de datos lista

- Características

- Operaciones

# CADP – Temas de la clase de hoy



**COMPUESTO:** pueden tomar varios valores a la vez que guardan alguna relación lógica entre ellos, bajo un único nombre.

## TIPO DE DATO

### SIMPLE

### COMPUESTO

#### DEFINIDO POR EL LENGUAJE

Integer  
Real  
Char  
Boolean

#### DEFINIDO POR EL PROGRAMADOR

Subrango  
Puntero

#### DEFINIDO POR EL LENGUAJE

String

#### DEFINIDO POR EL PROGRAMADOR

Registros  
Arreglos  
Lista

# CADP – Tipo de Dato

LISTA



## OPERACIONES

Creación de una lista.

Agregar nodos al comienzo de la lista.

Recorrido de una lista.

Agregar nodos al final de la lista.

Insertar nodos en una lista ordenada

Eliminar nodos de una lista



Trabajaremos con una  
lista de enteros



### AGREGAR ADELANTE EN UNA LISTA

Implica generar un nuevo nodo y agregarlo como primer elemento de la lista.



`pri = nil`



48

nil

nuevo

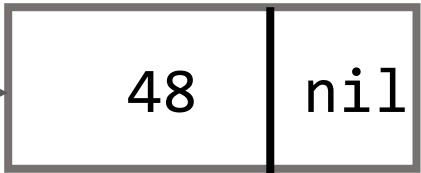
pri

167

nil



`pri <> nil`

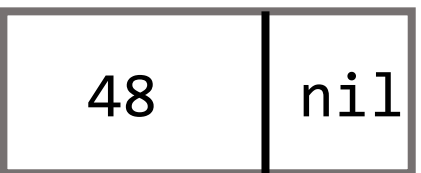


nuevo

pri

32

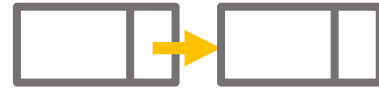
nil



nuevo

pri

*Cómo lo escribo?*



## AGREGAR ADELANTE EN UNA LISTA

Implica generar un nuevo nodo y agregarlo como primer elemento de la lista.

Reservo espacio en memoria **nuevo elemento**.

**si** (es el primer elemento a agregar)

asigno al puntero inicial la dirección del **nuevo elemento**.

**sino**

indico que el siguiente de **nuevo elemento** es el puntero inicial.

actualizo el puntero inicial de la lista con la dirección del **nuevo elemento**



## AGREGAR ADELANTE EN UNA LISTA

Implica generar un nuevo nodo y agregarlo como primer elemento de la lista.

```
Program uno;  
  
Type listaE= ^datosEnteros;  
  
    datosEnteros= record  
        elem:integer;  
        sig:listaE;  
    end;  
  
Var  
    pri: listaE;  
    num:integer;  
Begin  
    crear(pri);  
    read (num);  
    agregarAdelante (pri,num);  
End.
```



```
procedure agregarAdelante (var pI:listaE; num:integer);
Var
  nuevo:listaE;  creo espacio para el nuevo elemento
Begin
  new (nuevo); nuevo^.elem:= num; nuevo^.sig:=nil;

  if (pI = nil) then pI:= nuevo
  else begin
    nuevo^.sig:= pI;
    pI:=nuevo;
  end;
End;
```

*Evalúo el caso y reasigno los punteros*

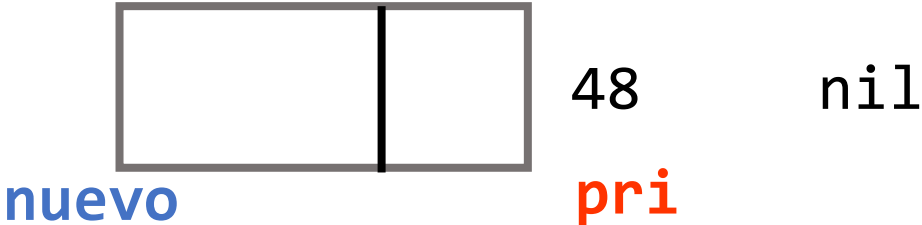




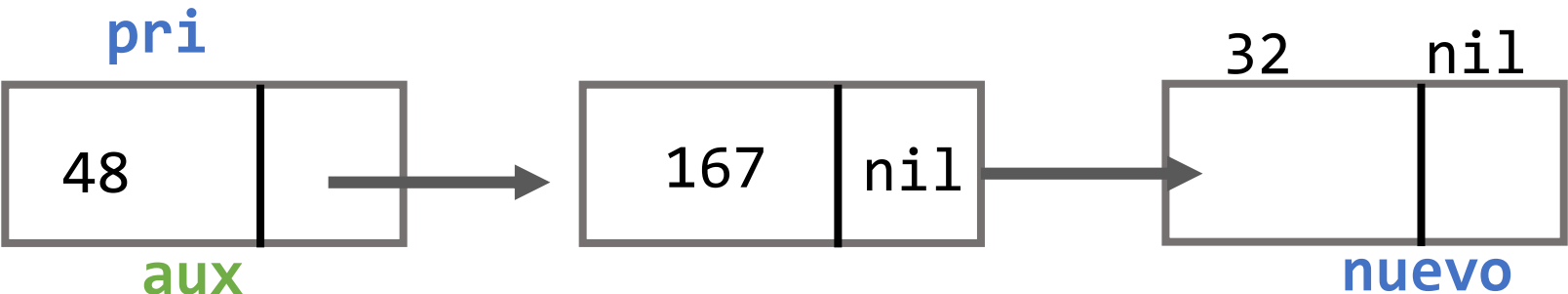
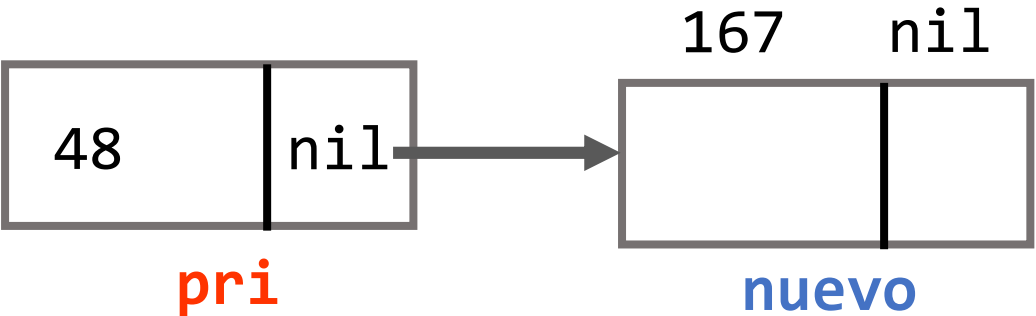
### AGREGAR AL FINAL EN UNA LISTA

Implica generar un nuevo nodo y agregarlo como último elemento de la lista.

