Departamento de Informática
Universidad de Valladolid
Campus de Segovia

TEMA 3:
ESTRUCTURAS DINÁMICAS LINEALES.

LISTAS ENLAZADAS, PILAS Y COLAS

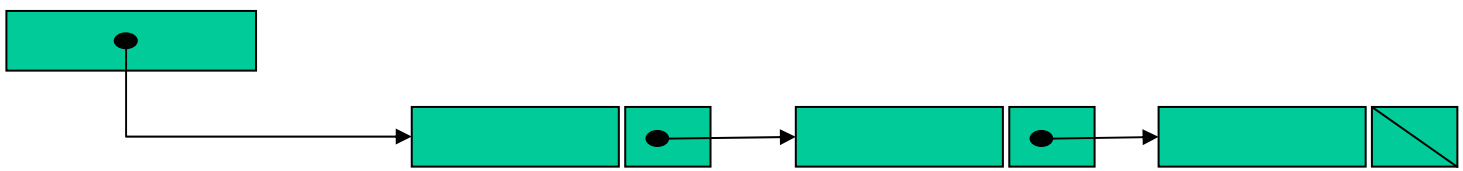
ÍNDICE

DEFINICIÓN DEL TIPO LISTA

- Una lista es una colección lineal de elementos, llamados nodos, dispuestos secuencialmente.
- A los nodos de esta estructura, salvo al primero, les corresponde un único predecesor.
- El número de nodos puede variar rápidamente en un proceso, aumentando por inserción de nodos o disminuyendo por supresión de nodos.
- Su implementación se realiza mediante punteros y variables dinámicas.

REPRESENTACIÓN MEDIANTE VARIABLES DINÁMICAS

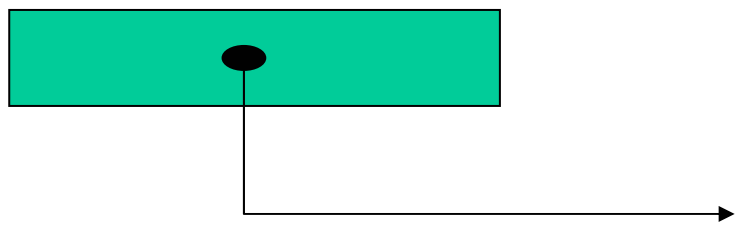
- Cada nodo de la lista es un registro con al menos dos componentes, la primera almacena el dato y la segunda es un puntero para poder señalar al siguiente nodo o con el valor “nil” si es el último.



IMPLEMENTACIÓN MEDIANTE REGISTROS Y PUNTEROS

```
TYPE
tElem=<lo que corresponda>
PtrNodo=^tNodo
tNodo=record
    info:tElem;
    sig:PtrNodo;
End; {tNodo}

VAR
lista:Ptrnodo;
```

Lista



OPERACIONES CON LISTAS ENLAZADAS

- Asociada a las listas enlazadas se pueden definir un conjunto de operaciones básicas que le dan identidad como tipo (TAD).

- Estas operaciones son básicas. Sin embargo la consideración final de lo que es básico o no depende del problema a resolver.

ALGUNAS DE ESTAS OPERACIONES

- **Listavacia(L):** inicializa una lista
- **Esvacia(L):** Función que determina si es vacia o no.
- **Inserprim(x,L):** Inserta un nodo con la información x como primer nodo de la lista.
- **Inserta(x,P,L):** Inserta en la lista L un nodo con el campo x, delante del nodo de dirección P.
- **Inserfin(x,L):** Inserta un nodo con el campo x como último nodo de la lista L.
- **Localiza(x,L):** Función que devuelve la posición (dirección) donde está el campo x. Sino se encuentra devuelve nil.
- **Suprime(x,L):** Elimina el nodo de la lista que contiene x
- **Suprimedir(P,L):** Elimina de la lista el nodo cuya dirección viene dada por P.

ALGUNAS DE ESTAS OPERACIONES

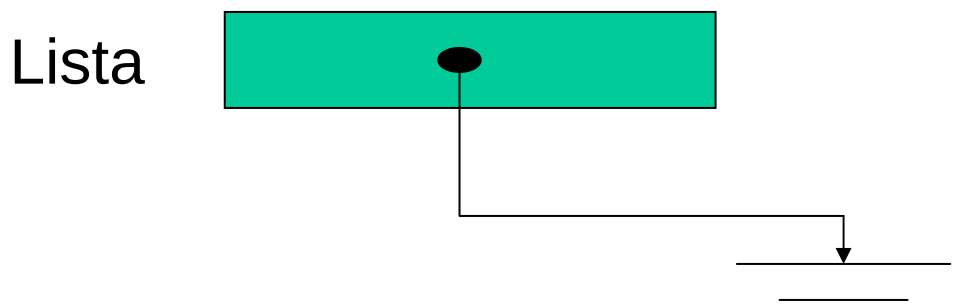
- **Siguiente(P,L):** Función que devuelve la dirección del nodo siguiente a P.
- **Anterior(P,L):** Función que devuelve la dirección del nodo anterior a P.
- **Primero(L):** Función que devuelve la dirección del primer nodo de la lista L.
- **Último(L):** Función que devuelve la dirección del último nodo de la lista L.
- **Anula(L):** Esta operación vacia la lista L.
- **Visualiza(L):** Visualiza el campo de información de todos los elementos de la lista.

INICIAR UNA LISTA ENLAZADA

- Las operaciones básicas de inicialización son:
  - **Listavacia(L):** Iniciar una lista sin nodos como lista vacía
  - **Esvacia(L):** Determinar si una lista dada está vacía

LISTAVACIA

```
PROCEDURE Listavacia( var L:PtrNodo);
    BEGIN
        L:=nil
    END; {Listavacia}
```



ESVACIA

```
FUNCTION Esvacia(L:PtrNodo):boolean;
    BEGIN
        Esvacia:=(L=nil)
    END; {Esvacia}
```

BÚSQUEDA EN UNA LISTA

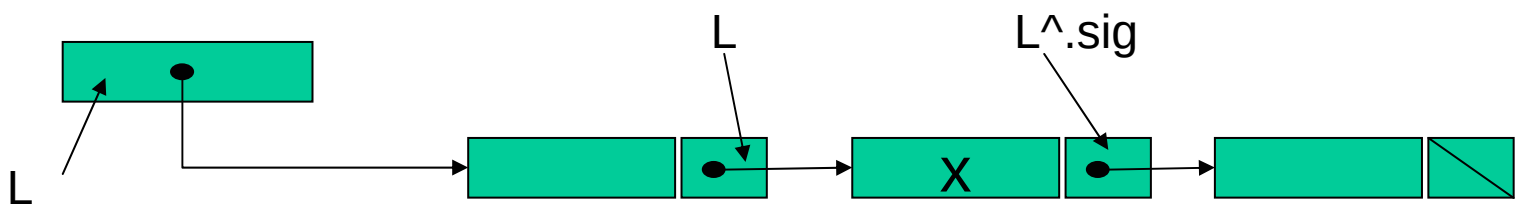
- La búsqueda en una lista requiere de las siguiente operaciones:

  - **Localiza(x,L):** que permite localizar la posición (dirección) de un determinado nodo que contenga un determinado campo de información.

  - **Existe(x,L):** que determina si un nodo con una determinada información existe.

FUNCIÓN LOCALIZA

```
FUNCTION Localiza (x:tElem;L:PtrNodo):PtrNodo;
{Dev. la dirección de un nodo que contiene una determinada información
    y sino devuelve el valor nil.}
BEGIN
    WHILE (L^.sig<>nil) AND (L^.info<>x) DO
        L:=L^.sig; {Avanza a través de los nodos de la lista}
    IF L^.info<>x THEN
        Localiza:=nil
    ELSE
        Localiza:=L
END; {Localiza}
```



FUNCIÓN EXISTE

```
FUNCTION Existe (x:tElem;L:PtrNodo):Boolean;
{Dev.  True si el nodo está presente y false sino lo está}

BEGIN
    IF not Esvacia(L) THEN BEGIN
        WHILE (L^.sig<>nil) AND (L^.info<>x) DO
          L:=L^.sig; {Avanza a través de los nodos de la lista}
          Existe:=(L^.info=x)
        END; {IF}
    ELSE
         Existe:=false
    END; {Localiza}
```

OPERACIONES DE DIRECCIÓN

- Estas operaciones permiten obtener la dirección del nodo anterior o siguiente a uno dado o el último de una determinada lista.

