

Conceptos de Algoritmos Datos y Programas

CADP – TIPOS DE DATOS - LISTA



Creación de una lista.

Agregar nodos al comienzo de la lista.

Recorrido de una lista.

Agregar nodos al final de la lista.

Buscar un elemento en una lista

Eliminar un elemento de una lista

Insertar un elemento en una lista ordenada



CADP – TEMAS



● Operación de 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.

Si la lista está desordenada seguramente la búsqueda se realizará hasta encontrar el elemento o hasta que se termina la lista.

Si la lista está ordenada seguramente la búsqueda se realizará hasta que se termina la lista o no se encuentre un elemento mayor al buscado.

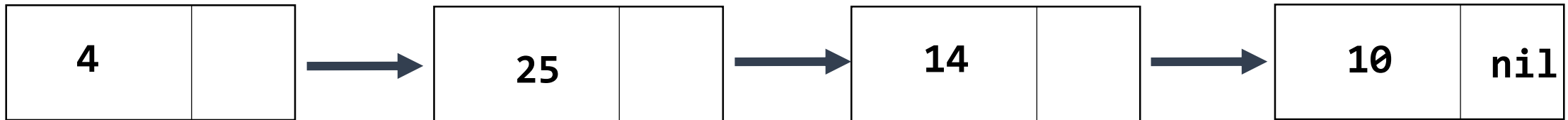
Existen 3 casos:

- que elemento a eliminar no se encuentre en la lista
- que elemento a eliminar sea el primero de la lista
- que elemento a eliminar no sea el primero en la lista



anterior

actual



Pri

num = 20

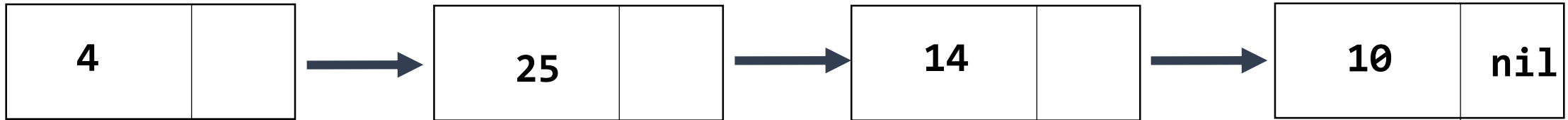
Caso 1:

Recorrí toda la lista y el elemento a eliminar no se encuentra.

OBSERVAR QUE actual QUEDÓ EN nil



anterior
actual



Pri

num = 4

Caso 2:

Empiezo a recorrer la lista.

Mientras (no encuentro el elemento a borrar) y (no se termine la lista)

el puntero anterior toma la dirección del puntero actual

avanzo el puntero actual

Como (el elemento está) y (es el primer elemento)

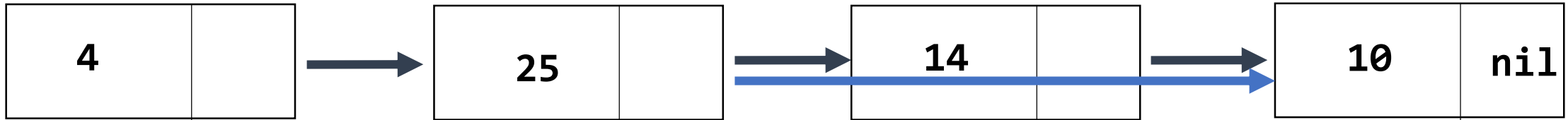
actualizo el puntero inicial de la lista

elimino la dirección del puntero actual

OBSERVAR QUE **actual** HABIA QUEDADO IGUAL A **pri**



anterior
actual



Pri

num = 14

Caso 3:

Empiezo a recorrer la lista.

Mientras (no encuentro el elemento a borrar) y (no se termine la lista)
el puntero anterior toma la dirección del puntero actual
avanzo el puntero actual

Como (el elemento está) y (NO es el primer elemento)
actualizo el siguiente del puntero anterior con el siguiente de actual
elimino la dirección del puntero actual

OBSERVAR QUE actual HABIA QUEDADO <> nil y de pri



ELIMINAR EN UN LISTA DESORDENADA

Comienzo a recorrer la lista desde el nodo inicial.