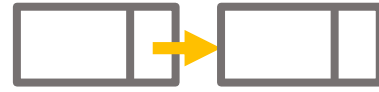
➤ `pri = nil`



➤ `pri <> nil`



*Cómo lo escribo?*



## AGREGAR AL FINAL EN UNA LISTA

Implica generar un nuevo nodo y agregarlo como último elemento de la lista.

Reservo espacio en memoria **nuevo elemento**.

**si** (es el primer elemento a agregar)

asigno al puntero inicial la dirección del **nuevo elemento**.

**sino**

inicializo un puntero auxiliar **aux**

mientras (no llegue al último elemento)

avanzo en la lista.

actualizo como siguiente del último nodo al **nuevo elemento**



## AGREGAR AL FINAL EN UNA LISTA

Implica generar un nuevo nodo y agregarlo como último elemento de la lista.

```
Program uno;
```

```
Type listaE= ^datosEnteros;
```

```
    datosEnteros= record
                        elem:integer;
                        sig:listaE;
                    end;
```

```
Var
```

```
    pri: listaE;
    num:integer;
```

```
Begin
```

```
    crear(pri);
    read (num);
    agregarAlFinal (pri, num);
```

```
End.
```

# CADP – Tipo de Dato - LISTA

AGREGAR AL FINAL



```
procedure agregarAlFinal (var pI:listaE; num:integer);
```

```
Var
```

```
    nuevo,aux:listaE;
```

```
Begin
```

```
    new (nuevo); nuevo^.elem:= num; nuevo^.sig:=nil;
```

```
    if (pI = nil) then pI:= nuevo <----- Evaluó si la lista está vacía
    else begin
```

```
        aux:= pI;
```

```
        while (aux ^.sig <> nil) do
```

```
            aux:= aux^.sig;
```

} Recorro y quedo  
parado en el último  
elemento

```
        aux^.sig:=nuevo; <----- Le indico al último que ahora
    end;                    su siguiente es nuevo
```

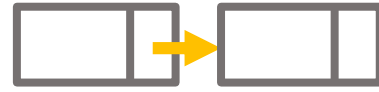
```
End;
```

Si agrego al final por qué  
paso por referencia el  
puntero inicial?

Por qué en la  
condición del while  
se pregunta por el  
aux^.sig?

# CADP – Tipo de Dato - LISTA

AGREGAR AL FINAL




```
Procedure incognita (var pI:listaE num:integer);  
Var  
    nuevo,act,ant:listaE;  
Begin  
    new (nuevo); nuevo^.elem:= num; nuevo^.sig:=nil;  
  
    if (pI = nil) then  
        pI:= nuevo  
    else begin  
        act:= pI; ant:=pI;  
        while (act <> nil) do begin  
            ant:=act;  
            act:= act^.sig;  
        end;  
        ant^.sig:= nuevo;  
    end;  
End;
```



Qué hace?

## AGREGAR AL FINAL EN UNA LISTA (OPCION 2)



 Cada vez que debo agregar un nuevo elemento al final de la lista, se tiene que recorrer la misma de manera completa hasta llegar al final.