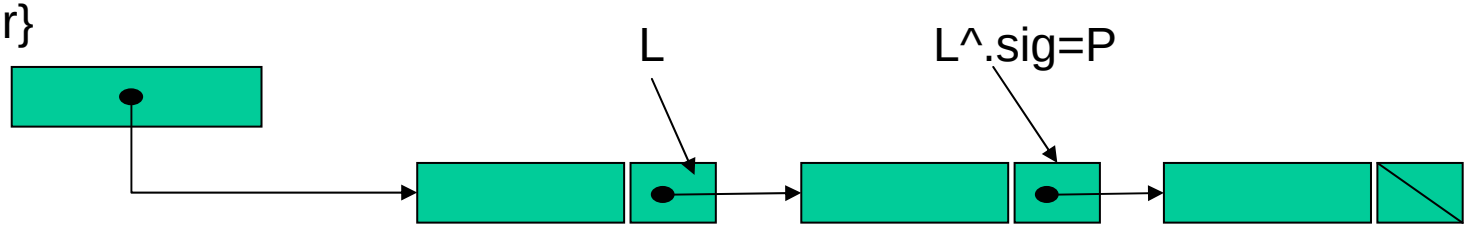
  - **Anterior(P,L):** permite localizar la posición (dirección) del nodo anterior a uno dado.

  - **Siguiente(P,L):** permite localizar la posición (dirección) del nodo siguiente a uno dado.

  - **Último(L):** permite localizar la posición (dirección) del último nodo de una lista dada.

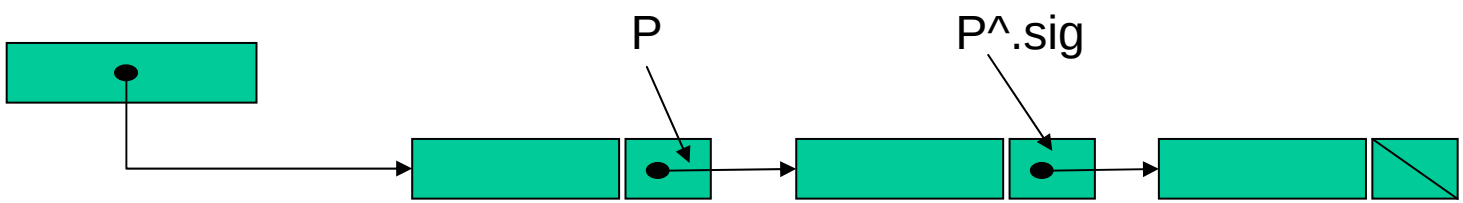
FUNCIÓN ANTERIOR

```
FUNCTION Anterior (P,L:PtrNodo):PtrNodo;
{Dev. la dirección anterior a un nodo dado si existe y nil si este no existe  o si es
    una lista vacia o si L=P}
BEGIN
    IF Esvacia(L)  OR (L=P) THEN
          Anterior:=nil
    ELSE BEGIN
      WHILE (L^.sig<>P) AND (L^.sig<>nil) DO
          L:=L^.sig; {Avanza a través de los nodos de la lista}
       IF L^.sig=P THEN
          Anterior:=L
       ELSE
          Anterior:=nil
    END {ELSE}
END; {Anterior}
```



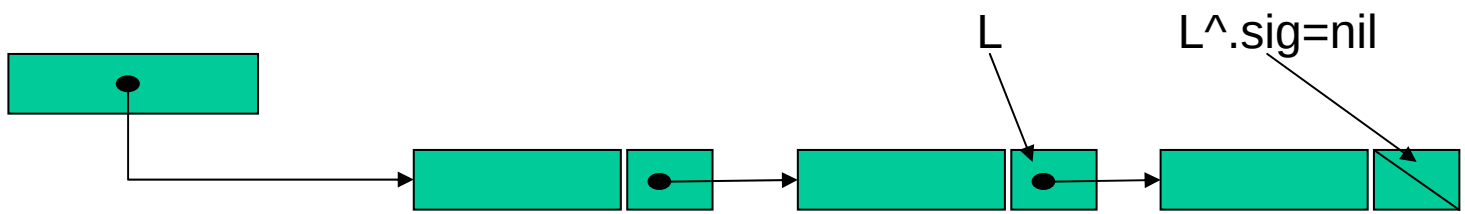
FUNCIÓN SIGUIENTE

```
FUNCTION Siguiente (P,L:PtrNodo):PtrNodo;
{Dev. la dirección siguiente a un nodo dado si existe y nil si este no existe
    o si es una lista vacia o si L=P}
BEGIN
    IF Esvacia(L)  OR (P=nil) THEN
          Siguiente:=nil
    ELSE
          Siguiente:=P^.sig
END; {Siguiente}
```



FUNCIÓN ÚLTIMO

```
FUNCTION Ultimo (L:PtrNodo):PtrNodo;
{Dev. la dirección siguiente a un nodo dado si existe y nil si este no existe
    o si es una lista vacia o si L=P}
BEGIN
    IF Esvacia(L)  THEN
          Ultimo:=nil
    ELSE  BEGIN
       WHILE  (L^.sig<>nil) DO
          L:=L^.sig; {Avanza a través de los nodos de la lista}
          Ultimo:=L
      END {ELSE}
END; {Ultimo}
```



INSERCIÓN DE UN ELEMENTO EN UNA LISTA

- Esta operación supone la creación de un nuevo nodo más los movimientos necesarios para situar el nodo en su lugar.

  - **Crear(x):** Crea un nodo que contiene como campo información x.

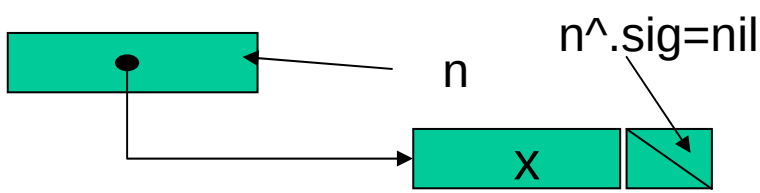
  - **Inserprim(x,L):** Inserta el nodo creado con la información x en el primer lugar de una lista L dada.

  - **Inserta(x,P,L):** Inserta un nodo creado con la información x delante del nodo de dirección P de una lista L dada.

  - **Inserfin(x,L):** Inserta un nodo creado con la información x al final de una lista L dada.

