



# Conceptos de Algoritmos Datos y Programas



# CADP – TEMAS



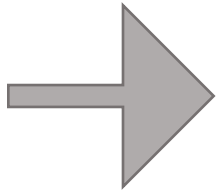
- Estructura de Datos - LISTA
- Características de una LISTA
- Operaciones de una LISTA

# CADP – TIPOS DE DATOS - LISTA

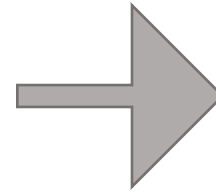


Realizar un programa que lea números que representan edades de personas hasta leer la edad -1. Finalizada la lectura se quiere informar cual fue la edad máxima leída.

10  
4  
57  
62  
39  
-1



**Dónde  
almaceno las  
edades?**



Necesito una estructura que pueda ir agregando datos y por lo tanto su tamaño pueda ir variando en la ejecución del programa (estructura dinámica)

**LISTA**

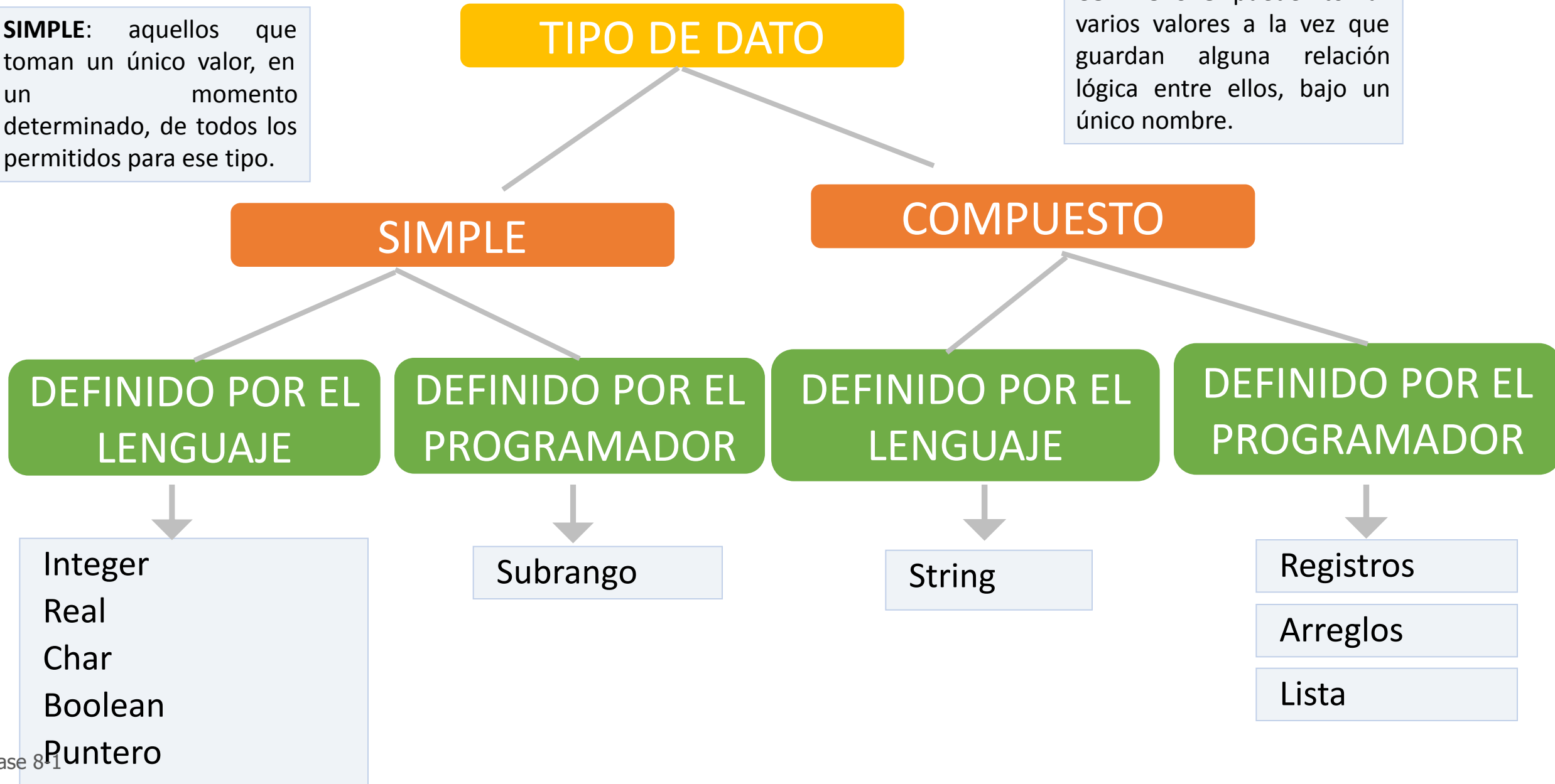


# CADP – TIPOS DE DATOS - LISTA



**SIMPLE:** aquellos que toman un único valor, en un momento determinado, de todos los permitidos para ese tipo.

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

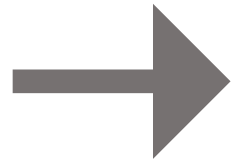


# CADP – TIPOS DE DATOS - LISTA



Es una colección de nodos. Cada nodo contiene un elemento (valor que se quiere almacenar en la lista) y una dirección de memoria dinámica que indica donde se encuentra el siguiente nodo de la lista.

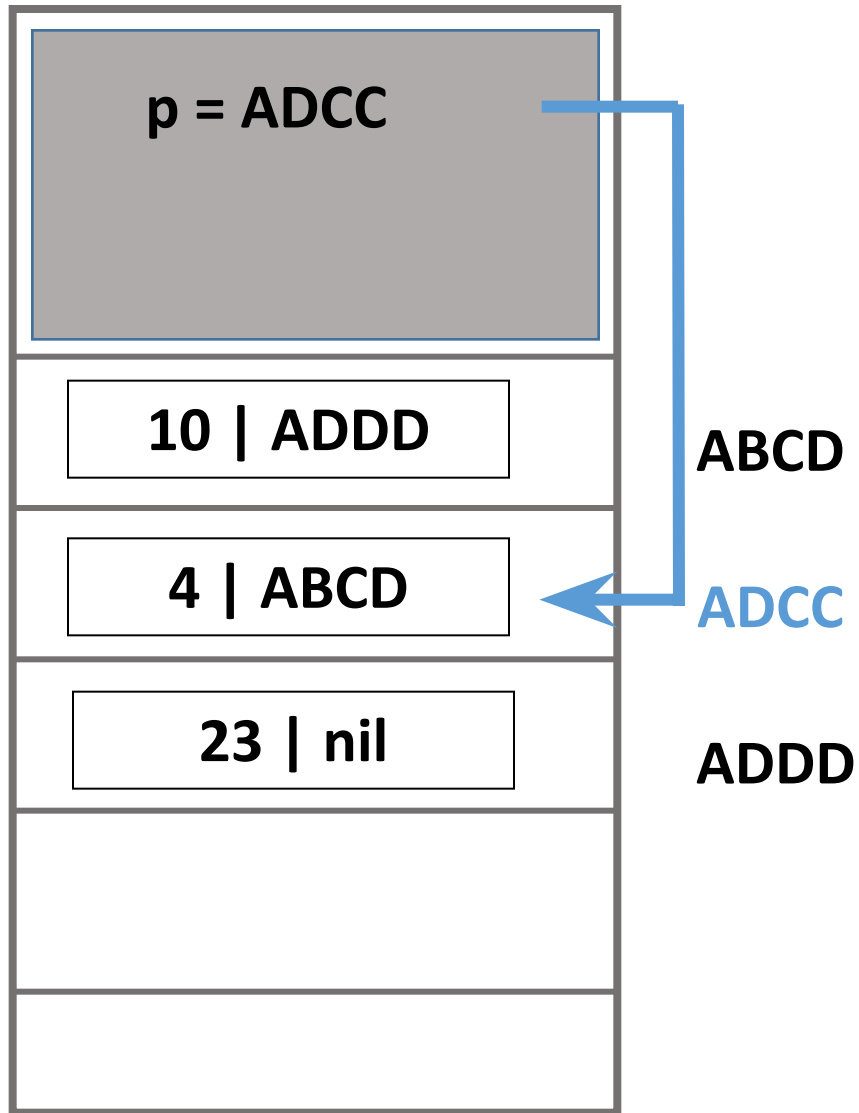
Toda lista tiene un nodo inicial.



Los **nod**os que la componen pueden no ocupar posiciones contiguas de memoria. Es decir, pueden aparecer dispersos en la memoria, pero mantienen un orden lógico interno.

Gráficamente ...