**SE PUEDE MEJORAR?**

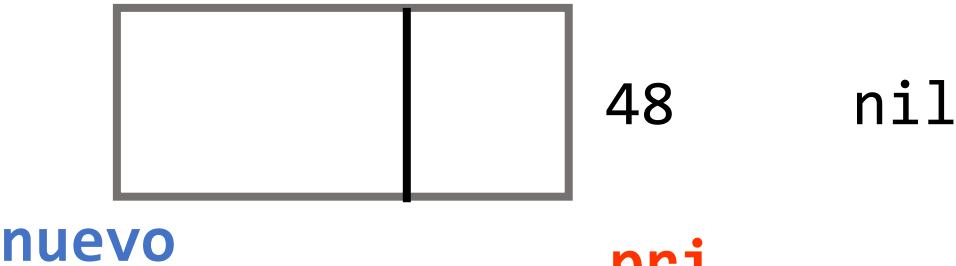
*Cómo lo escribo?*




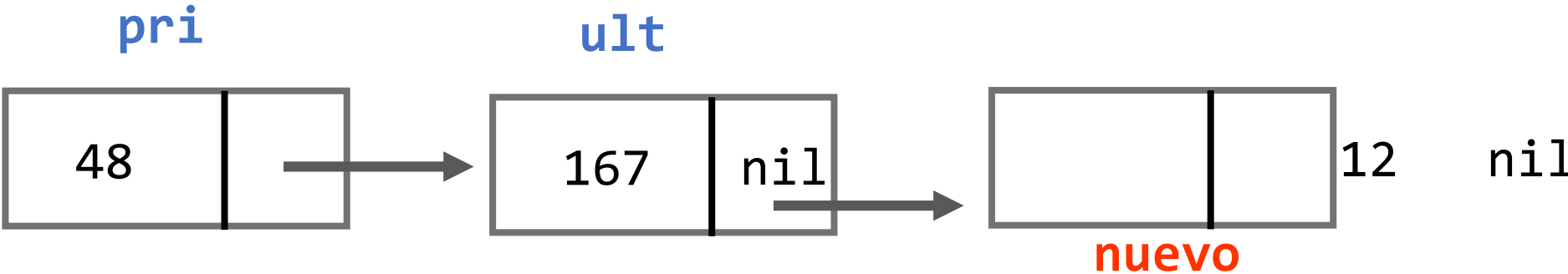
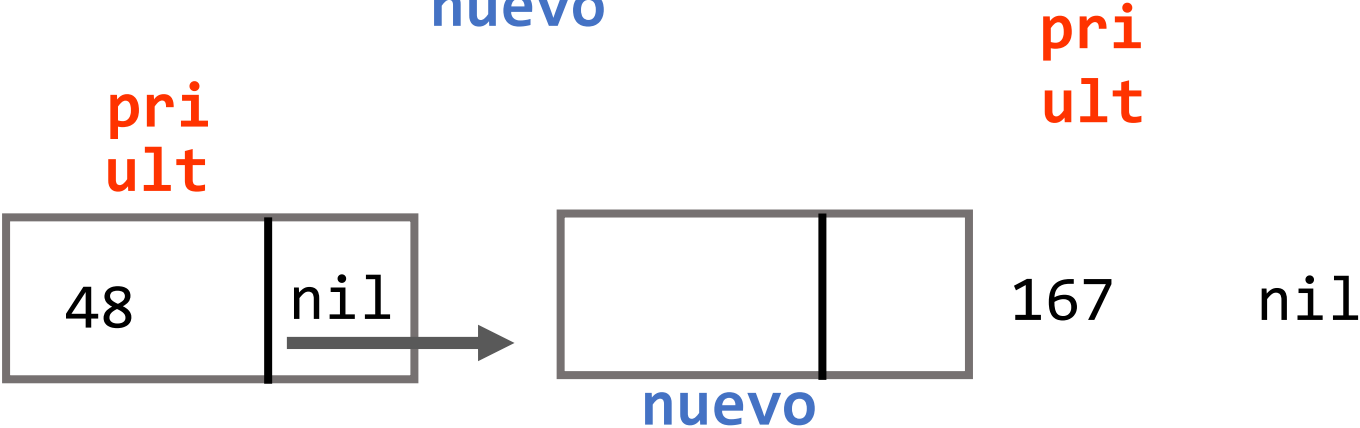
nuevo

## AGREGAR AL FINAL EN UNA LISTA (OPCION 2)

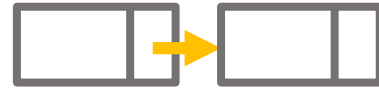
 `pri = nil`



 `pri <> nil`



*Cómo lo escribo?*



### AGREGAR AL FINAL EN UNA LISTA (OPCION 2)

Implica generar un nuevo nodo y agregarlo como último elemento de la lista.

Reservo espacio en memoria **nuevo elemento**.

**si** (es el primer elemento a agregar)

asigno al puntero inicial la dirección del **nuevo elemento**.

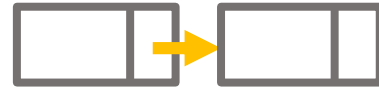
asigno al puntero final la dirección del **nuevo elemento**.

**sino**

actualizo como siguiente del puntero final al **nuevo elemento**

actualizo el la dirección del puntero final





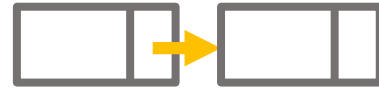
## AGREGAR AL FINAL EN UNA LISTA (opción 2)

Implica generar un nuevo nodo y agregarlo como último elemento de la lista.

```
Program uno;  
  
Type listaE= ^datosEnteros;  
  
    datosEnteros= record  
        elem:integer;  
        sig:listaE;  
    end;  
  
Var  
    pri,ult: listaE;  
    num:integer;  
Begin  
    crear(pri);  
    read (num);  
    agregarAlFinal2 (pri, ult, num);  
End.
```

# CADP – Tipo de Dato - LISTA

AGREGAR AL FINAL-2



```
procedure agregarAlFinal2 (var pI,pU:listaE; num:integer);
Var
    nuevo:listaE;

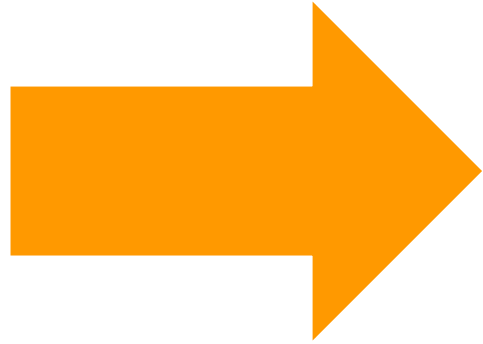
Begin
    new (nuevo); nuevo^.elem:= num; nuevo^.sig:=nil;

    if (pI = nil) then begin
        pI:= nuevo;
        pU:= nuevo;
    end
    else begin
        pU^.sig:=nuevo;
        pU:= nuevo;
    end;

End;
```



## Opción 1



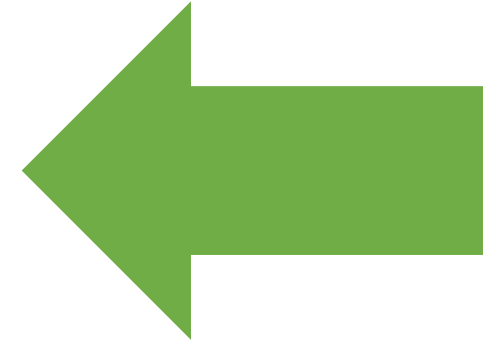
Genero espacio para el nuevo elemento

Recorro la lista hasta llegar al último elemento.

Reasigno los punteros

*Cuál elijo?*

## Opción 2

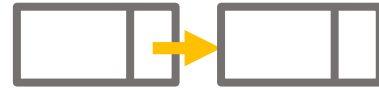


Genero espacio para el nuevo elemento

Utilizo un puntero que mantiene la dirección del último elemento.

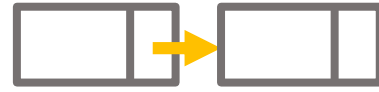
Reasigno los punteros.