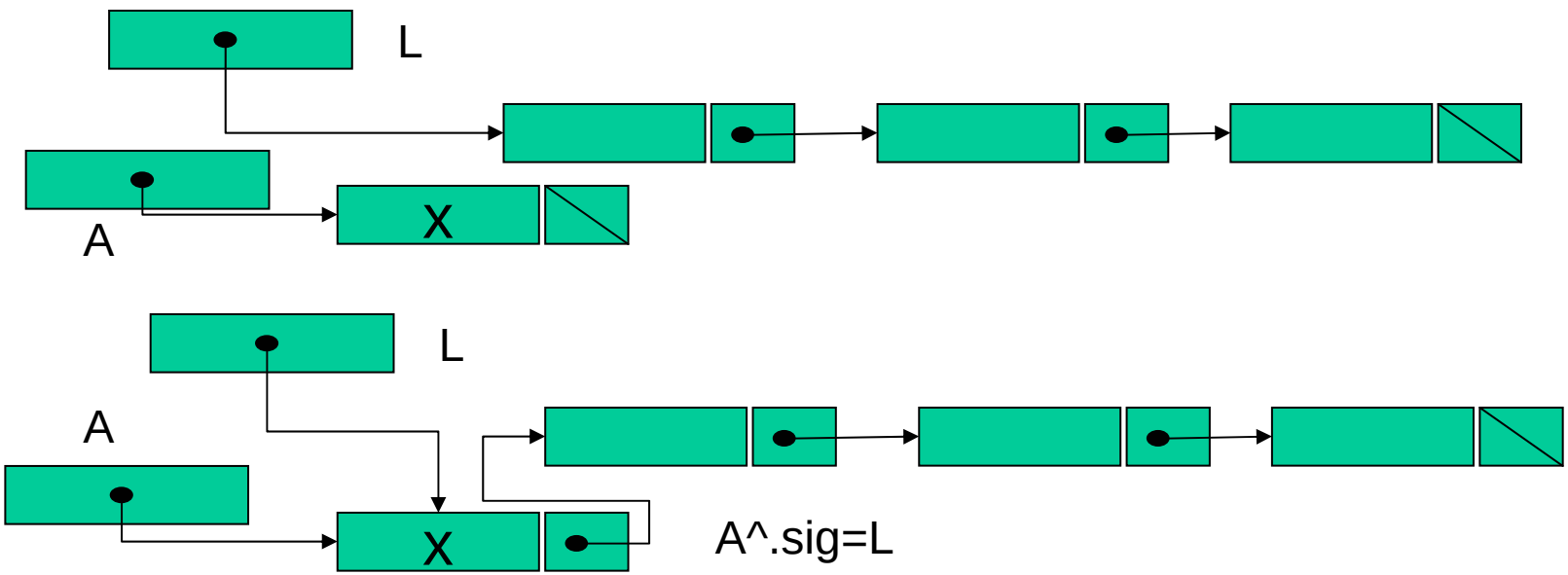
FUNCIÓN CREAR

```
FUNCTION Crear (x:tElem):PtrNodo;
{Dev. la dirección de un nodo creado con el valor x en su campo de
    contenido}
VAR
    n:PtrNodo;
BEGIN
    New(n);
    n^.info:=x;
    n^.sig:=nil;
    Crear:=n
END; {Crear}
```



PROCEDIMIENTO INSERPRIM

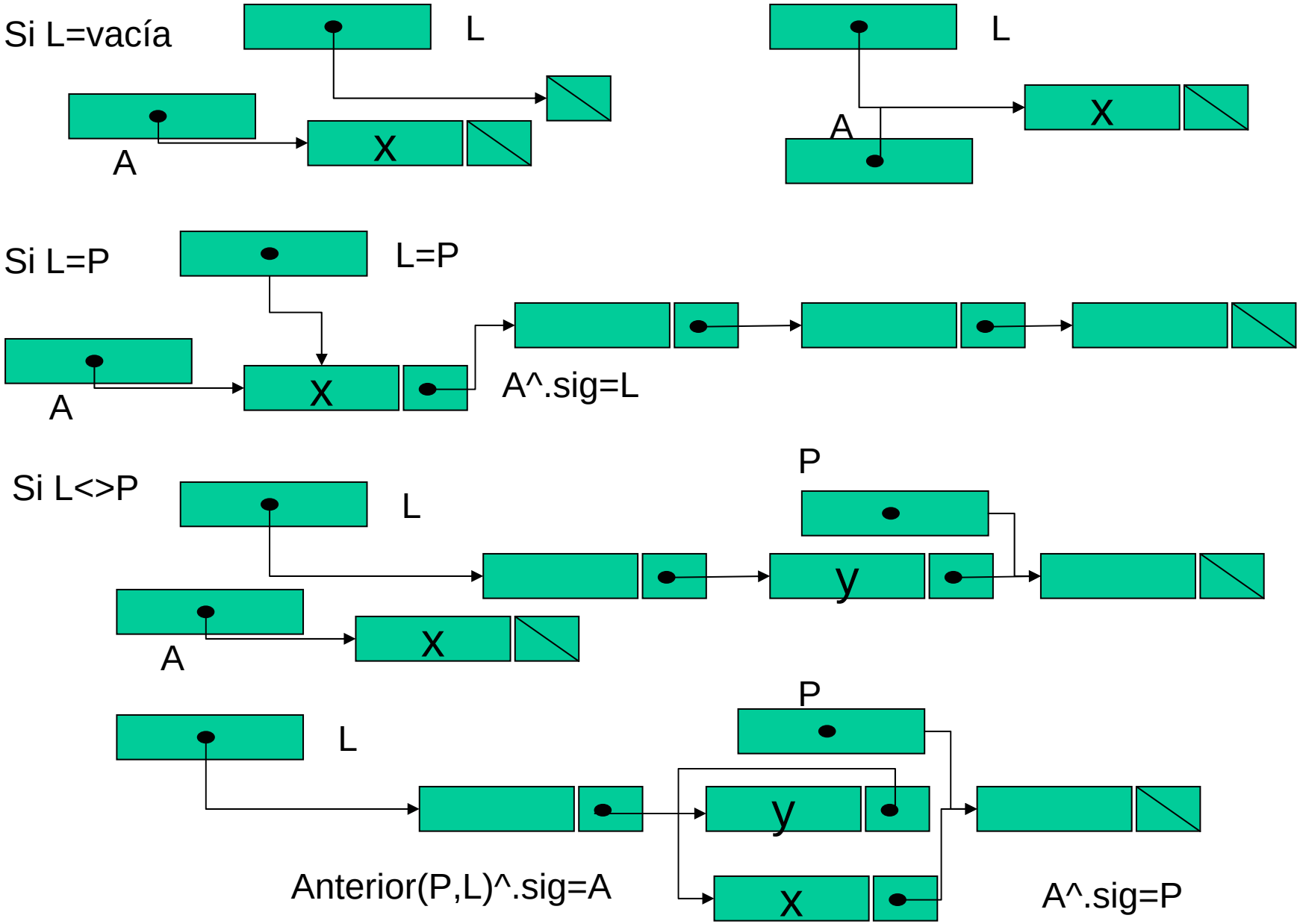
```
PROCEDURE Inserprim (x:tElem; var L:PtrNodo);
{Efecto. Inserta el nodo creado en el primer lugar de la lista}
VAR
    A:PtrNodo;
BEGIN
    A:=Crear(x);
    A^.sig:=L;
    L:=A
END; {Inserprim}
```



PROCEDIMIENTO INSERTA

```
PROCEDURE Inserta (x:tElem; P:PtrNodo; var L:PtrNodo);
{Efecto. Inserta el nodo creado en una lista dada delante del nodo de dirección P}
VAR
    A:PtrNodo;
BEGIN
    A:=Crear(x);
    IF  Esvacia(L) THEN
        L:=A
        ELSE IF P=L THEN BEGIN
           A^.sig:=P;
           L:=A
        END {Else if}
           ELSE BEGIN
               Anterior(P,L)^.sig:=A;
               A^.sig:=P
           END {Else}
END; {Inserta}
```