# CADP – TIPOS DE DATOS - LISTA



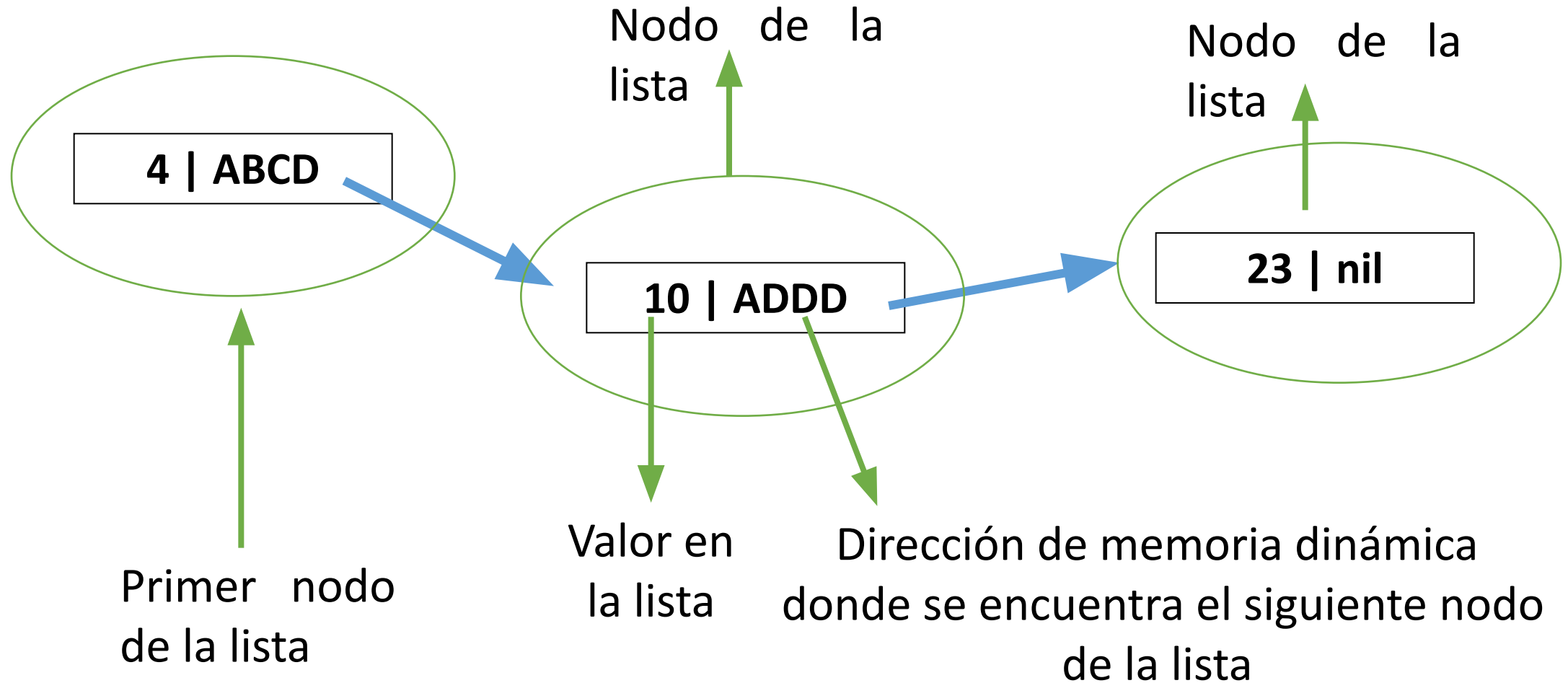
En **memoria estática** se declara una variable tipo PUNTERO (ya que son las única que pueden almacenar direcciones). La dirección almacenada en esa variable representa la dirección donde comienza la lista. Inicialmente ese puntero no contiene ninguna dirección.



Luego a medida que se quiere agregar elementos a la lista (nodo), se reserva una dirección de **memoria dinámica** y se carga el valor que se quiere guardar.

El último nodo de la lista indica que la dirección que le sigue es nil.

# CADP – TIPOS DE DATOS - LISTA



# CADP – TIPOS DE DATOS - LISTA



## CARACTERISTICAS

Cómo se  
declara?

**HOMOGENEA**



Los elementos pueden ser del mismo tipo .

**DINAMICA**



El tamaño puede cambiar durante la ejecución del programa.

**LINEAL**



Cada nodo de la lista tiene un nodo que lo sigue (salvo el último) y uno que lo antecede (salvo el primero).

**SECUENCIAL**



El acceso a cada elemento es de manera secuencial, es decir, para acceder al elemento 5 (por ejemplo) debo pasar por los 4 anteriores.



Cada vez que se necesite agregar un nodo se deberá reservar memoria dinámica (new) y cuando se quiera eliminar un nodo se debe liberar la memoria dinámica (dispose) .





```
Program uno;
```

```
Type
```

```
    nombreTipo= ^nombreNodo;
```

```
    nombreNodo = record
```

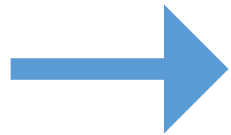
```
        elemento: tipoElemento;
```

```
        punteroSig: nombreTipo;
```

```
    end;
```

```
Var
```

```
    Pri: nombreTipo;
```



**tipoElemento** es cualquiera de los tipos vistos (entero,char,boolean,registro,arreglo,real,subrangel).

Es una estructura recursiva.

El orden de la declaración debe respetarse



```
Program uno;
```

```
Type
```

```
  listaE= ^nodo;
```

```
  nodo = record
```

```
    elemento: integer;
```

```
    punteroSig: listaE;
```

```
  end;
```

```
Var
```

```
  Pri: listaE;
```





Program dos;

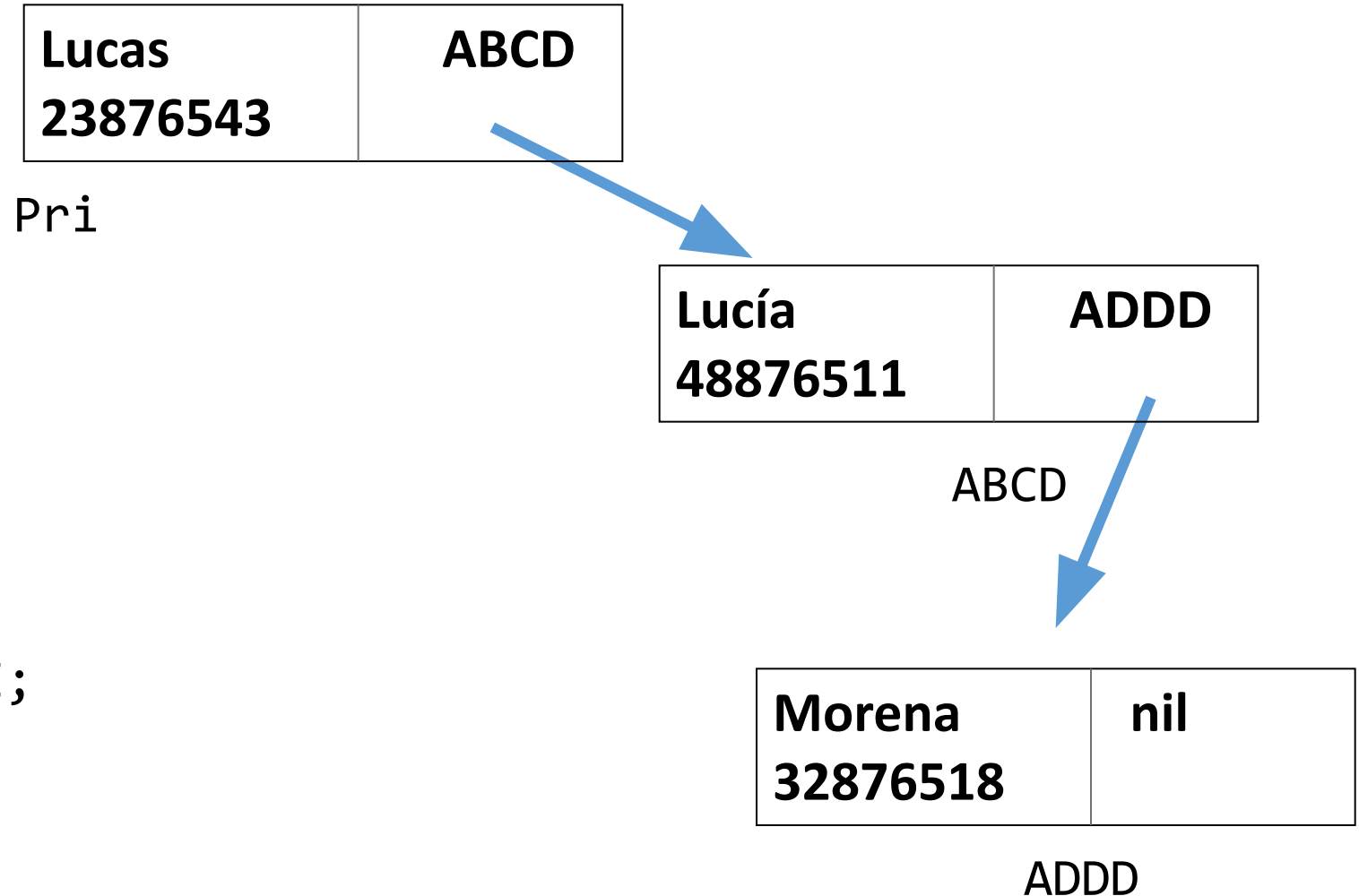
Type

```
persona = record
  nom:string;
  dni:integer;
end;
```

```
listaE= ^nodo;
nodo = record
  elemento: persona;
  punteroSig: listaE;
end;
```

Var

```
Pri: listaE;
```



# CADP – TIPOS DE DATOS - LISTA



Creación de una lista.