Reasigno la dirección del último elemento



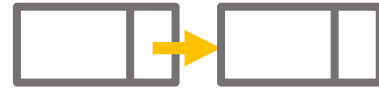
Suponga que dada una lista de enteros se quiere realizar un módulo que devuelva la lista donde cada elemento ha sido multiplicado por 3. Son correctos los siguientes módulos?

```
Procedure triplicar (var p:listaE);  
Begin  
    while (p^.sig <>nil) do  
        begin  
            p^.elem:= p^.elem *3;  
            p:= p^.sig;  
        end;  
End;
```



Suponga que dada una lista de enteros se quiere realizar un módulo que devuelva la lista donde cada elemento ha sido multiplicado por 3. Son correctos los siguientes módulos?

```
Procedure triplicar (var p:listaE);  
Begin  
    while (p <> nil) do  
        begin  
            p^.elem:= p^.elem *3;  
            p:= p^.sig;  
        end;  
End;
```



Suponga que dada una lista de enteros se quiere realizar un módulo que devuelva la lista donde cada elemento ha sido multiplicado por 3. Son correctos los siguientes módulos?

```
Procedure triplicar (p:listaE);  
Begin  
  while (p <> nil) do  
    begin  
      p^.elem:= p^.elem *3;  
      p:= p^.sig;  
    end;  
End;
```



Suponga que dada una lista de enteros se quiere realizar un módulo que devuelva la lista donde cada elemento ha sido multiplicado por 3. Son correctos los siguientes módulos?

```
function triplicar (p:listaE):listaE;  
Begin  
    while (p <> nil) do  
        begin  
            p^.elem:= p^.elem *3;  
            p:= p^.sig;  
        end;  
    triplicar:= p;  
End;
```



## BUSCAR UN ELEMENTO

Implica recorrer la lista desde el comienzo pasando nodo a nodo hasta encontrar el elemento buscado o que se termine la lista.

```
Program uno;
```

```
Type listaE= ^datosEnteros;
```

```
    datosEnteros= record  
        elem:integer;  
        sig:listaE;  
    end;
```

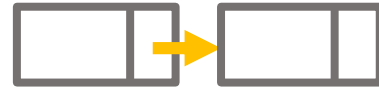
```
Var
```

```
    pri: listaE; {Memoria estática reservada}
```



# CADP – Tipo de Dato - LISTA

BUSCAR



```
Program uno;
```

```
Type listaE= ^datosEnteros;
```

```
    datosEnteros= record
                    elem:integer;
                    sig:listaE;
                end;
```

```
Var
```

```
    pri: listaE;
    num:integer;
    esta: boolean;
```

```
Begin
```

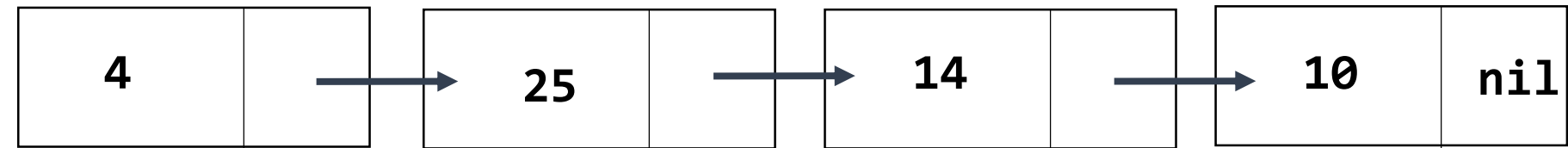
```
    crear (pri);
    cargarLista (pri);
    read (num);
```

```
    esta:= buscar (pri,num);
```

```
End.
```

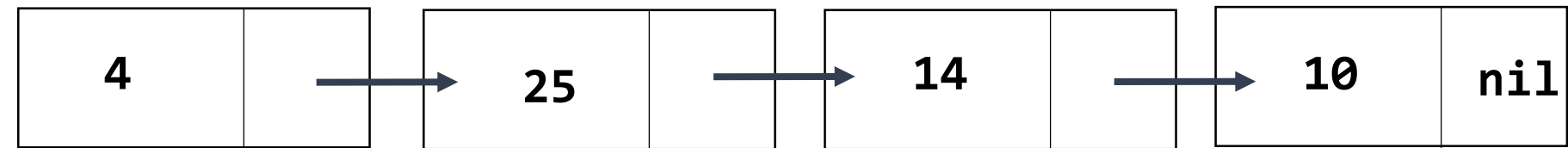
# CADP – Tipo de Dato - LISTA

BUSCAR EN UNA LISTA 



Num = 14

PI  
aux



Num = 3

PI  
aux



### BUSCAR UN ELEMENTO

Inicializo una variable **auxiliar** con la dirección del puntero inicial

mientras (no sea el final de la lista y no encuentre el elemento)

    si es el elemento buscado detengo la búsqueda

    sino

        avanzo al siguiente elemento

# CADP – Tipo de Dato - LISTA

## BUSCAR UN ELEMENTO



```
function buscar (pI: listaE; valor:integer):boolean;  
Var  
  aux:listaE;  
  encontré:boolean;  
  
Begin  
  encontré:= false;  
  aux:= pI;  
  while ((aux <> nil) and (encontré = false)) do  
    begin  
      if (aux^.elem = valor) then encontré:=true  
      else  
        aux:= aux^.sig;  
      end;  
    buscar:= encontré;  
  end;  
end;
```