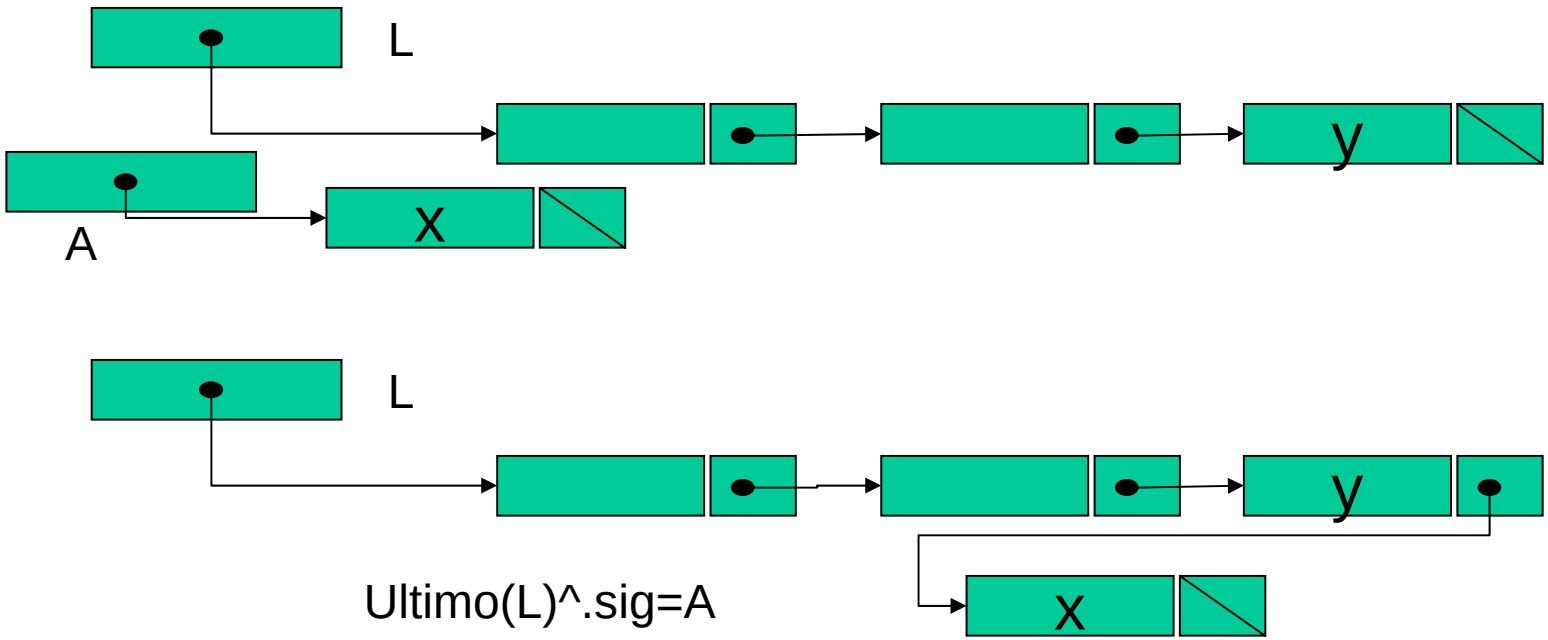
PROCEDIMIENTO INSERTA



PROCEDIMIENTO INSERFIN

```
PROCEDURE Inserfin (x:tElem;  var L:PtrNodo);
{Efecto. Inserta el nodo creado al final de una lista dada}
VAR
    A:PtrNodo;
BEGIN
    A:=Crear(x);
    IF  Esvacia(L) THEN
        L:=A
       ELSE
         Ultimo(L)^.sig:=A;
END; {Inserfin}
```

PROCEDIMIENTO INSERFIN



SUPRESIÓN DE UN ELEMENTO DE UNA LISTA

- Este tipo de operaciones supone enlazar el nodo anterior con el nodo siguiente al que va a ser suprimido y liberar la memoria ocupada.

  - **Suprime(x,L):** Elimina el nodo que contiene como campo de información el valor x.

  - **Suprimedir(P,L):** Elimina el nodo situado en la dirección P de una lista L dada.

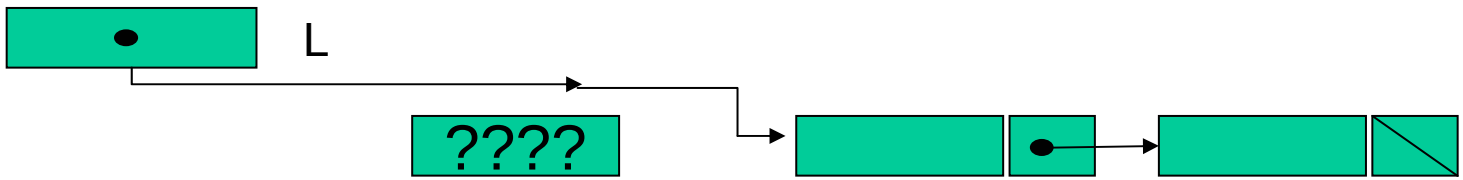
  - **Anula(L):** Esta operación libera toda la memoria ocupada por los nodos de la lista L.

PROCEDIMIENTO SUPRIME

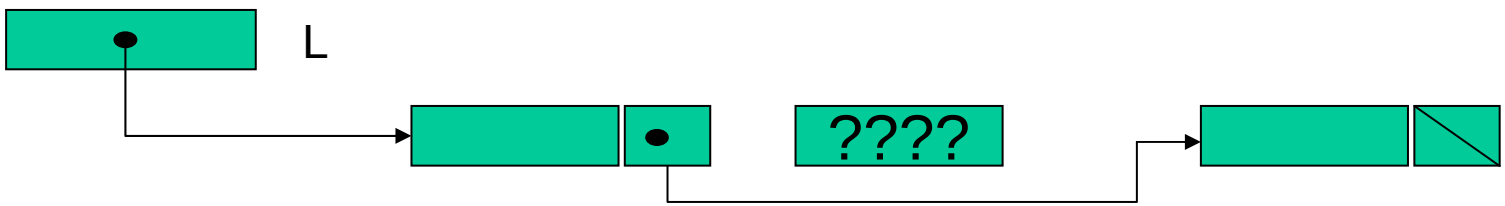
```
PROCEDURE Suprime (x:tElem;  var L:PtrNodo);
{Efecto. Elimina el nodo con campo  x de una lista dada}
VAR
    A:PtrNodo;
BEGIN
    A:=Localiza(x,L);
    IF  A<>nil THEN BEGIN
            IF A=L THEN {Primer nodo}
                    L:=L^.sig
             ELSE
                    Anterior(A,L)^.sig:=A^.sig;
            Dispose(A)
    END {If}
END; {Suprime}
```