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

el puntero anterior toma la dirección del puntero actual
avanzo el puntero actual

si (encontré el elemento) entonces


si (es el primer nodo) entonces

actualizo el puntero inicial de la lista

elimino la dirección del puntero actual 

sino

actualizo el siguiente del puntero anterior con el siguiente de actual

elimino la dirección del puntero actual 



ELIMINAR EN UN LISTA DESORDENADA

Comienzo a recorrer la lista desde el nodo inicial.

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

 el puntero anterior toma la dirección del puntero actual
 avanzo el puntero actual

si (encontré el elemento) entonces

 si (es el primer nodo) entonces

 actualizo el puntero inicial de la lista

 sino

 actualizo el siguiente del puntero anterior con el siguiente de actual

 elimino la dirección del puntero actual



```
Program uno;
```

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

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

```
Var
```

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

```
Begin
```

```
    crear (pri);
    cargar (pri); //se dispone
    read (num);
    eliminar(pri,num);
```

```
End.
```



```
procedure eliminar (Var pI: listaE; valor:integer);
Var
  actual,ant:listaE;

Begin
  actual:=pI;
  while (actual <> nil) and (actual^.elem <> valor) do begin
    ant:=actual;
    actual:= actual^.sig;
  end;
  if (actual <> nil) then
    if (actual = pI) then
      pI:= pI^.sig;
    else
      ant^.sig:= actual^.sig;

      dispose (actual);

  End;
```

**Qué modifico si el elemento
puede repetirse?**

CADP – TIPOS DE DATOS - LISTA

ELIMINAR



```
procedure eliminar (Var pI: listaE; valor:integer);
```

```
Var
```

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

```
Begin
```

```
    actual:=pI;
```

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

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

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

```
        end;
```

```
        else begin
```

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

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

```
            else
```

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

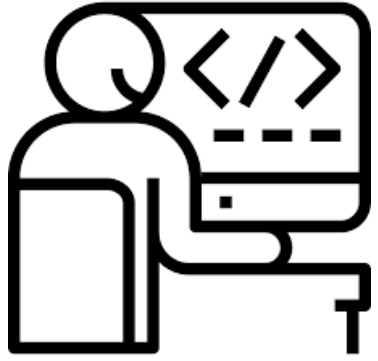
```
            dispose (actual);
```

```
            actual:= ant;
```

```
        end;
```

```
End;
```

Qué modifico si la lista está
ordenada y ele elemento un
única vez ?



Conceptos de Algoritmos Datos y Programas

CADP – TIPOS DE DATOS - LISTA



Creación de una lista.

Agregar nodos al comienzo de la lista.

Recorrido de una lista.

Agregar nodos al final de la lista.

Buscar un elemento en una lista

Eliminar un elemento de una lista

Insertar un elemento en una lista ordenada



CADP – TEMAS



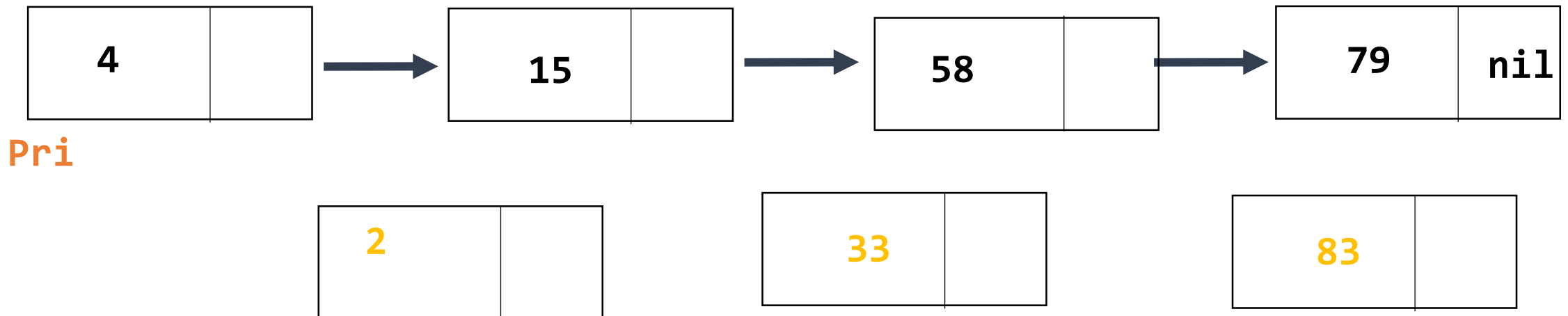
● Operación de INSERTAR un ELEMENTO



Implica agregar un nuevo nodo a una lista ordenada por algún criterio de manera que la lista siga quedando ordenada.