*Funciona si la lista  
que recibo es vacía?*

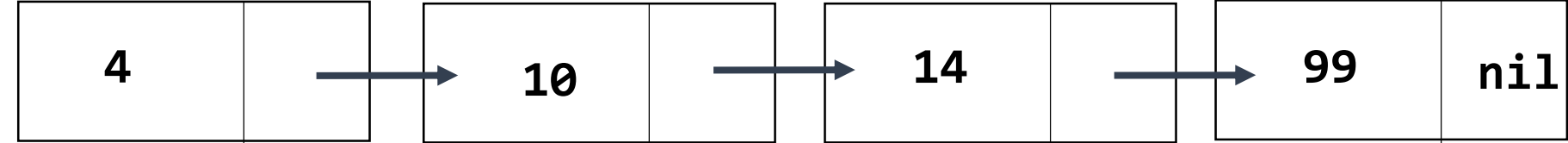
*Necesito usar aux?*



**Funciona si la lista está ordenada?  
Si funciona es la mejor solución?**

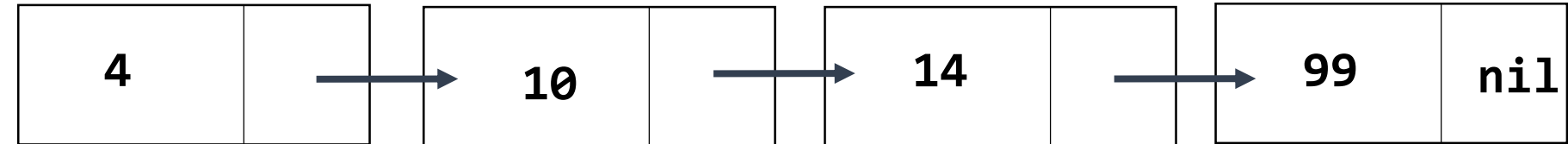
# CADP – Tipo de Dato - LISTA

BUSCAR EN UNA LISTA 



Num = 14

PI aux



Num = 7

PI aux

# CADP – Tipo de Dato - LISTA

## BUSCAR UN ELEMENTO



```
function buscar (pI: listaE; valor:integer):boolean;
```

```
Var
```

```
    aux:listaE;
```

```
    encontré:boolean;
```

```
Begin
```

```
    encontré:= false;
```

```
    aux:= pI;
```

```
    while ((aux <> nil) and (aux^.elem < valor)) do
```

```
        begin
```

```
            aux:= aux^.sig;
```

```
        end;
```

```
    if (aux <> nil) and (aux^.elem = valor) then encontré:= true;
```

```
    buscar:= encontré;
```

```
end;
```

**Funciona si la lista  
que recibo es  
vacía?**

**Necesito respetar el  
orden de la doble  
condición?**

**Necesito chequear las  
dos condiciones al  
salir del while?**



**A la búsqueda en que estructura  
es igual esta búsqueda?**



### INSERTAR UN ELEMENTO

Se necesita que la estructura tenga un orden e implica agregar el elemento a la lista de manera que la misma siga ordenada.

```
Program uno;
```

```
Type listaE= ^datosEnteros;
```

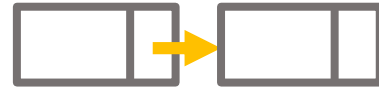
```
    datosEnteros= record  
        elem:integer;  
        sig:listaE;  
    end;
```

```
Var
```

```
    pri: listaE; {Memoria estática reservada}
```

# CADP – Tipo de Dato - LISTA

## INSERTAR EN UNA LISTA



Existen 4 casos:

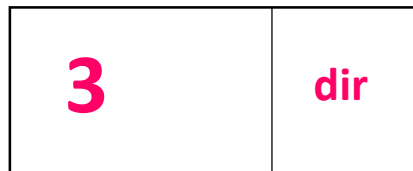
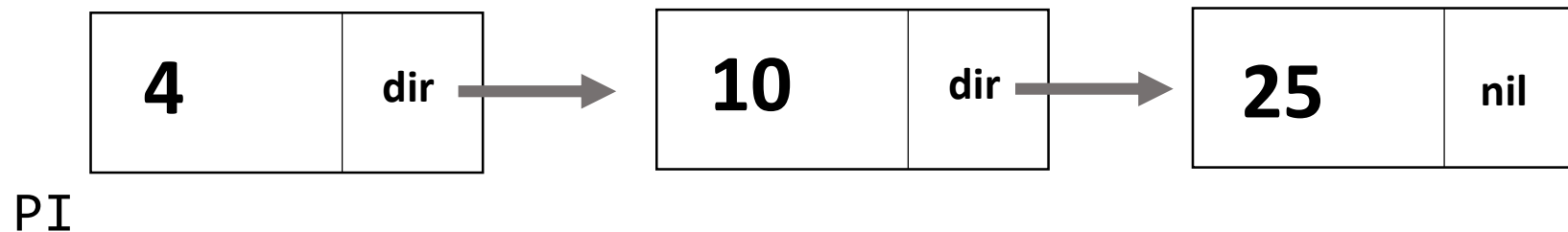
1. La lista está vacía.

Si la lista **NO** está vacía: se debe recorrer hasta encontrar el lugar donde va el elemento:

2. El elemento a insertar va al comienzo de la lista.

3. El elemento a insertar va al medio de la lista.

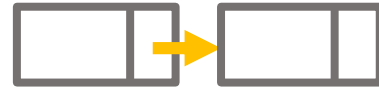
4. El elemento a insertar va al final de la lista.





# CADP – Tipo de Dato - LISTA

## INSERTAR EN UNA LISTA



**Caso 1:** la lista está vacía

PI = nil



4

nil

nuevo

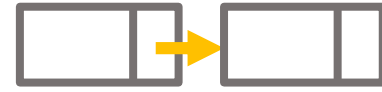
PI

Generar un nuevo nodo (nuevo).

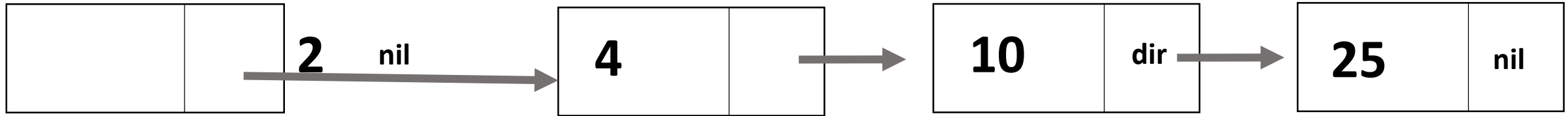
Asignar a la dirección del puntero inicial (PI) la del nuevo nodo (nuevo)

# CADP – Tipo de Dato - LISTA

## INSERTAR EN UNA LISTA



**Caso 2:** la lista no está vacía, y el elemento va al principio de la lista.



nuevo

PI  
actual

Generar un nuevo nodo (nuevo).

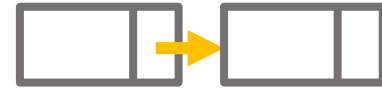
Asignar a la dirección del puntero siguiente del nuevo la dirección del nodo inicial (PI).