PROCEDIMIENTO SUPRIME

Primer nodo de L



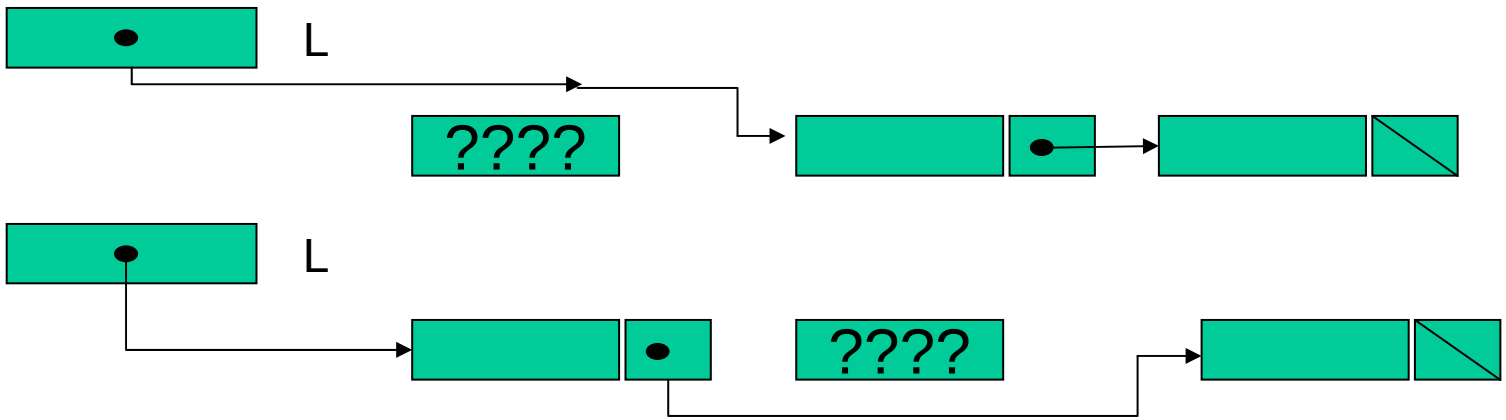
Cualquier otro nodo de L



PROCEDIMIENTO SUPRIMEDIR

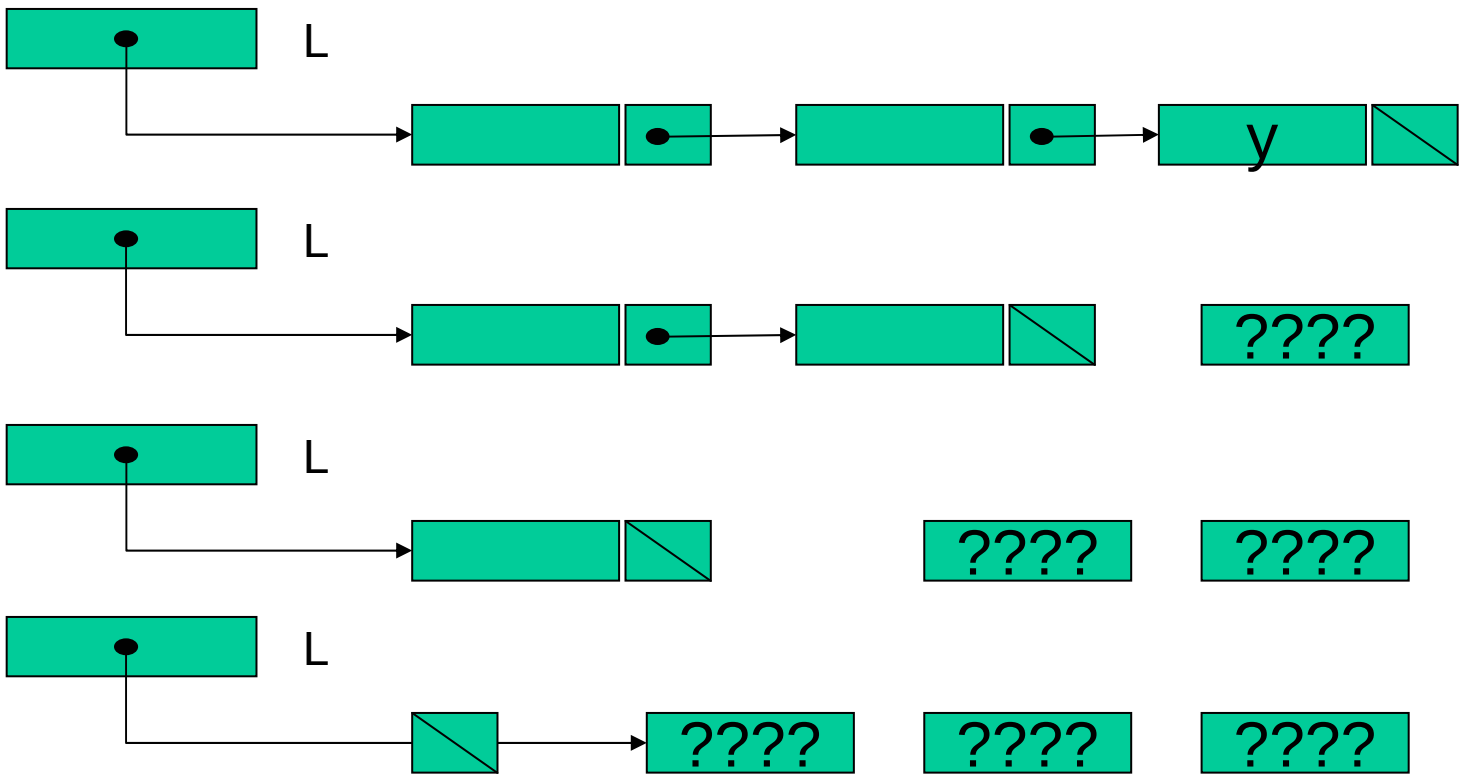
```
PROCEDURE Suprimedir(P:PtrNodo;  var L:PtrNodo);
{Efecto. Elimina el nodo con dirección P de una lista dada}
BEGIN
    IF P=L THEN BEGIN{Primer nodo}
            L:=L^.sig;
            Dispose(P)
    END {If}
            ELSE
                IF  Anterior(P,L)<>nil THEN BEGIN
                      Anterior(P,L)^.sig:=Siguiente(P,L);
                      Dispose(P)
                END {Else if}
END; {Suprimedir}
```

PROCEDIMIENTO SUPRIMEDIR



PROCEDIMIENTO ANULA

```
PROCEDURE Anula (var L:PtrNodo);
{Efecto. Libera toda la memoria ocupada por la lista L}
BEGIN
    WHILE not Esvacia(L)  DO
          Suprimedir(ultimo(L),L)
END; {Anula}
```



RECORRIDO DE UNA LISTA

- Esta tipo de operaciones permite visualizar el contenido de los nodos de una lista.
  - **Visualiza(L):** Visualiza el contenido de todos los nodos de una lista dada.

PROCEDIMIENTO VISUALIZA

```
PROCEDURE Visualiza ( L:PtrNodo);
{Efecto. Visualiza el contenido de toda la lista}
BEGIN
    WHILE L<>nil DO BEGIN
            write(L^.info,' ');
            L:=L^.sig
    END {While}
END; {Visualiza}
```

LISTAS ORDENADAS

- Si el tipo de información almacenada en una lista es un tipo ordinal o tiene un subcampo ordinal se puede mantener la lista ordenada respecto de dicho campo.

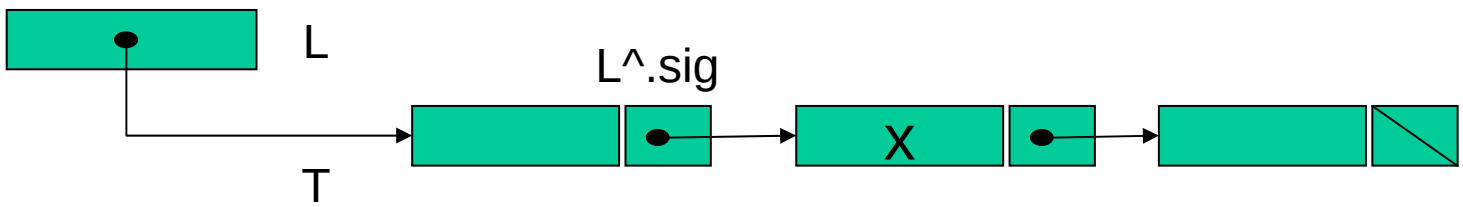
- Las operaciones para este tipo de lista son:

  - **Posinser(x,L):** Devuelve la dirección del nodo anterior al que contiene el campo x según la ordenación dada y nil si es el anterior al primero.
  - **Inserorden(x,L):** Si la lista está vacía el nodo se inserta como el primero de la lista, sino se inserta en la posición que le corresponde.

FUNCTION POSINSER

```
FUNCTION Posinser (x:tElem;   L:PtrNodo):PtrNodo;
{Dev. La dirección del nodo anterior a x, y nil si es el anterior al primero o es vacia}
VAR
    T:PtrNodo;
BEGIN
    T:=nil;
    IF  Not Esvacia(L) THEN BEGIN
          WHILE (x≥L^.info) AND (L^.sig<>nil) DO BEGIN
                T:=L;
                 L:=L^.sig
           END; {While}
           IF x≥L^.info THEN
                  T:=L
    END; {If}
    Posinser:=T
END; {Posinser}
```