Agregar nodos al comienzo de la lista.

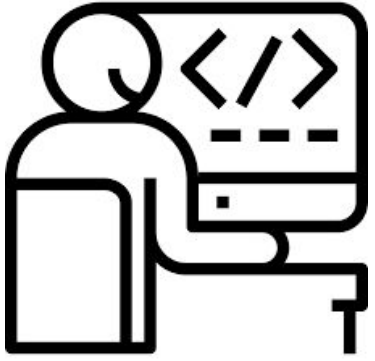
Recorrido de una lista.

Agregar nodos al final de la lista.

Insertar nodos en una lista ordenada

Eliminar nodos de una lista





# 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.

Insertar nodos en una lista ordenada

Eliminar nodos de una lista



# CADP – TEMAS



● Estructura de Datos - LISTA

● Operación de CREACION

● Operación de RECORRIDO



## CREAR UNA LISTA

Implica marcar que la lista no tiene una dirección inicial de comienzo.

```
Program uno;
```

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

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

```
Var
```

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

*Qué valor se le asigna a un puntero para indicar que no tiene una dirección asignada?*



# CADP – TIPOS DE DATOS - LISTA

CREAR



```
Program uno;
```

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

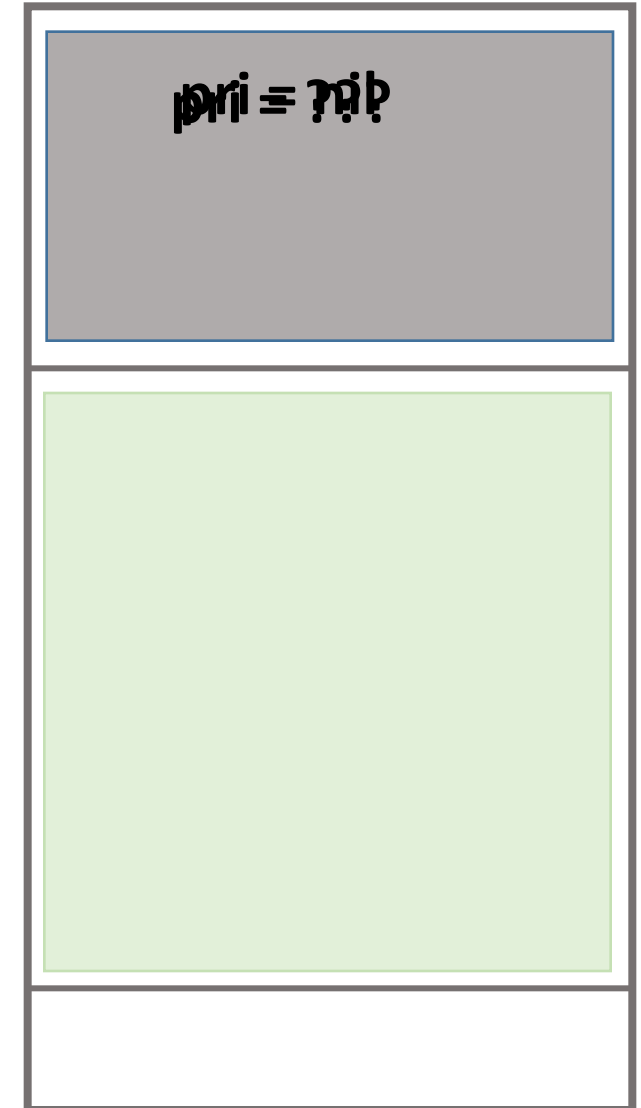
```
    datosEnteros= record  
        elem:integer;  
        sig:listaE;  
    end;
```

```
Var  
    pri: listaE;
```

```
Begin  
    pri:=nil;  
End.
```

Por qué no se hace  
new (pri)?

Se puede  
modularizar el  
crear?



# CADP – TIPOS DE DATOS - LISTA

CREAR



```
Program uno;
```

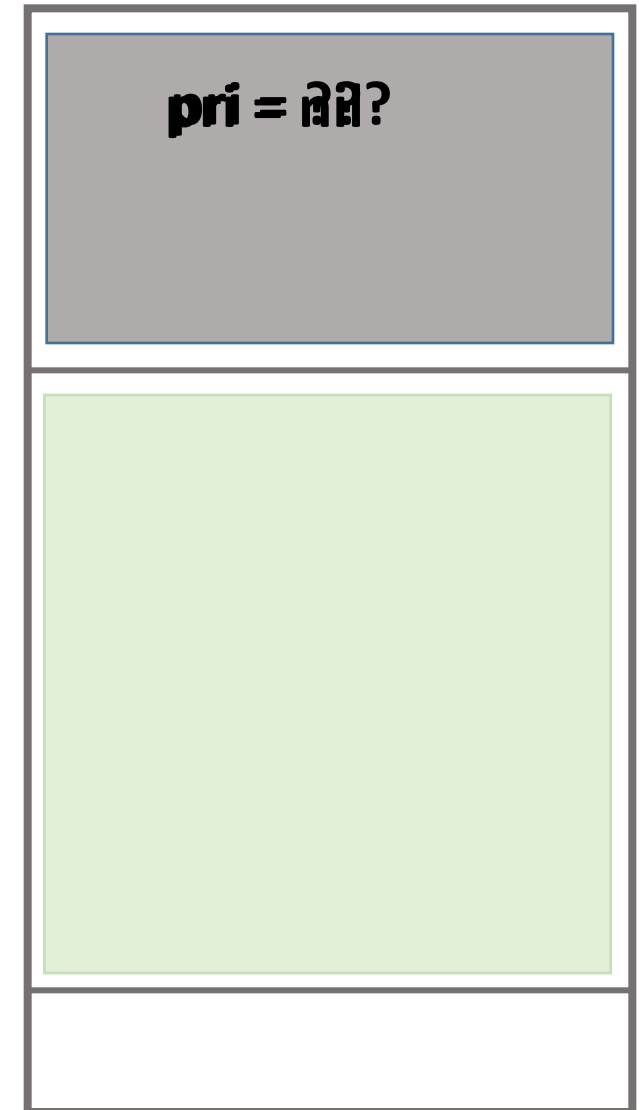
```
Type listaE= ^datosEnteros;
```

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

```
Procedure crear (var p: listaE);
begin
    p:= nil;
end;
```

```
Var
    pri: listaE;
```

```
Begin
    crear (pri);
End.
```





## RECORRER UNA LISTA

**Implica posicionarse al comienzo de la lista y a partir de allí ir “pasando” por cada elemento de la misma hasta llegar al final.**

```
Program uno;
```

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

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

```
Var  
    pri: listaE;
```