Actualizar con la dirección del nuevo nodo la dirección del puntero inicial (PI)

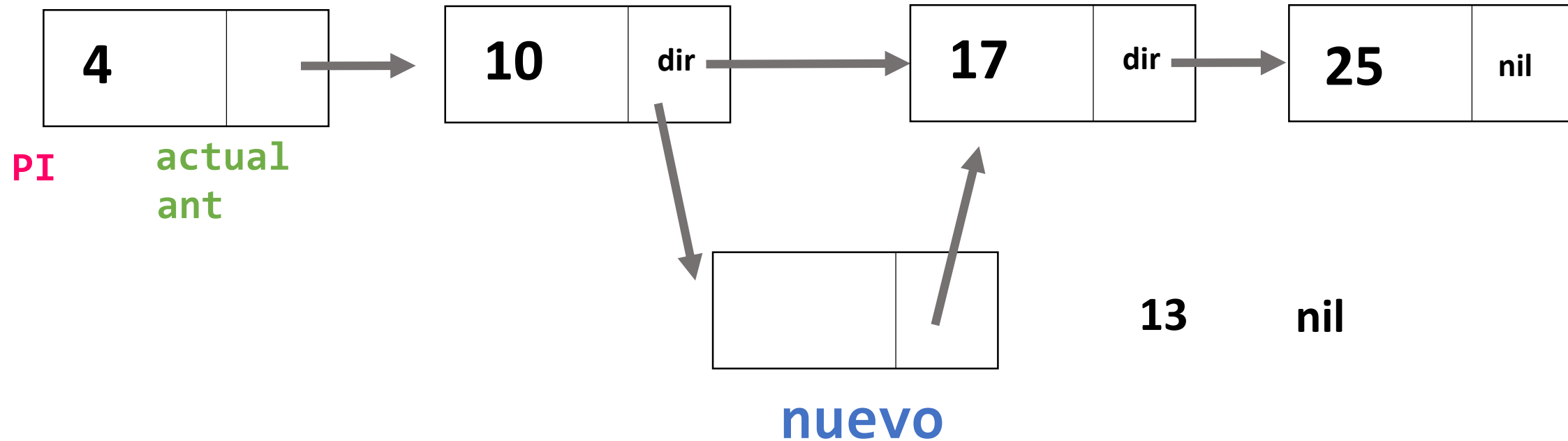
Mirar donde  
queda actual

# CADP – Tipo de Dato - LISTA

## INSERTAR EN UNA LISTA



**Caso 3:** la lista no está vacía, y el elemento va en el medio de la lista.



Generar un nuevo nodo (nuevo).  
Preparo los punteros para el recorrido  
Busco la posición  
Reasigno punteros

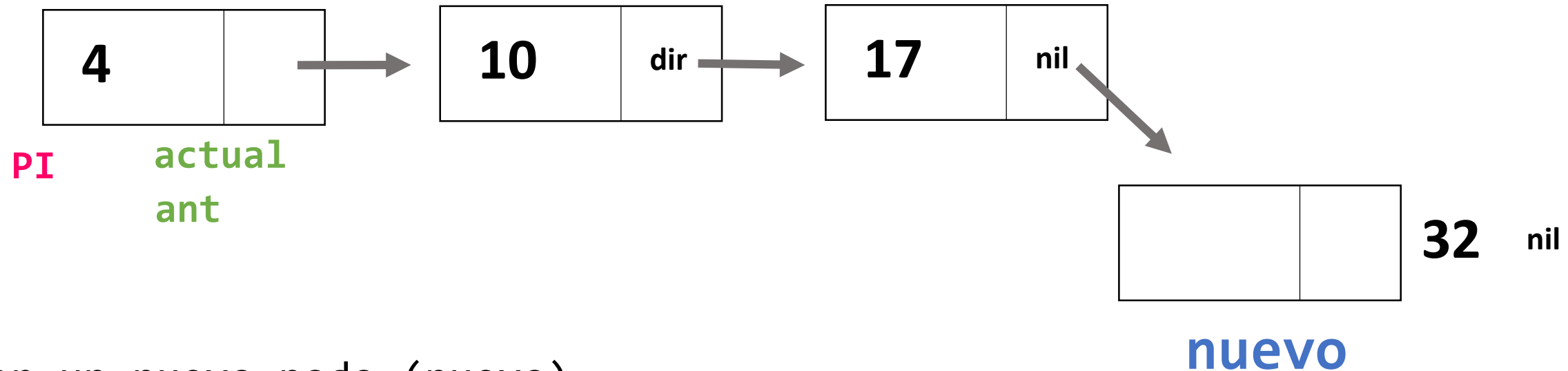
**Mirar donde  
queda actual**

# CADP – Tipo de Dato - LISTA

## INSERTAR EN UNA LISTA

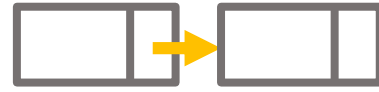


**Caso 3:** la lista no está vacía, y el elemento va en el medio de la lista.



Generar un nuevo nodo (nuevo).  
Preparo los punteros para el recorrido  
Busco la posición  
Reasigno punteros

Mirar donde  
queda actual



## INSERTAR UN ELEMENTO

Generar un nuevo nodo (nuevo).

Si la lista está vacía

Caso 1  $pI = \text{nil}$

Actualizo la dirección del nodo inicial (PI)

Sino

Preparo los punteros para el recorrido (ant, actual)

Busco la posición

Si va al principio

Caso 2  $\text{actual} = pI$

Asigno como siguiente del nodo nuevo al nodo inicial

Actualizo la dirección del nodo inicial (PI)

Si va en el medio

Caso 3  $\text{actual} \neq \text{nil}$

La dirección del siguiente del puntero ant es la dirección del nodo nuevo

La dirección del siguiente del nodo nuevo es la dirección del actual

sino

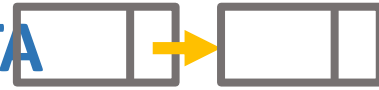
Caso 4  $\text{actual} \neq \text{nil}$

La dirección del siguiente del puntero ant es la dirección del nodo nuevo

La dirección del siguiente del nodo nuevo es la dirección nil

# CADP – Tipo de Dato - LISTA

INSERTAR EN UNA LISTA



```
Program uno;
```

```
Type listaE= ^datosEnteros;
```

```
    datosEnteros= record  
        elem:integer;  
        sig:listaE;  
    end;
```

```
Var
```

```
    pri: listaE;  
    num:integer;
```