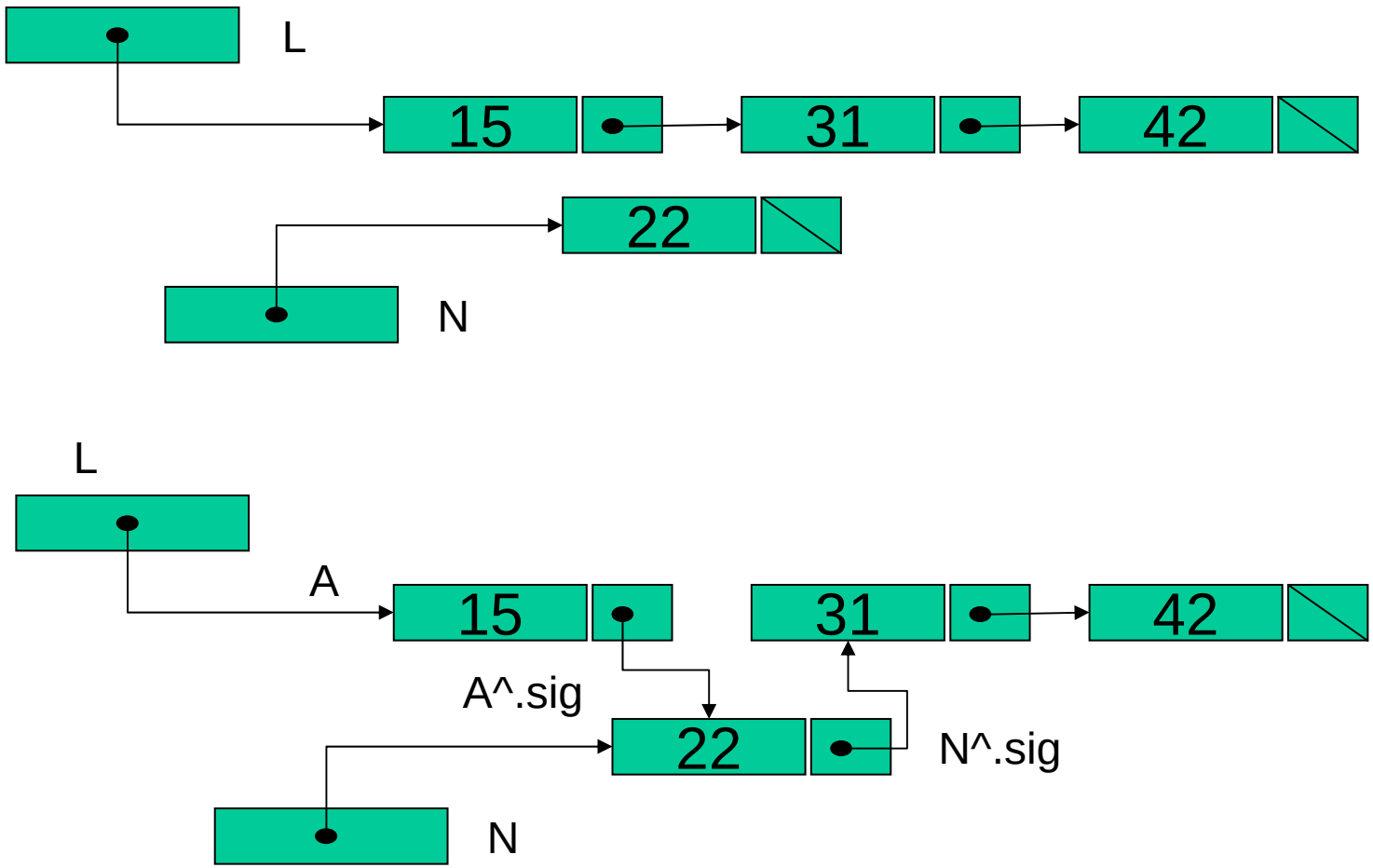
FUNCTION POSINSER



PROCEDURE INSERORDEN

```
PROCEDURE Inserorden (x:tElem;   var L:PtrNodo);
{Efecto. Si la lista es vacia inserta el nodo en el primer lugar, sino donde le corresponda}
VAR
    A,N:PtrNodo;
BEGIN
    N:=Crear(x);
    IF  Esvacia(L) THEN
          L:=N
        ELSE BEGIN
              A:=Posinser(x,L);
          IF A=nil THEN BEGIN{primera posición}
                N^.sig:=L;
                L:=N
          END; {If}
              ELSE BEGIN{posición intermedia}
                    N^.sig:=A^.sig;
                    A^.sig:=N
              END {Else}
         END {Else}
END; {Inserorden}
```

PROCEDURE INSERORDEN



BÚSQUEDA EN UNA LISTA ORDENADA

- Ahora la operación de búsqueda es más eficiente ya que para decidir si un elemento está o no en la lista basta con encontrar un elemento mayor.

  – **Buscorden (x,L):** Devuelve la dirección del nodo que contiene el campo x o nil si no se encuentra en la lista.

FUNCTION BUSCORDEN

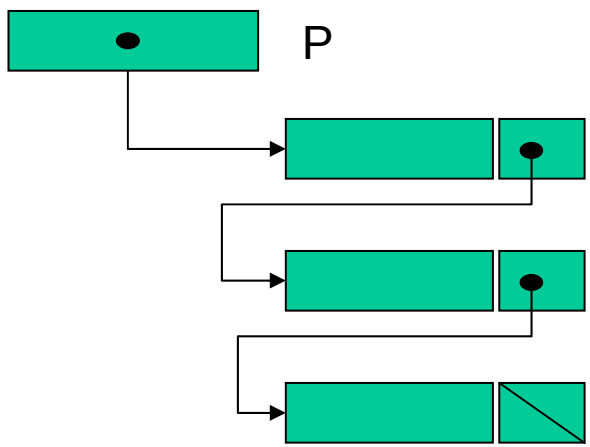
```
FUNCTION Buscorden (x:tElem;   L:PtrNodo):PtrNodo;
{Dev. La dirección del nodo con campo  x o  nil si no está en la lista}
BEGIN
     WHILE (L^.sig<>nil) AND (L^.info<x) DO
       L:=L^.sig;
     IF L^.info=x THEN
                Buscorden:=L
     ELSE
                Buscorden:=nil
END; {Buscorden}
```

DEFINICIÓN DEL TIPO PILA

- Una pila es un tipo lista en el que todas las inserciones y eliminaciones de elementos se realizan siempre por el mismo extremo.

- Estas estructuras son también llamadas listas LIFO (Last In-First Out)

- Una aplicación importante se encuentra en la implementación de la recursión.



DECLARACIÓN DEL TIPO PILA MEDIANTE EL ESQUEMA DE NODOS ENLAZADOS

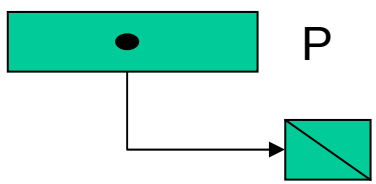
```
TYPE
tElem=<lo que corresponda>
tPila=^tNodoPila
tNodoPila=record
    info:tElem;
    sig:tPila;
End; {tNodoPila}

VAR
Pila:tPila;
```

OPERACIONES BÁSICAS DEL TIPO PILA

- **Crearpila(P):** Crea una pila vacia.
- **Espilavacia(P):** Función que Averigua si una pila es vacia.
- **Cima(P):**Consulta del contenido del primer elemento de una pila.
- **Crearnodo(x):** Crea un nodo de la pila para poder ser apilado.
- **Apilar(x,P):** Inserta un nuevo elemento en la pila (PUSH).
- **Suprimirdepila(P):** Elimina el elemento superior de la pila (POP).

PROCEDURE CREARPILA

```
PROCEDURE Crearpila(var pila:tPila);
{Postc. La pila que se crea es una pila vacia}
Begin
        pila:=nil
End; {Crearpila}
```

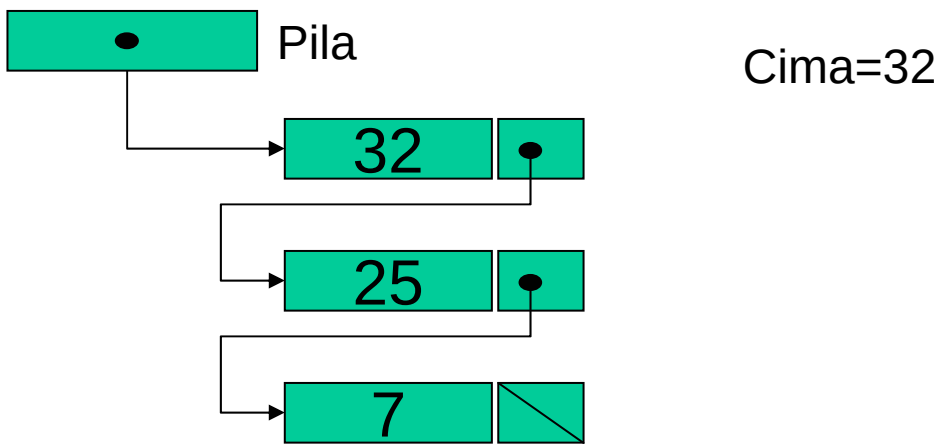


FUNCTION ESPILAVACIA

```
FUNCTION Espilavacia(pila:tPila):Boolean;
{Dev. Si la pila es vacia o no}
Begin
        Espilavacia:=(pila=nil)
End; {Espilavacia}
```