# CADP – TIPOS DE DATOS - LISTA

## RECORRER UNA LISTA



```
Program uno;
```

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

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

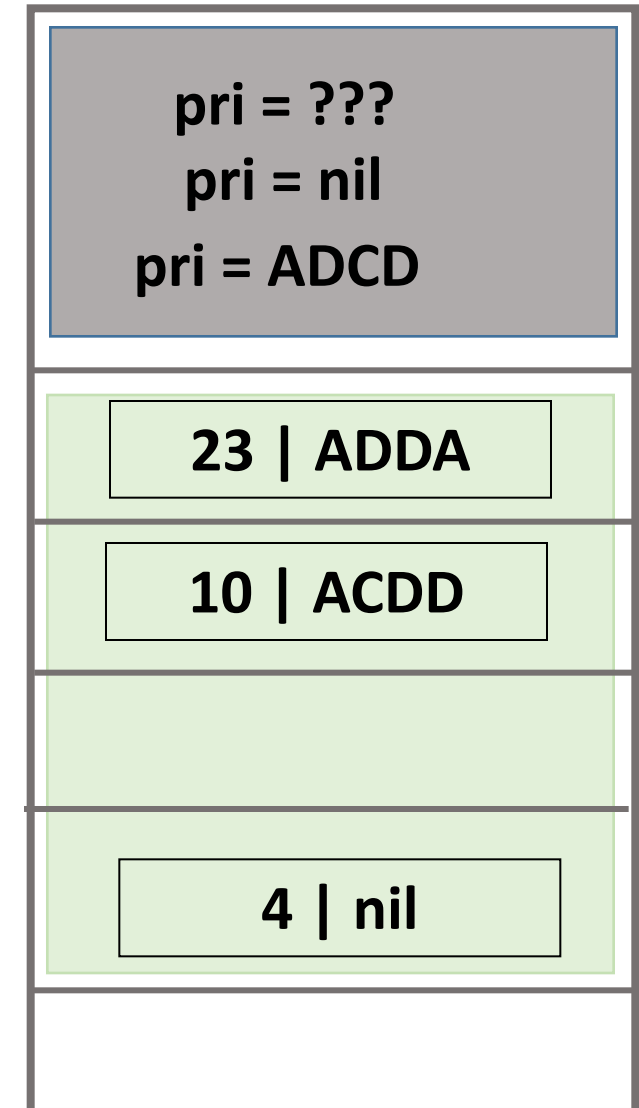
```
Var  
    pri: listaE;
```

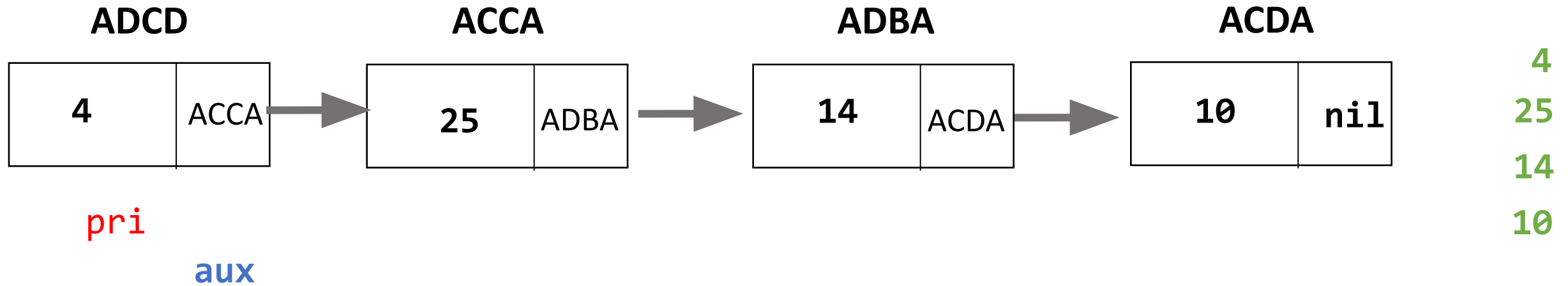
```
Begin  
    crear (pri);  
    cargarLista (pri); //Lo implementaremos más adelante  
    recorrerLista (pri);  
End.
```

**ACDD**

**ADCD**

**ADDA**



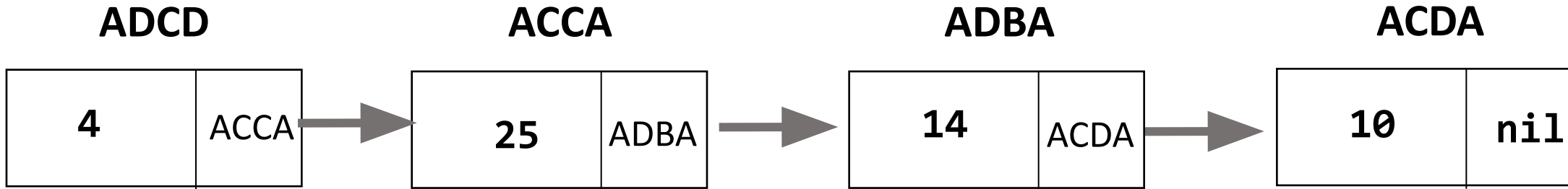


Inicializo una variable **auxiliar (aux)** con la dirección del puntero inicial de la lista

mientras (no sea el final de la lista)

    proceso el elemento (ej: imprimo, sumo, modifiko)

    avanzo al siguiente elemento de **auxiliar (aux)**



pI

```

procedure recorrerLista (pI: listaE);
Var
  aux:listaE;

begin
  aux:= pI;
  while (aux^.sig <> nil) do
    begin
      write (aux^.elem);
      aux:= aux^.sig;
    end;
end;
    
```

Es correcto?



Si la lista está **vacía**, (aux^.sig) da error.

Si la lista tiene **un solo elemento** (aux^.sig <> nil) da falso.

Si la lista tiene **muchos elementos** no imprime el último



```
procedure recorrerLista (pI: listaE);
```

```
Var
```

```
    aux:listaE;
```

Es necesaria  
la variable  
aux?

```
begin
```

```
    aux:= pI;
```

```
    while (aux <> nil) do
```

```
        begin
```

```
            write (aux^.elem);
```

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

```
        end;
```

```
end;
```

### ALTERNATIVA

```
procedure recorrerLista (pI: listaE);
```

```
begin
```

```
    while (pI <> nil) do
```

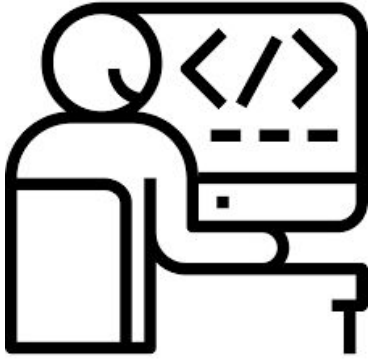
```
        begin
```

```
            write (pI^.elem);
```

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

```
        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.

Insertar nodos en una lista ordenada

Eliminar nodos de una lista



# CADP – TEMAS



- Operación de AGREGAR ADELANTE
- Operación de AGREGAR AL FINAL

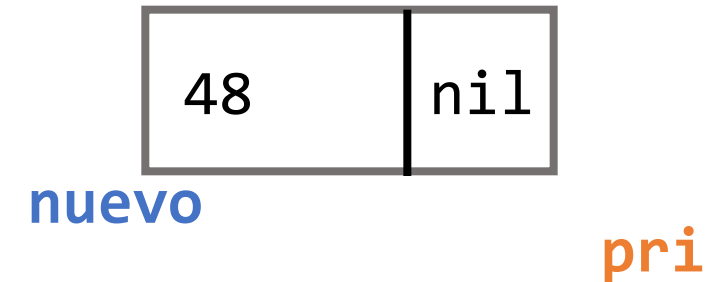
# CADP – TIPOS DE DATOS - LISTA



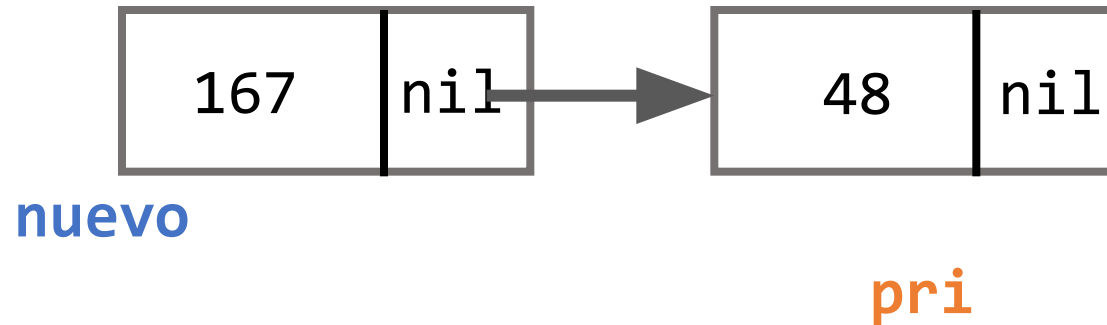
## AGREGAR ADELANTE

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

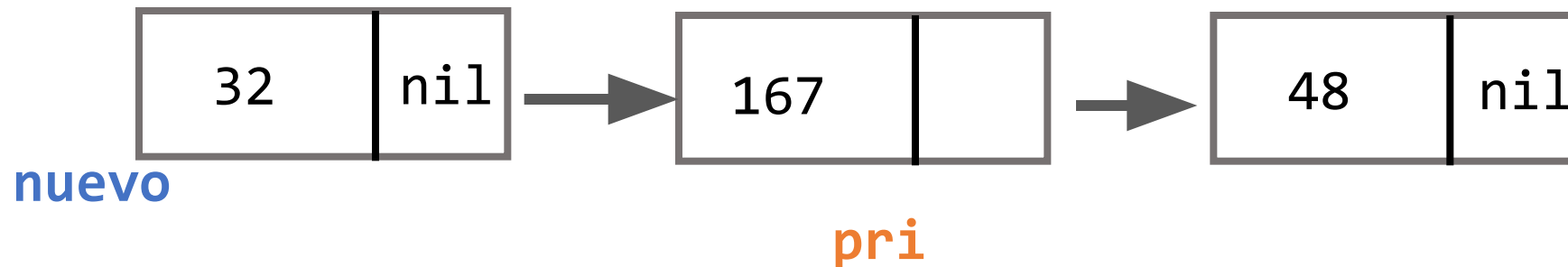
» `pri = nil`



» `pri <> nil`



*Cómo lo escribo?*





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**.

# CADP – TIPOS DE DATOS - LISTA

AGREGAR ADELANTE



```
Program uno;
```

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

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