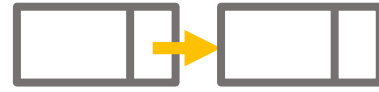
```
Begin
```

```
    crear(pri);  
    read (num);  
    insertar (pri,num);
```

```
End.
```

# CADP – Tipo de Dato - LISTA

## INSERTAR EN UNA LISTA



```
Procedure insertar (var pI:listaE valor:integer);
```

```
Var
```

```
    nuevo,actual,ant:listaE;
```

```
Begin
```

```
    new (nuevo); nuevo^.elem:= valor; nuevo^.sig:=nil;
```

```
    if (pI = nil) then
```

```
        pI:= nuevo
```

```
    else begin
```

```
        actual:= pI; ant:=pI;
```

```
        while (actual <> nil) and (actual^.elem < nuevo^.elem) do
```

```
            begin
```

```
                ant:=actual;
```

```
                actual:= actual^.sig;
```

```
            end;
```

```
        end;
```



**Caso 1**

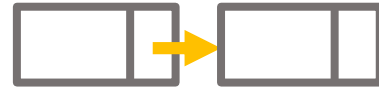


**Busco la posición**



# CADP – Tipo de Dato - LISTA

## INSERTAR EN UNA LISTA



```
if (actual = pI) then
```

```
begin
```

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

```
    pI:= nuevo;
```

```
end
```

```
else if (actual <> nil) then
```

```
begin
```

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

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

```
end
```

```
else
```

```
begin
```

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

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

```
end;
```

```
End;
```

}

Caso 2

}

Caso 3

}

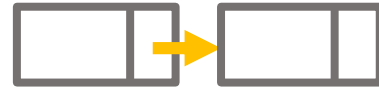
Caso 4

Todo junto



# CADP – Tipo de Dato - LISTA

## INSERTAR EN UNA LISTA



```
Procedure insertar (var pI:listaE valor:integer);
Var
    nuevo,actual,ant:listaE;
Begin
    new (nuevo); nuevo^.elem:= valor; nuevo^.sig:=nil;
    if (pI = nil) then
        pI:= nuevo
    else begin
        actual:= pI; ant:=pI;
        while (actual <> nil) and (actual^.elem < nuevo^.elem) do
            begin
                ant:=actual;
                actual:= actual^.sig;
            end;

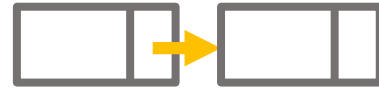
        if (actual = pI) then
            begin
                nuevo^.sig:= pI;
                pI:= nuevo;
            end
        else if (actual <> nil) then
            begin
                ant^.sig:=nuevo;
                nuevo^.sig:= actual;
            end
        else
            begin
                ant^.sig:= nuevo;
                nuevo^.sig:= nil;
            end;
    end;
End;
End;
```



Se puede mejorar?

# CADP – Tipo de Dato - LISTA

## INSERTAR EN UNA LISTA



```
Procedure insertar (var pI:listaE valor:integer);
Var
    nuevo,actual,ant:listaE;
Begin
    new (nuevo); nuevo^.elem:= valor; nuevo^.sig:=nil;
    if (pI = nil) then
        pI:= nuevo
    else begin
        actual:= pI; ant:=pI;
        while (actual <> nil) and (actual^.elem < nuevo^.elem) do
            begin
                ant:=actual;
                actual:= actual^.sig;
            end;

        if (actual = pI) then
            begin
                nuevo^.sig:= pI;
                pI:= nuevo;
            end
        else
            begin
                ant^.sig:=nuevo;
                nuevo^.sig:= actual;
            end
        end
    End;
End;
```



## ELIMINAR UN ELEMENTO

Implica recorrer la lista desde el comienzo pasando nodo a nodo hasta encontrar el elemento y en ese momento eliminarlo (dispose). El elemento puede no estar en la lista

```
Program uno;
```

```
Type listaE= ^datosEnteros;
```

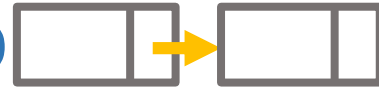
```
    datosEnteros= record  
        elem:integer;  
        sig:listaE;  
    end;
```

```
Var
```

```
    pri: listaE; {Memoria estática reservada}
```

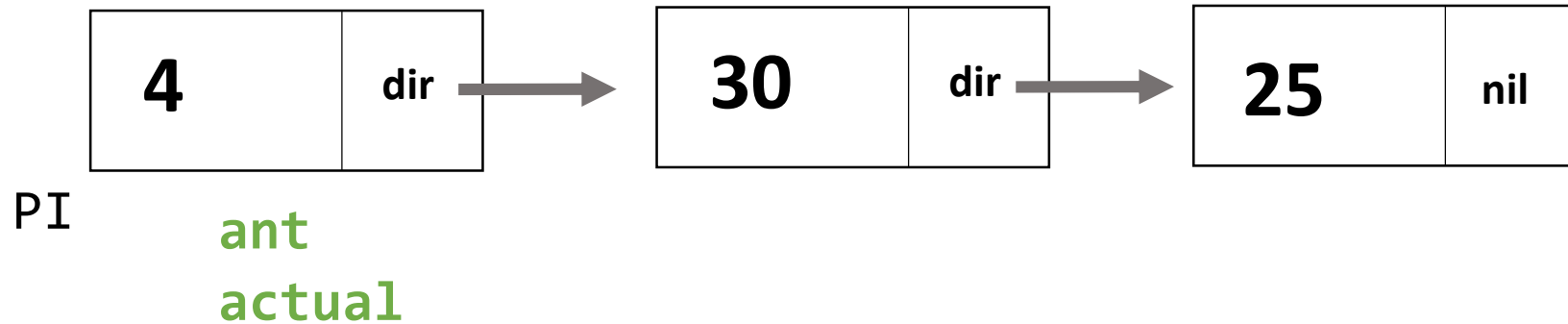
# CADP – Tipo de Dato - LISTA

ELIMINAR UNA ELEMENTO



Existen 2 casos:

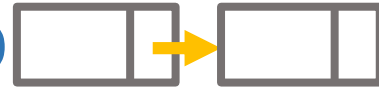
- 1.El elemento a eliminar sea el primer nodo de la lista.
- 2.El elemento a eliminar NO sea el primer nodo de la lista.