Existen 4 casos:

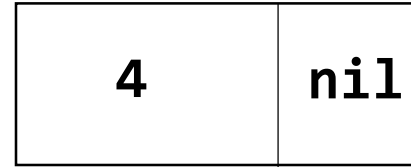
- que la lista esté vacía.
- que elemento vaya al comienzo de la lista (es menor al 1er nodo de la lista)
- que elemento vaya al “medio” de la lista (es menor al último nodo de la lista)
- que elemento vaya al final de la lista (es mayor al último nodo de la lista)





CASO 1: lista vacía

`Pri = nil`



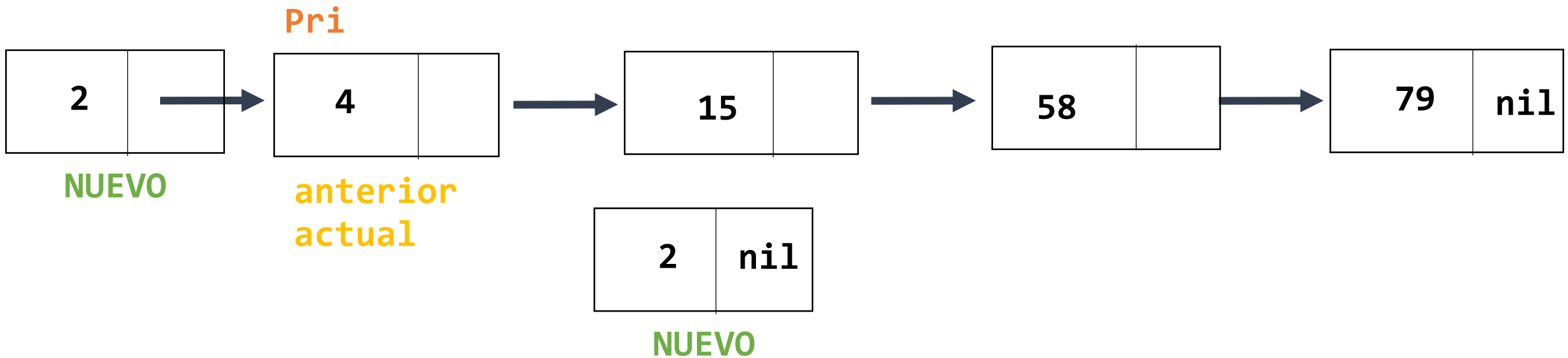
NUEVO

Generar un nuevo nodo (NUEVO).