```
Var
```

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

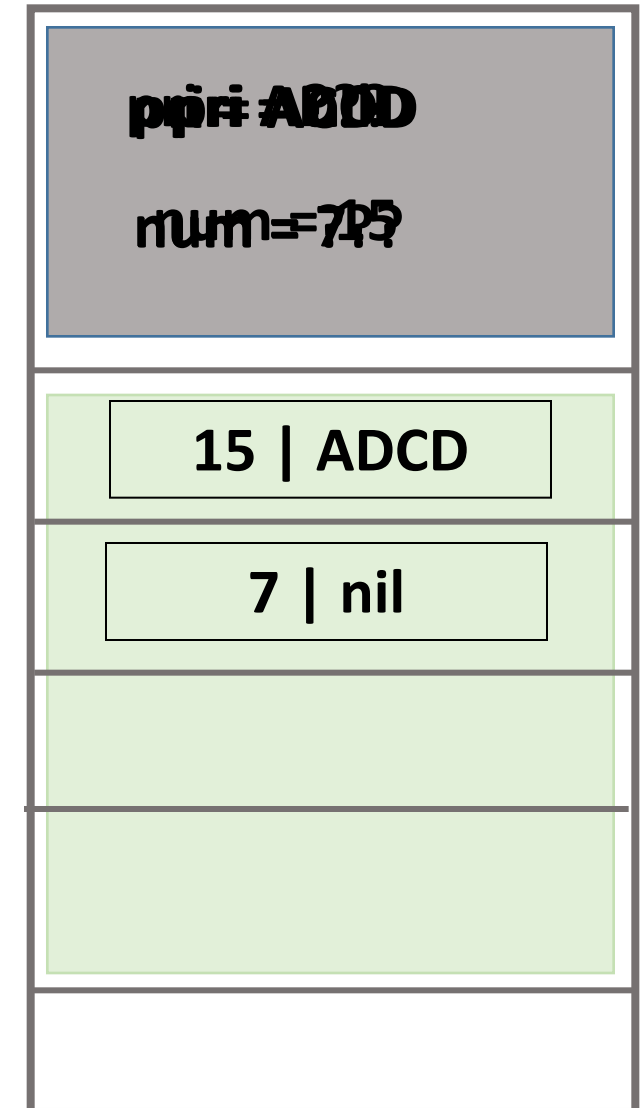
```
Begin
```

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

ACDD

ADCD

ADDA





```
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;
```

*Evalúo el caso y  
reasigno los  
punteros*

```
End;
```

# CADP – TIPOS DE DATOS - LISTA

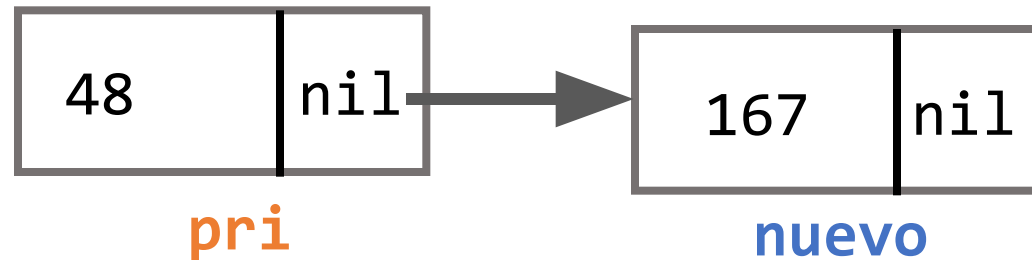
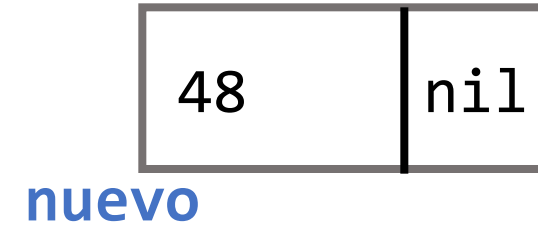


## AGREGAR AL FINAL

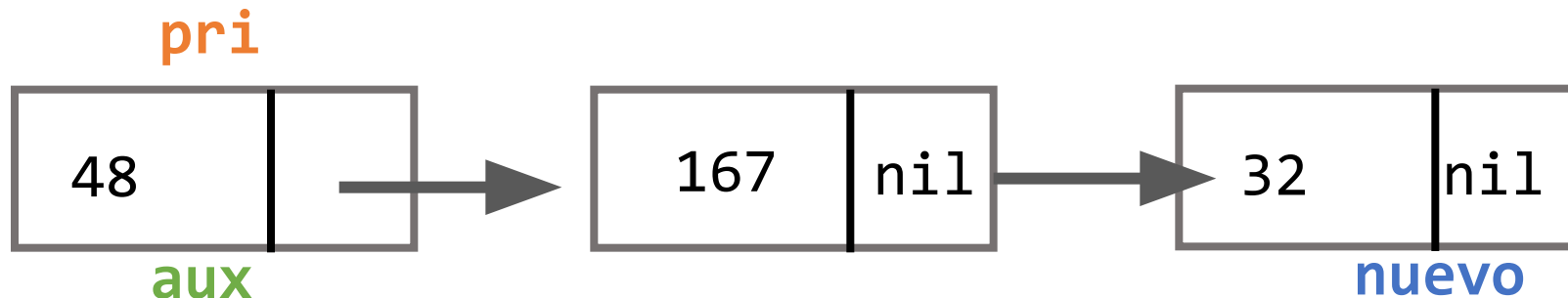
Implica generar un nuevo nodo y agregarlo como último elemento de la lista.

➤ `pri = nil`

➤ `pri <> nil`



*Cómo lo escribo?*





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**



# CADP – TIPOS DE DATOS - LISTA

AGREGAR AL FINAL



```
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);

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

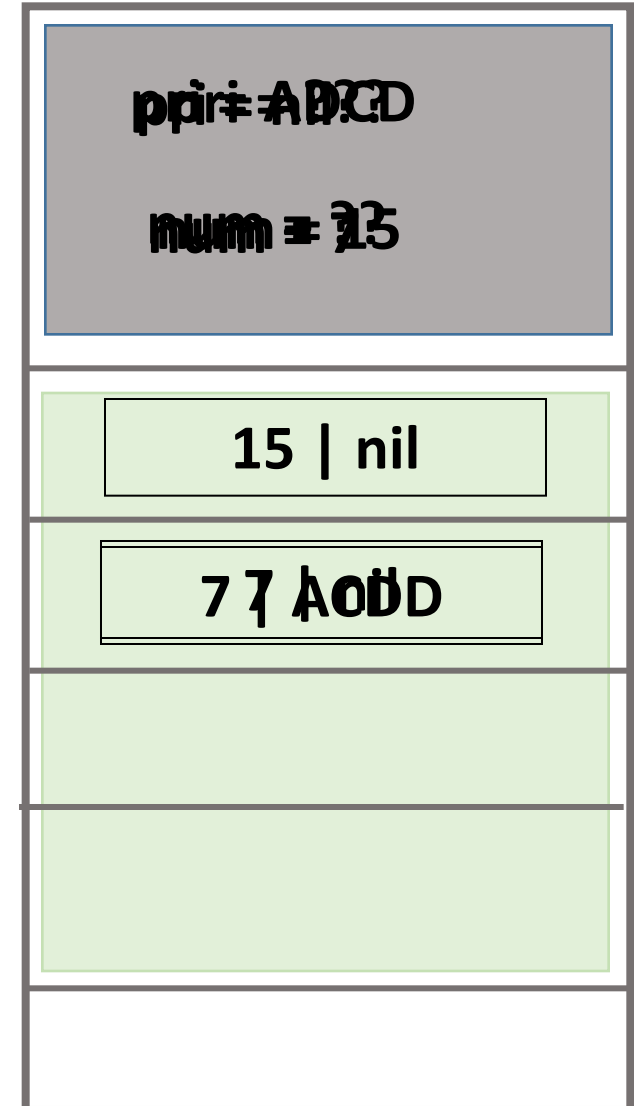
ACDD

15 | nil

ADCD

7 7 ADDD

ADDA



# CADP – TIPOS DE DATOS - 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
    else begin
```

```
        aux:= pI;
```

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

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

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

```
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?