FUNCTION CIMA

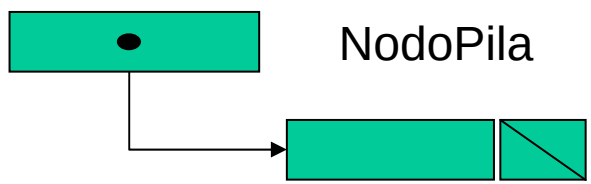
```
FUNCTION Cima(pila:tPila):tElem;
{Prec. La pila no esta vacia}
{Dev. El contenido del primer nodo}
Begin
        Cima:=pila^.info
End; {Cima}
```



Cima=32

FUNCTION CREARNODO

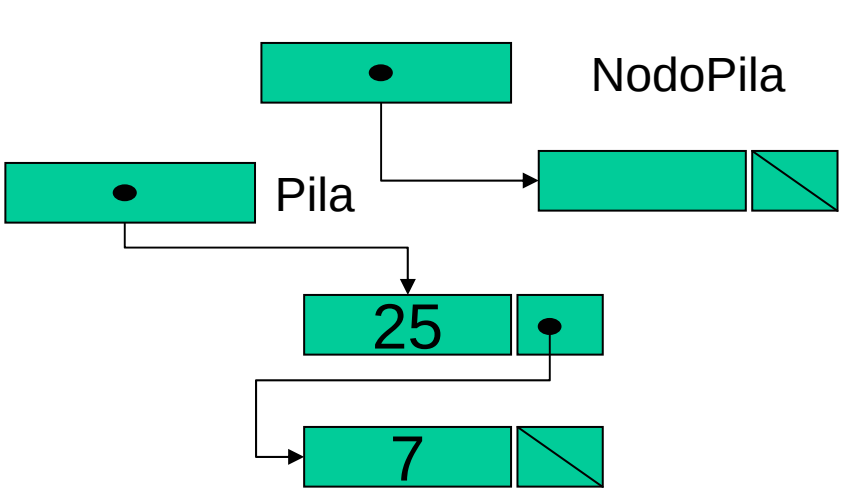
```
FUNCTION Crearnodo(x:tElem):tpila;
{Dev. Crea un nodo apilable}
VAR
    nodopila:tpila;
Begin
        New(nodopila);
        nodopila^.info:=x;
        nodopila^.sig:=nil;
        creanodo:=nodopila
End; {Crearnodo}
```

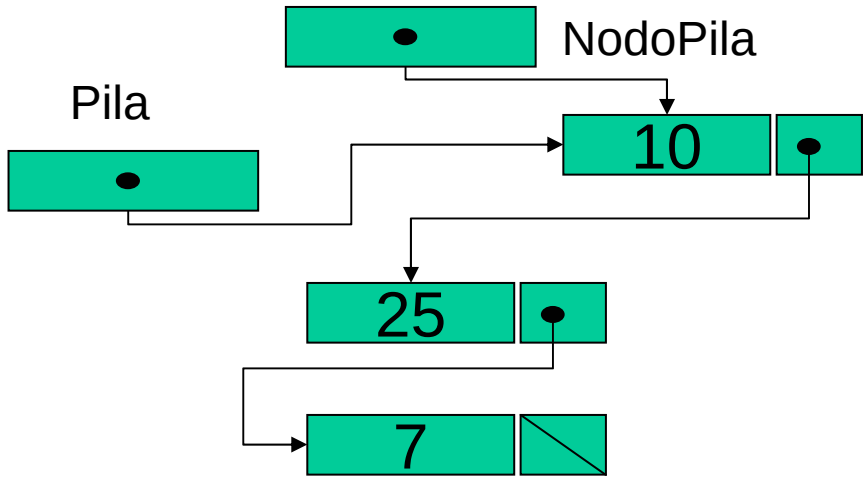


PROCEDURE APILAR

```
PROCEDURE Apilar(x:telem; var pila:tPila);
{Efecto. Se añade un nuevo nodo sobre la pila}
VAR
    pilaaux:tPila;
Begin
        pilaaux:=crearnodo(x);
        pilaaux^.sig:=pila;
        pila:=pilaaux;
End; {Apilar}
```

PROCEDURE APILAR

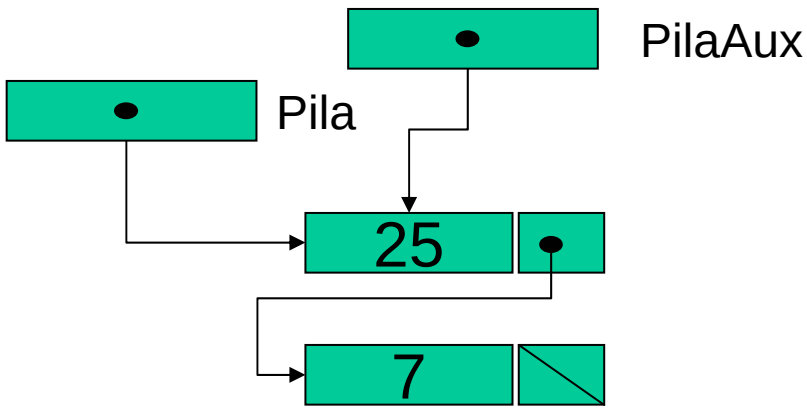


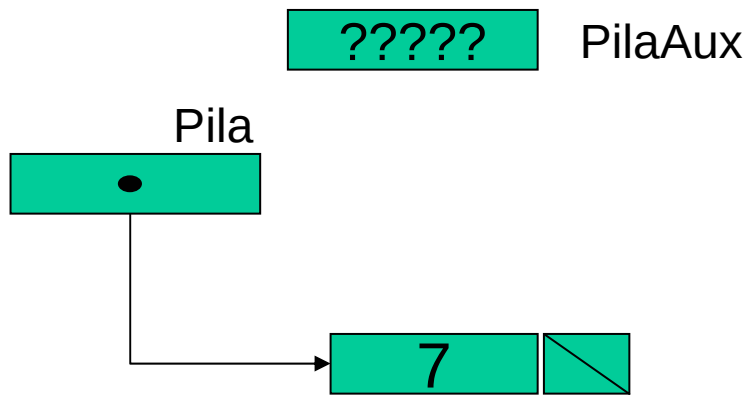


PROCEDURE SUPRIMIRDEPILA

```
PROCEDURE Suprimirdepila(var pila:tPila);
{Prec. La pila no está vacia}
{Efecto. Suprime un nodo de la pila}
VAR
    pilaAux:tPila;
Begin
        pilaAux:=pila;
        pila:=pilaAux^.sig;
        Dispose(pilaAux)
End; {Suprimirdepila}
```

PROCEDURE SUPRIMIRDEPILA





APLICACIONES DE LAS PILAS

Pilas y recursión:

- En cada llamada se añade una tabla de activación en una pila denominada recursiva. En esta pila se almacenan los argumentos y objetos locales con su valor en el momento de producirse la llamada.

- La recursión se puede implementar mediante una unidad pila. Bajo esta perspectiva la recursión se convierte en un par de bucles. El primero apila el segundo las desapila y evalúa.

EJEMPLO: FACTORIAL DE n

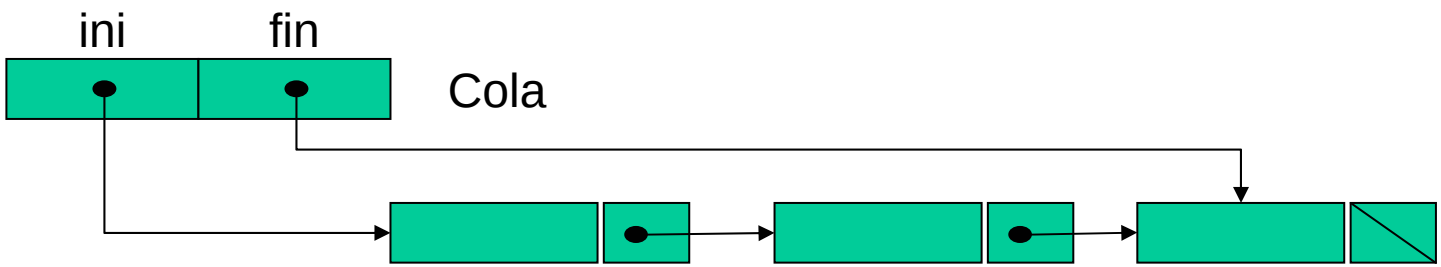
```
FUNCTION Faciter(num:integer):integer;
{Prec. num≥0}
{Dev. num!}
VAR
    pilarec:tpila;
    n,fac:integer;
BEGIN
    crearpila(pilarec);
    {primer bucle: apila las distintas llamadas}
    FOR  n:=num DOWNTO 1 DO
        Apilar(n,pilarec);
    {Segundo bucle:  resuelve las llamadas}
    fac:=1 {caso base}
    WHILE pilarec<>nil DO BEGIN
        fac:=cima(pilarec)*fac;
        suprimirdepila(pilarec)
    END; {while}
    Faciter:=fac
END; {Faciter}
```

DEFINICIÓN DEL TIPO COLA

- Una cola es un tipo lista en el que todas las inserciones se realizan por un extremo y todas las eliminaciones de elementos se realizan por el otro.