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



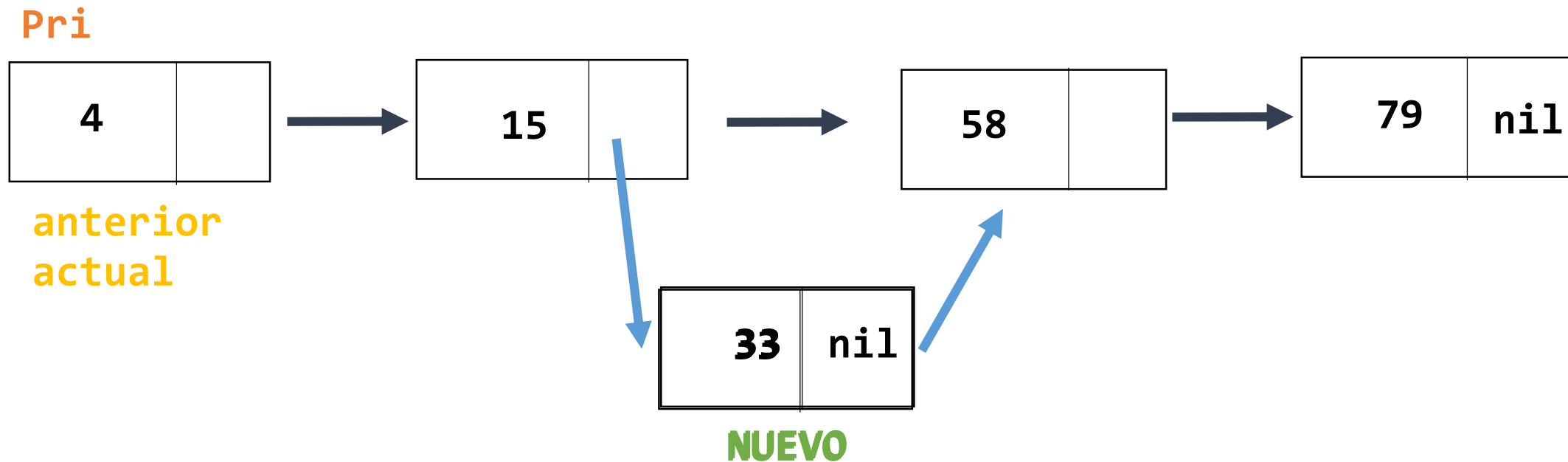
CASO 2: lista no vacía, va al principio



Generar un nuevo nodo (nuevo). Preparar punteros para el recorrido.
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)
OBSERVAR QUE actual HABIA QUEDADO = pri



CASO 3: lista no vacía, va en el “medio”



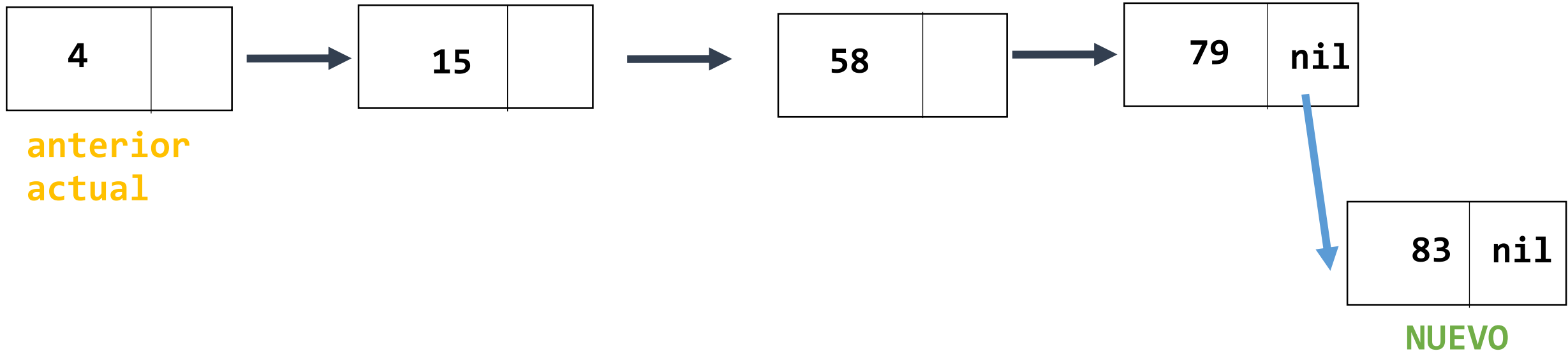
Generar un nuevo nodo (nuevo). Preparo los punteros para el recorrido
 Recorro hasta encontrar la posición
 Reasigno punteros, el siguiente de anterior es NUEVO y el siguiente
 de NUEVO es actual.

OBSERVAR QUE actual HABIA QUEDADO <> nil



CASO 4: lista no vacía, va al final

Pri



Generar un nuevo nodo (nuevo). Preparo los punteros para el recorrido
Recorro hasta encontrar la posición
Reasigno punteros, el siguiente de anterior es NUEVO y el siguiente de NUEVO es nil.

OBSERVAR QUE actual HABIA QUEDADO = nil

CADP – TIPOS DE DATOS - LISTA

INSERTAR



Generar un nuevo nodo (NUEVO).

Si la lista está vacía

Actualizo la dirección del nodo inicial (pri)

Caso 1 pri=nil

Sino

Preparo los punteros para el recorrido (anterior,actual)

Recorro hasta encontrar la posición.