← Evalúo si la lista está vacía

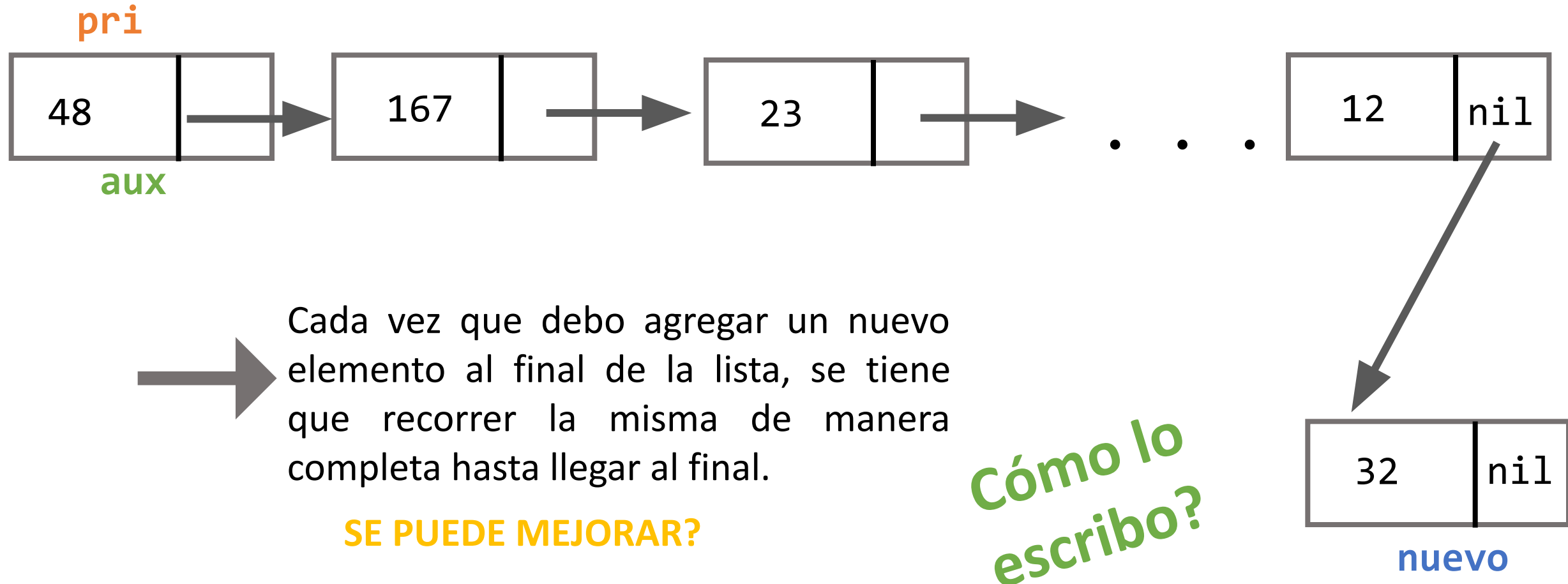
Recorro y quedo  
parado en el último  
elemento

← Le indico al último que ahora  
su siguiente es nuevo

# CADP – TIPOS DE DATOS - LISTA

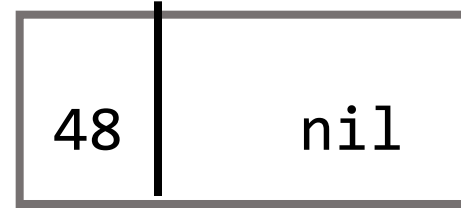


## AGREGAR AL FINAL EN UNA LISTA (OPCION 2)





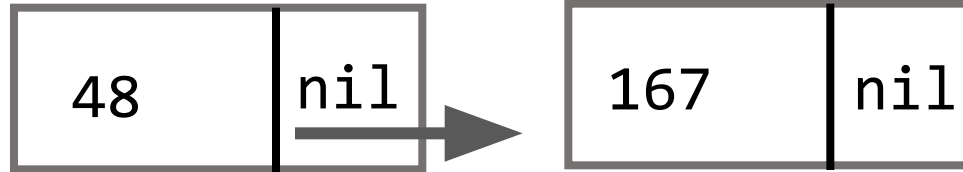
► `pri = nil`



nuevo

► `pri <> nil`

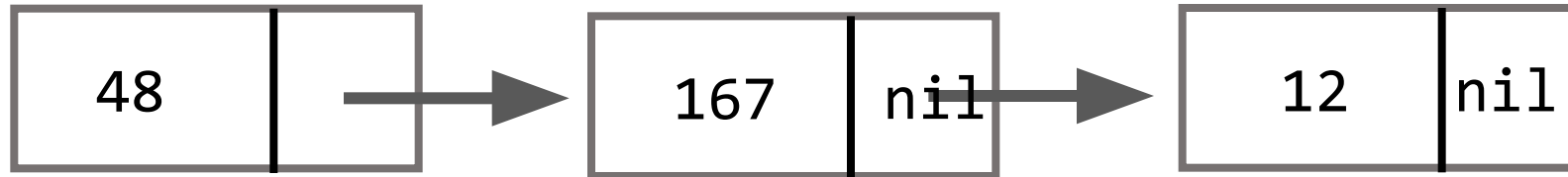
pri  
ult



nuevo

pri

ult



nuevo

*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 la dirección del **puntero final**

# CADP – TIPOS DE DATOS - LISTA

AGREGAR AL FINAL -2



```
Program uno;
```

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

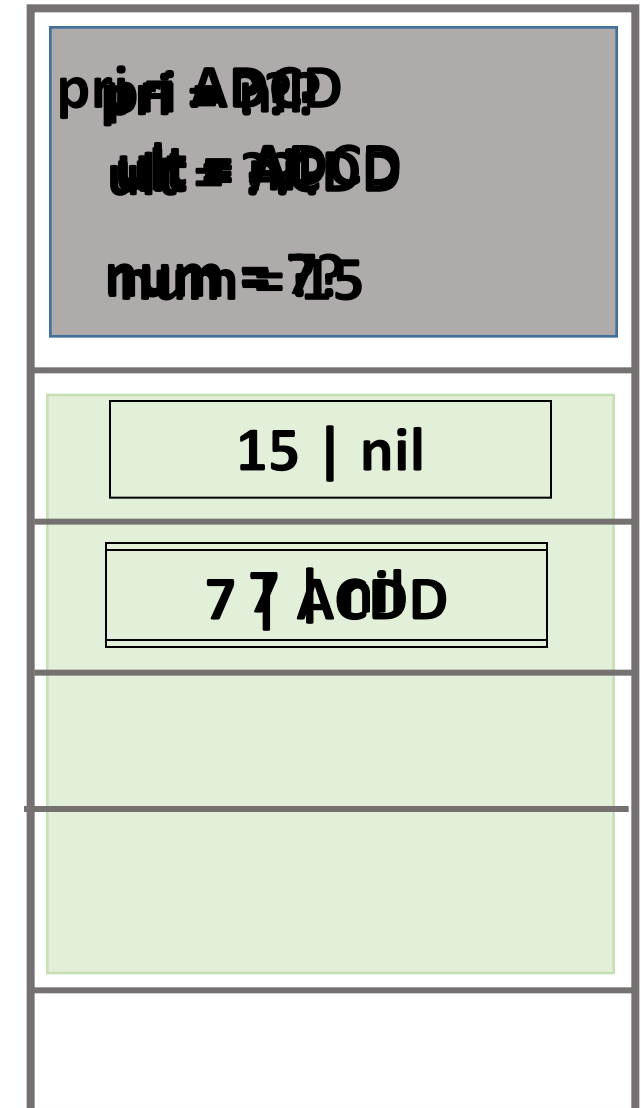
```
    datosEnteros= record
                        elem:integer;
                        sig:listaE;
                    end;
```

```
Var
```

```
    pri,ult: listaE;
    num:integer;
```

```
Begin
```

```
    crear (pri,ult);
    read (num);
    agregarAlFinal2 (pri,ult,num);
    read (num);
    agregarAlFinal2 (pri,,ult,num);
```





```
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;
```

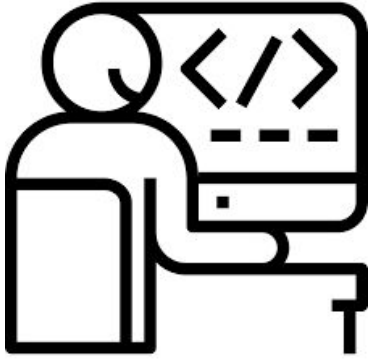
```
        pU:= nuevo;
```

```
    end;
```

```
End;
```