- Estas estructuras son también llamadas listas FIFO (First In-First Out)

- Una aplicación importante se encuentra en la gestión de los trabajos de una impresora en red..



DECLARACIÓN DEL TIPO COLA MEDIANTE EL ESQUEMA DE NODOS ENLAZADOS

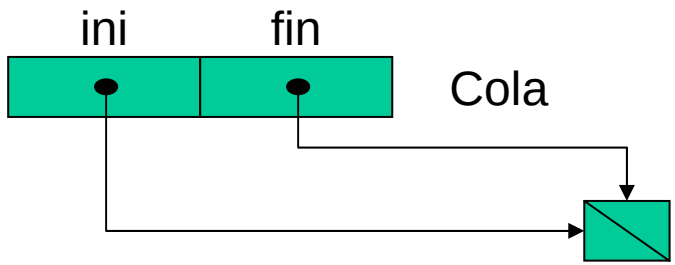
```
TYPE
tElem=<lo que corresponda>
tApNodo=^tNodoCola
tNodoCola=record
    info:tElem;
    sig:tApNodo;
End; {tNodoCola}

tCola=record
    ini:tApNodo;
    fin:tApNodo
end;{tCola}

VAR
Cola:tCola;
```

OPERACIONES BÁSICAS DEL TIPO COLA

- **Crearcola(C):** Crea una cola vacía.
- **Escolavacia(C):** Función que Averigua si una cola es vacia.
- **Consultaprim(C):**Consulta del contenido del primer elemento de la cola
- **Crearnodoc(x):** Crea un nodo de la cola para poder ser añadido.
- **Annadir(x,C):** Inserta un nuevo elemento al comienzo de la cola.
- **Sacardecola(C):** Elimina el elemento del final de la cola.

PROCEDURE CREARCOLA

```
PROCEDURE Crearcola(var cola:tcola);
{Postc. La cola que se crea es una cola vacía}
Begin
        cola.ini:=nil;
        cola.fin:=nil
End; {Crearcola}
```



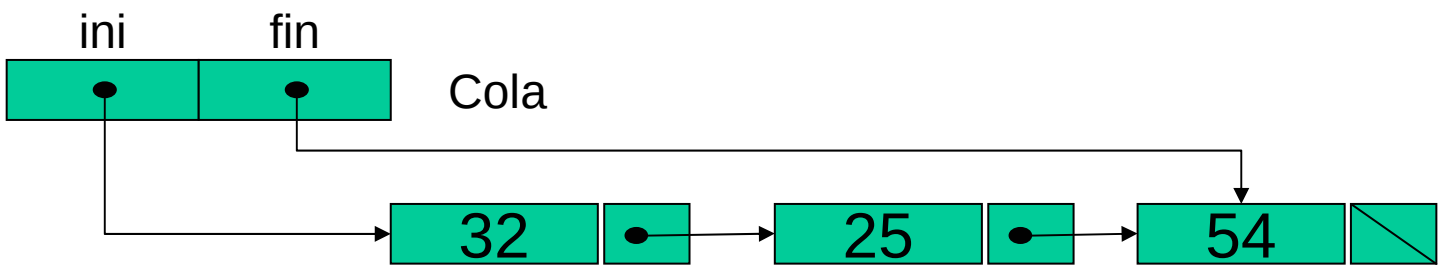
FUNCTION ESCOLAVACIA

```
FUNCTION Escolavacia(cola:tcola):Boolean;
{Dev. Si la cola es vacia o no}
Begin
    Escolavacia:=(cola.ini=nil)
End; {Escolavacia}
```

FUNCTION CONSULTAPRIM

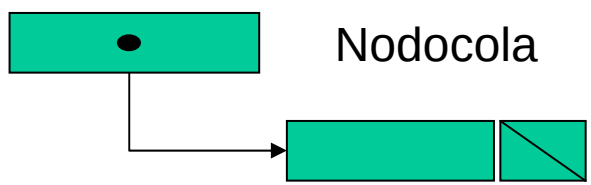
```
FUNCTION Consultaprim(cola:tcola):tElem;
{Prec. La cola no esta vacía}
{Dev. El contenido del primer nodo}
Begin
        Consultaprim:=cola.ini^.info
End; {Consultaprim}
```

Consultaprim=32



FUNCTION CREARNODOC

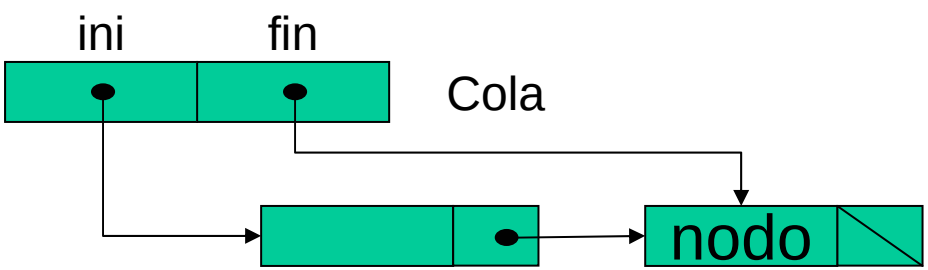
```
FUNCTION Crearnodoc(x:tElem):tApNodo;
{Dev. Crea un nodo que se puede añadir}
VAR
    nodocola:tApNodo;
Begin
        New(nodocola);
        nodocola^.info:=x;
        nodocola^.sig:=nil;
        creanodoc:=nodocola
End; {Crearnodo}
```

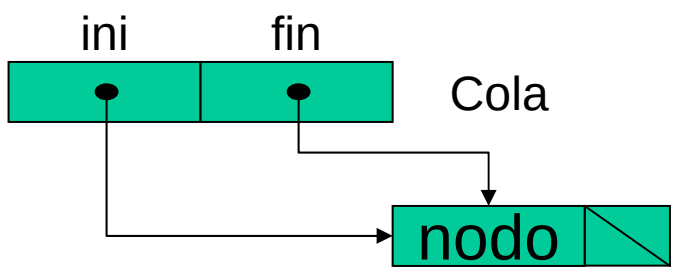


PROCEDURE ANNADIR

```
PROCEDURE Annadir(x:telem; var cola:tcola);
{Efecto. Se añade un nuevo nodo a la cola}
VAR
    nodo:tApNodo;
Begin
        nodo:=crearnodoc(x);
        if  not esvaciacola then begin
                cola.fin^.sig:=nodo;
                cola.fin:=nodo;
            end; {if}
        else begin {cola vacia}
                cola.fin:=nodo;
                cola.ini:=nodo
             end{else}
End; {Annadir}
```

PROCEDURE ANNADIR





PROCEDURE SACARDECOLA

```
PROCEDURE Sacardecola(var cola:tcola);
{Prec. La cola no es vacia}
{Efecto. Se elimina el primer nodo de la cola}
VAR
    nodoaux:tApNodo;
Begin
        nodoaux:=cola.ini
        if  not (cola.ini=cola.fin) then begin
                cola.ini:=nodoaux^.sig;
            end; {if}
        else begin {cola unitaria}
                cola.fin:=nil;
                cola.ini:=nil;
            end{else}
        dispose(nodoaux)
End; {Sacardecola}
```

PROCEDURE SACARDECOLA