Si va al principio

Asigno como siguiente del nodo nuevo al nodo inicial

Actualizo la dirección del nodo inicial (pri)

Caso 2 actual=pri

Si va en el medio

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

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

Caso 3 actual <> nil

sino

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

La dirección del siguiente del nodo nuevo es la dirección nil

Caso 4 actual <> nil

CADP – TIPOS DE DATOS - LISTA

INSERTAR



```
Program uno;
```

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

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

```
Var
```

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

```
Begin
```

```
    crear (pri);
    cargar (pri); //se dispone
    read (num);
    insertar(pri,num);
```

```
End.
```



```
procedure insertar (Var pI: listaE; valor:integer);
```

```
Var
```

```
    actual,anterior,nuevo: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
```

```
                anterior:=actual;
```

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

```
            end;
```

```
    end;
```



Caso 1 pri=nil



BUSCO LA
POSICION





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



Caso 2
pri=actual

```
else if (actual <> nil) then
begin
  anterior^.sigg := nuevo;
  nuevo^.sigg := actual;
end;
```



Casos 3 y 4
actual <> nil

```
End;
else
begin
  anterior^.sig := nuevo;
  nuevo^.sig := actual;
end;
End;
```



Caso 4
actual = nil

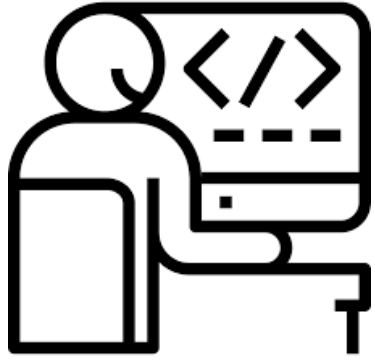


En el caso 4
cuánto vale
actual?



```
procedure insertar (Var pI: listaE; valor:integer);
Var
  actual,anterior,nuevo: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
        anterior:=actual;
        actual:= actual^.sig;
      end;
    end;
    if (actual = pI) then
      begin
        nuevo^.sig:= pI;  pI:= nuevo;
      end
    else
      begin
        anterior^.sig:= nuevo;  nuevo^.sig:= actual;
      end;
    end;
  End;
```



Conceptos de Algoritmos Datos y Programas

CADP – TIPOS DE DATOS - LISTA



Creación de una lista.

Agregar nodos al comienzo de la lista.

Recorrido de una lista.

Agregar nodos al final de la lista.

Buscar un elemento en una lista

Eliminar un elemento de una lista

Insertar un elemento en una lista ordenada



CADP – TEMAS



- Operación de BUSCAR un ELEMENTO



Significa recorrer la lista desde el primer nodo buscando un valor que puede o no estar. Se debe tener en cuenta si la lista está o no ordenada.

LISTA Desordenada

- Se debe recorrer toda la lista (en el peor de los casos), y detener la búsqueda en el momento que se encuentra el dato buscado o en el que la lista se terminó.

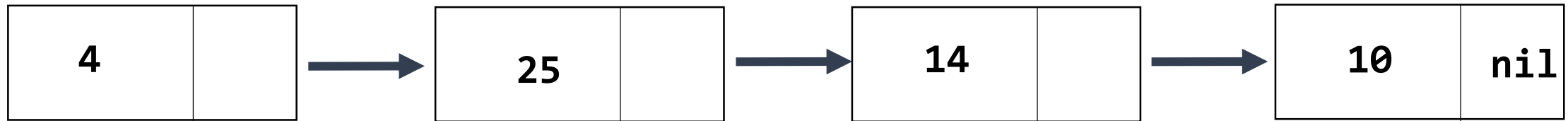
LISTA Ordenada

- Se debe recorrer la lista teniendo en cuenta el orden. La búsqueda se detiene cuando se termina la lista o el elemento buscado es mayor al elemento actual.



BUSQUEDA LISTA DESORDENADA

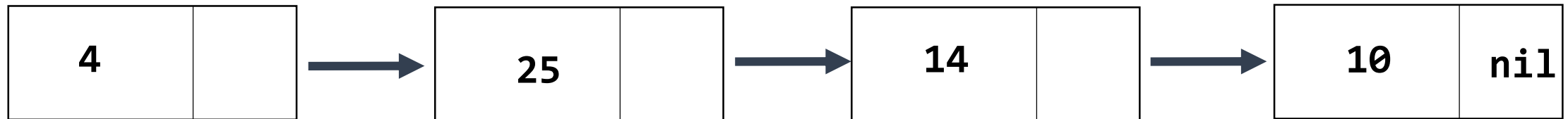
aux



Pri

num = 14

aux



Pri

num = 3



BUSQUEDA LISTA DESORDENADA

Comienzo a recorrer la lista desde el nodo inicial.