← Evalúo si la lista está vacía

← Actualizo el siguiente del último nodo y al último nodo



# 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

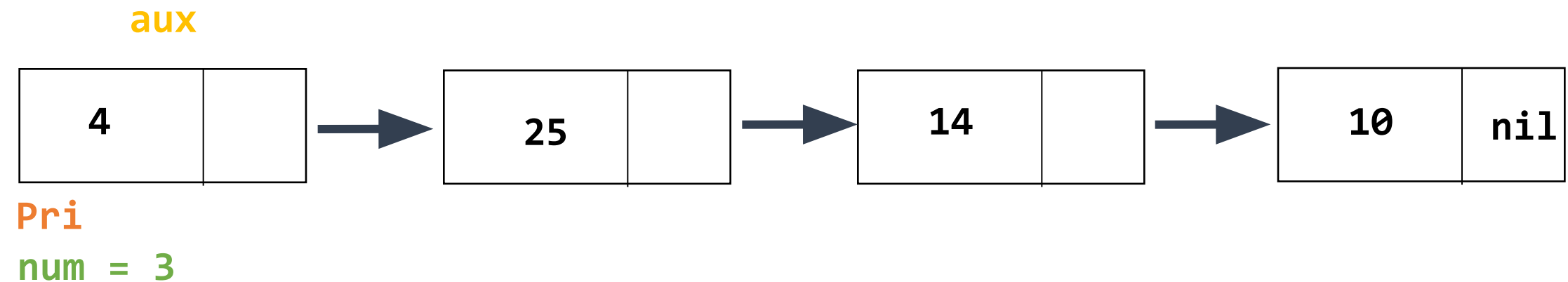
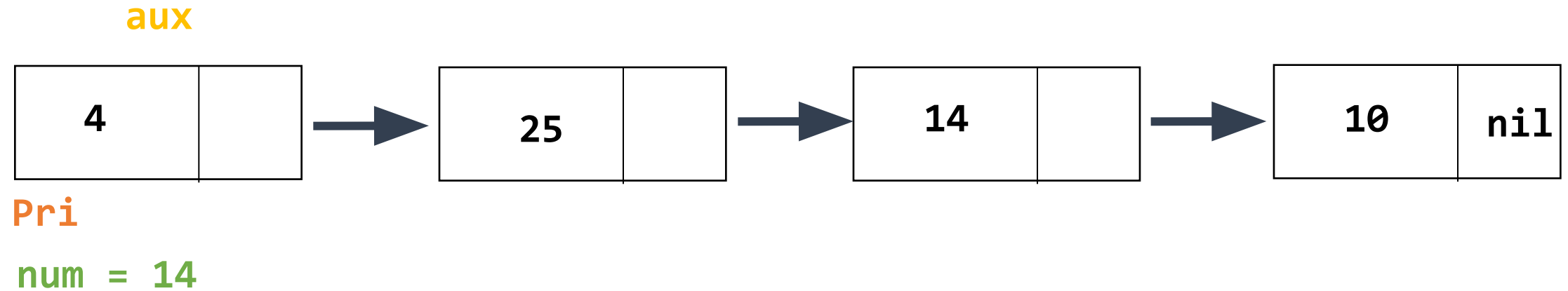
- Se debe recorrer toda la lista (en el peor de los casos), y detener la búsqueda en el momento que se encuentra el dato buscado o en el que la lista se terminó.

### 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 (si está ordenada de manera ascendente) al elemento actual.



## BUSQUEDA LISTA DESORDENADA





## 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;
```

```
Var  
    end;
```

```
    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;
```

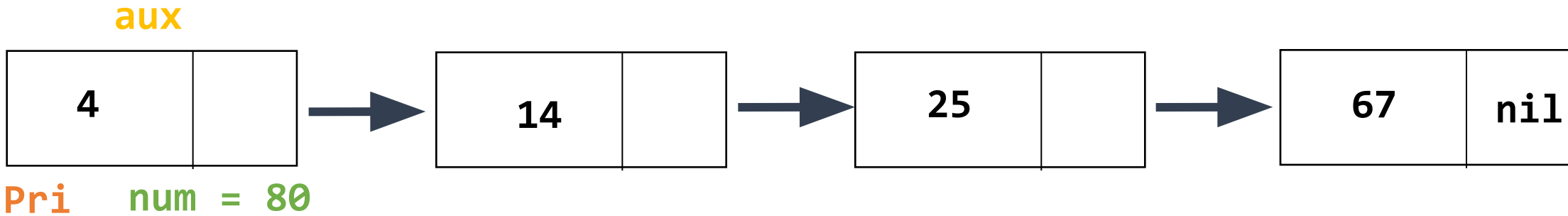
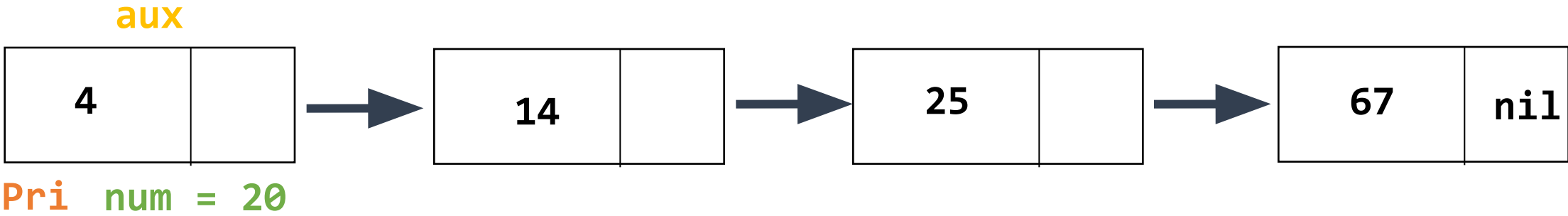
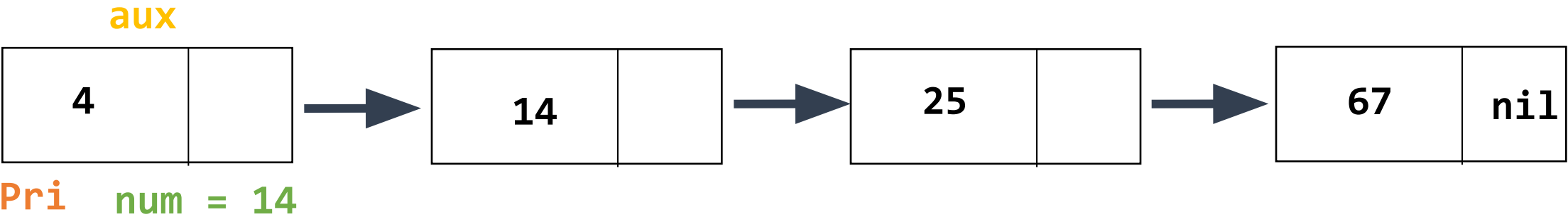
*Funciona si la  
lista que recibo  
es vacía?*

*Necesito  
usar aux?*

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

# CADP – TIPOS DE DATOS - LISTA

## 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 encuentre:= true;
```

```
    buscar:= encontré;
```

```
end;
```

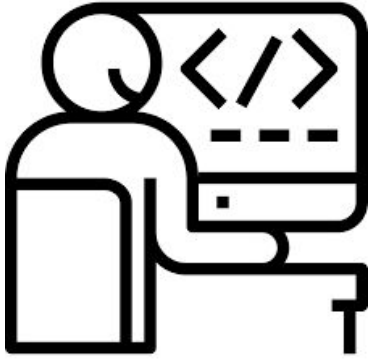
*Funciona si la lista que  
recibo es vacía?*

*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**



# 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.

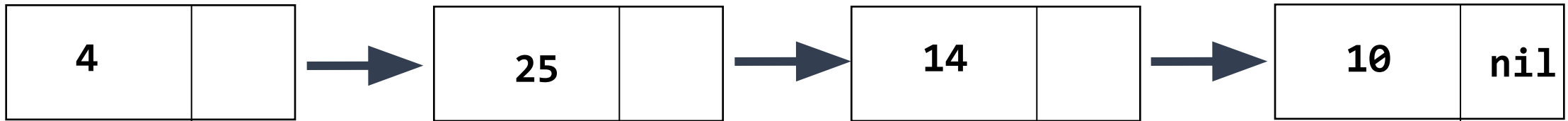
### Existen 3 casos:

- 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

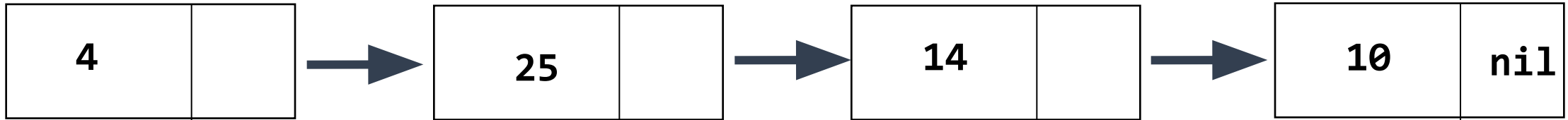
### Caso 1:

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)

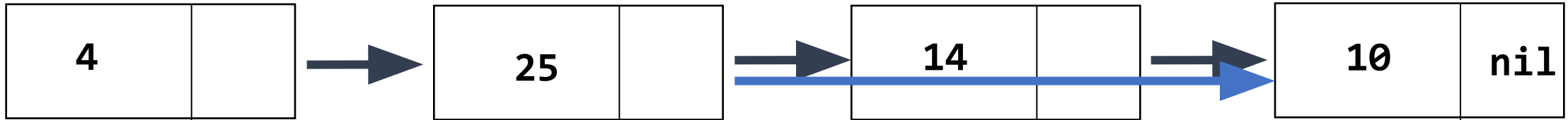
actualizo el puntero inicial de la lista

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

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



```
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;
```

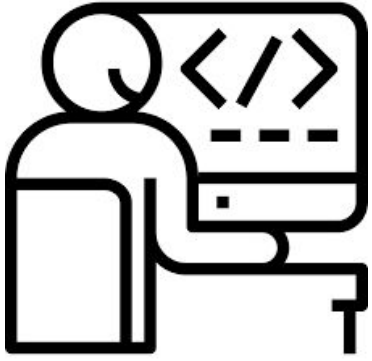
```
            dispose (actual);
```

```
            actual:= ant;
```

```
        end;
```

```
End;
```

**Qué modifico si la lista está  
ordenada y el elemento está  
una 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