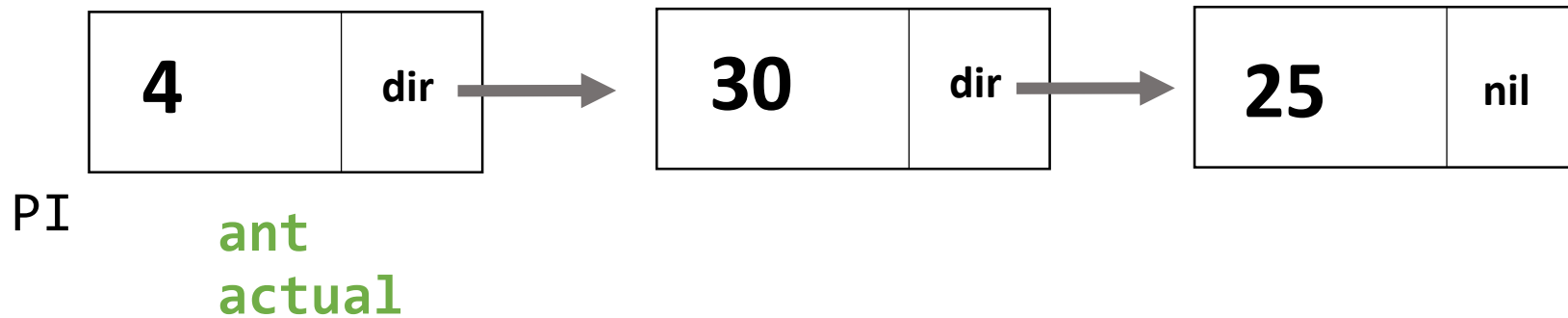
12

# CADP – Tipo de Dato - LISTA

## ELIMINAR UNA ELEMENTO



Existen 2 casos: caso 1 el elemento está al comienzo de la lista



4

Actualizar la dirección siguiente del puntero inicial pI con la dirección siguiente de actual.

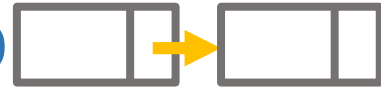
Actualizar la dirección del puntero inicial pI

Hacer el dispose

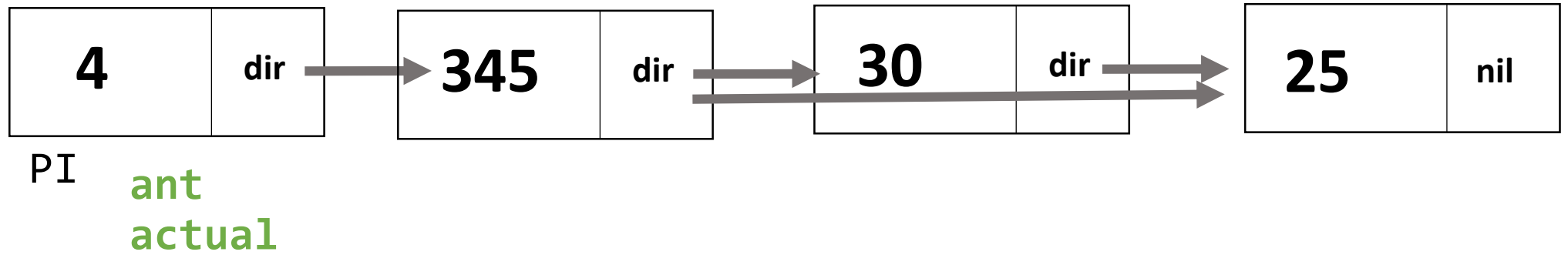
Mirar donde  
queda actual

# CADP – Tipo de Dato - LISTA

## ELIMINAR UNA ELEMENTO



Existen 2 casos: caso 1 el elemento está al comienzo de la lista



30

Actualizar la dirección siguiente del puntero ant con la dirección siguiente de actual.

Hacer el dispose

Mirar donde queda actual

# CADP – Tipo de Dato - LISTA

ELIMINAR EN UNA LISTA 



```
Program uno;
```

```
Type listaE= ^datosEnteros;
```

```
    datosEnteros= record  
        elem:integer;  
        sig:listaE;  
    end;
```

```
Var
```

```
    pri: listaE;
```

```
    num:integer;
```

```
Begin
```

```
    crear(pri);
```

```
    read (num);
```

```
    eliminar (pri,num);
```

```
End.
```

# CADP – Tipo de Dato - LISTA

ELIMINAR EN UNA LISTA 

```
Procedure eliminar(var pI:listaE valor:integer);
Var
  actual,ant:listaE;
Begin
  actual:=p;
  while (actual <> nil) and (actual^.elem <> valor) do
begin
  ant:=actual; actual:= actual^.sig;
end;
if (actual <> nil) then
  if (actual = pI) then begin
    pI:= pI^.sig; dispose (actual);
  end
  else begin
    ant^.sig:= actual^.sig;
    dispose (actual);
  end;
End;
```

*Por qué pI es pasado  
por referencia?*

*Funciona si la lista  
que recibo es vacía?*

**Caso 1**

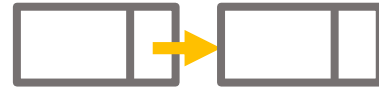
**Caso 2**

*Qué se puede  
mejorar?*



# CADP – Tipo de Dato - LISTA

## ELIMINAR EN UNA LISTA



```
Procedure eliminar(var pI:listaE valor:integer);
```

```
Var
```

```
    actual,ant:listaE;
```

```
Begin
```

```
    actual:=p;
```

```
    while (actual <> nil) do begin
```

```
        if (actual^.elem <> valor) then
```

```
            ant:=actual; actual:= actual^.sig;
```

```
        else begin
```

```
            if (actual <> nil) then
```

```
                begin
```

```
                    if (actual = pI) then
```

```
                        pI:= pI^.sig
```

```
                    else
```

```
                        ant^.sig:= actual^.sig;
```

```
dispose (actual);
```

```
actual:= ant;
```

```
end;
```



Y si la lista puede tener elementos repetidos?