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

 si (es el elemento buscado) entonces

 detengo la búsqueda

 sino

 avanzo al siguiente elemento

*Qué módulo
utilizo?*



```
Program uno;
```

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

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

```
Var  
    pri: listaE;  
    num:integer;  
  
Begin  
    crear (pri);  
    cargar (pri); //se dispone  
    read (num);  
    if (buscar(pri,num)) then write ("el elemento existe");  
End.
```




```
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;
```

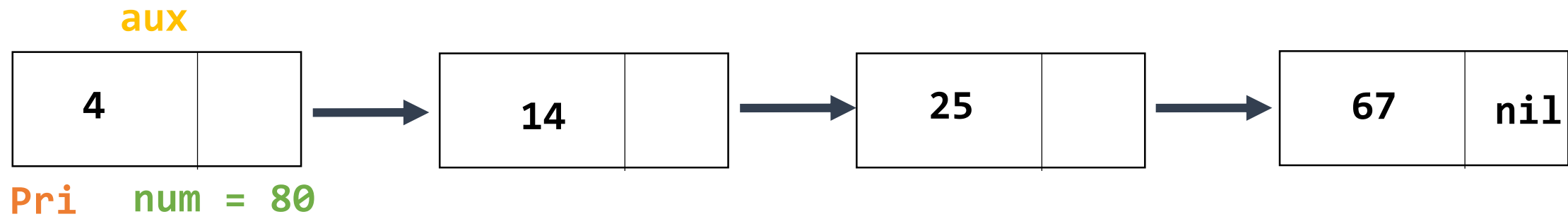
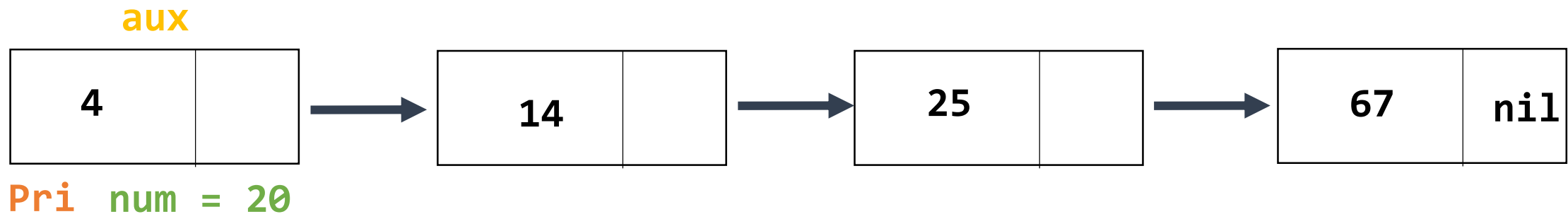
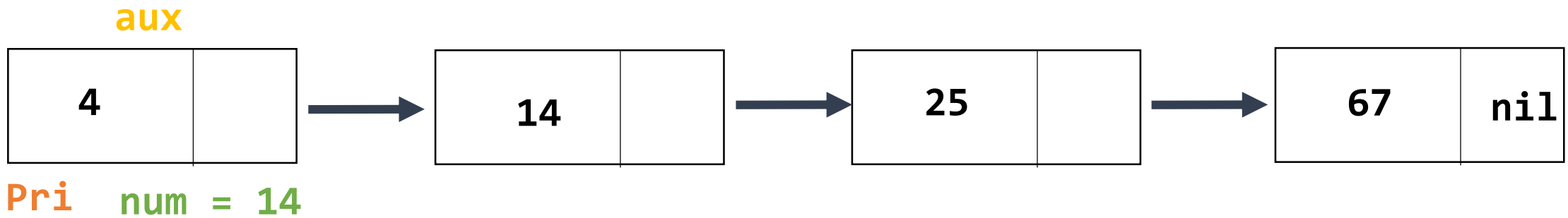
*Funciona si la
lista que recibo
es vacía?*

*Necesito
usar aux?*

**Qué modifiko si la lista está
ordenada?**



BUSQUEDA LISTA ORDENADA





```
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 usar
aux?*

*Es necesario respetar el
orden de las condiciones?*

*Necesito el chequeo
del final?*

**Buscar en una lista tiene las mismas
características que buscar en un vector**