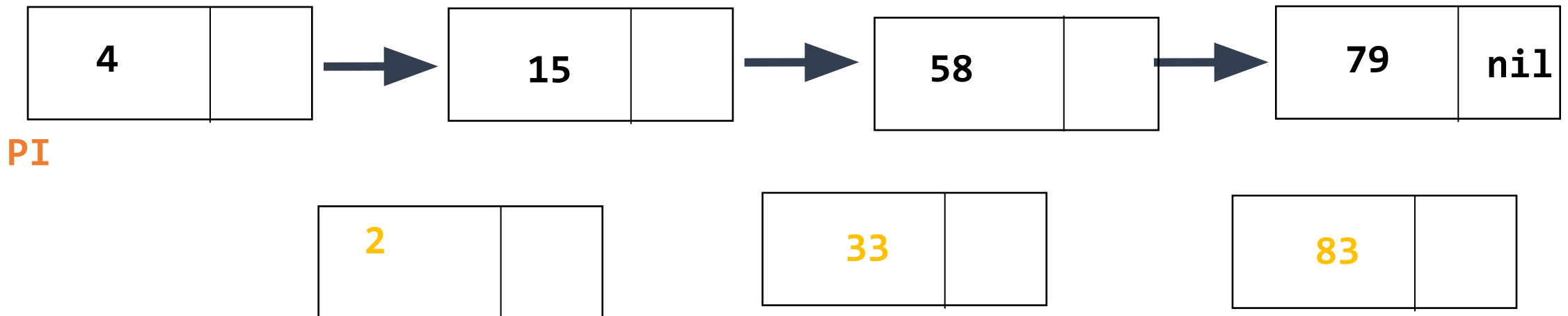




Implica agregar un nuevo nodo a una lista ordenada por algún criterio de manera que la lista siga quedando ordenada.

### Existen 4 casos:

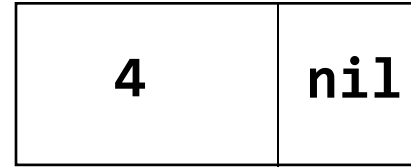
- que la lista esté vacía.
- que elemento vaya al comienzo de la lista (es menor al 1er nodo de la lista)
- que elemento vaya al “medio” de la lista (es menor al último nodo de la lista)
- que elemento vaya al final de la lista (es mayor al último nodo de la lista)





### CASO 1: lista vacía

PI = nil



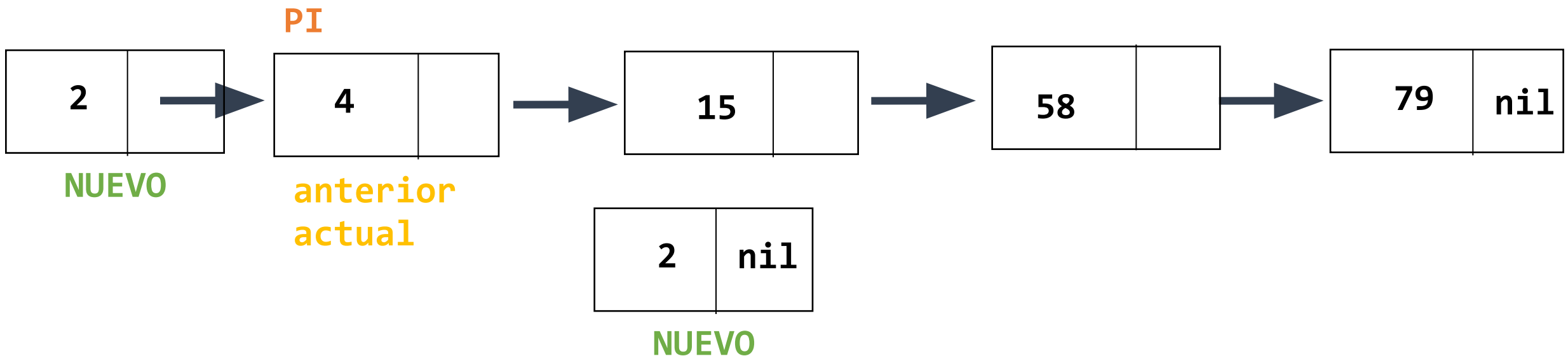
NUEVO

Generar un nuevo nodo (NUEVO).

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



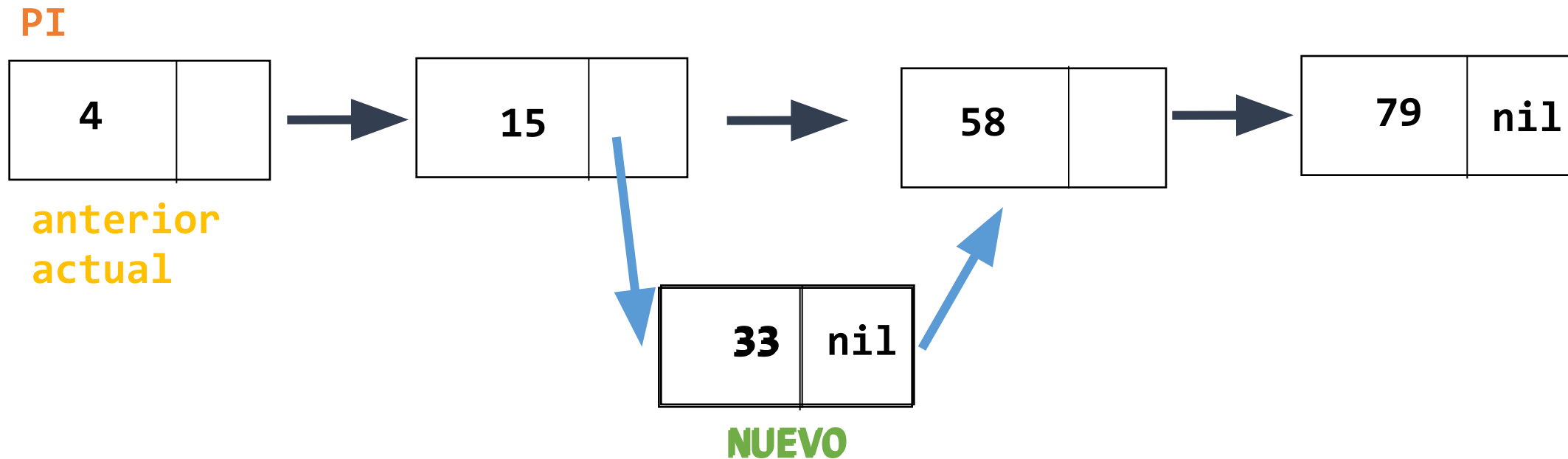
### CASO 2: lista no vacía, va al principio



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 = PI**



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



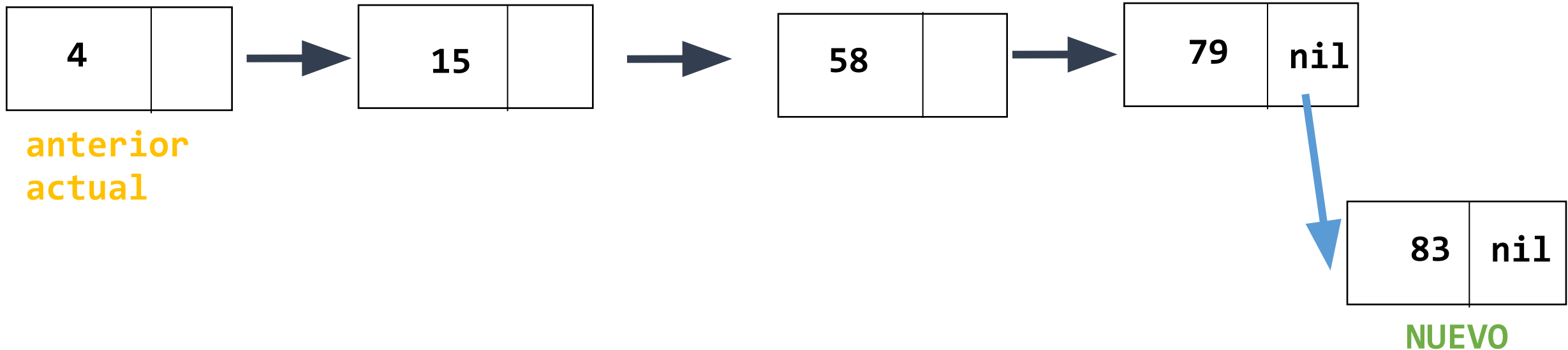
Generar un nuevo nodo (nuevo). Preparo los punteros para el recorrido  
 Recorro hasta encontrar la posición  
 Reasigno punteros, el siguiente de anterior es NUEVO y el siguiente de  
 NUEVO es actual.

**OBSERVAR QUE actual HABIA QUEDADO <> nil**



### CASO 4: lista no vacía, va al final

PI



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;
```



**BUSCO LA  
POSICION**







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



Caso 2  
pI=actual

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



Casos 3 y 4  
actual <> nil

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



Caso 4  
actual = nil

```
End;
```



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