

Tipos de Datos Abstractos

Tipo de dato Lista.

Tipos de Datos Abstractos

Tipo de dato Lista.

Tipo de Dato para la LISTA

A esta altura usted ya debe imaginar que para generar una lista dinámica necesitará declarar el tipo de dato para el nodo y para la lista ...

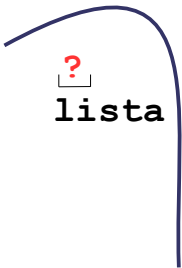
```
┌ 57 ─┐  
└ info tam sig ┘  
tNodo  
┌  
└ tLista
```

```
15  
16 typedef struct sNodo  
17 {  
18     void          *info;  
19     unsigned      tamInfo;  
20     struct sNodo  *sig;  
21 } tNodo;  
22 typedef tNodo *tLista;  
23  
24
```

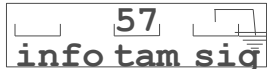
Tipos de Datos Abstractos

Tipo de dato Lista.

Variables del tipo tLista


lista

... y cuando declare una variable del tipo de dato tLista (con cualquier nombre, por ej.: lista), el resultado será (teniendo en cuenta que está recién declarada), que esta variable (que en el fondo es un puntero a nodo), tendrá algún valor indeterminado ...


info tam sig

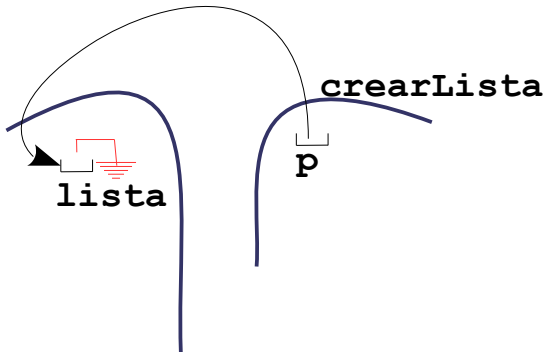
tNodo


tLista

Tipos de Datos Abstractos

Tipo de dato Lista.

Primitiva: crearLista



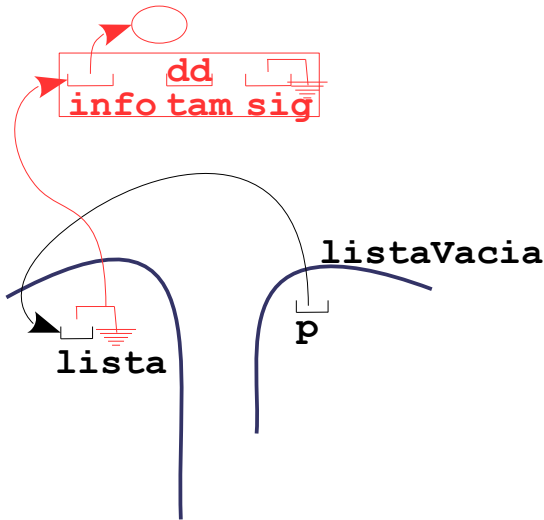
... por lo que será imprescindible la operación que crea la lista. Esta primitiva le asigna NULL al puntero lista del mismo modo que crear pila.

```
9
10 void crearLista(tLista *p)
11 {
12     *p = NULL;
13 }
14
```

Tipos de Datos Abstractos

Tipo de dato Lista.

Primitiva: listaVacia



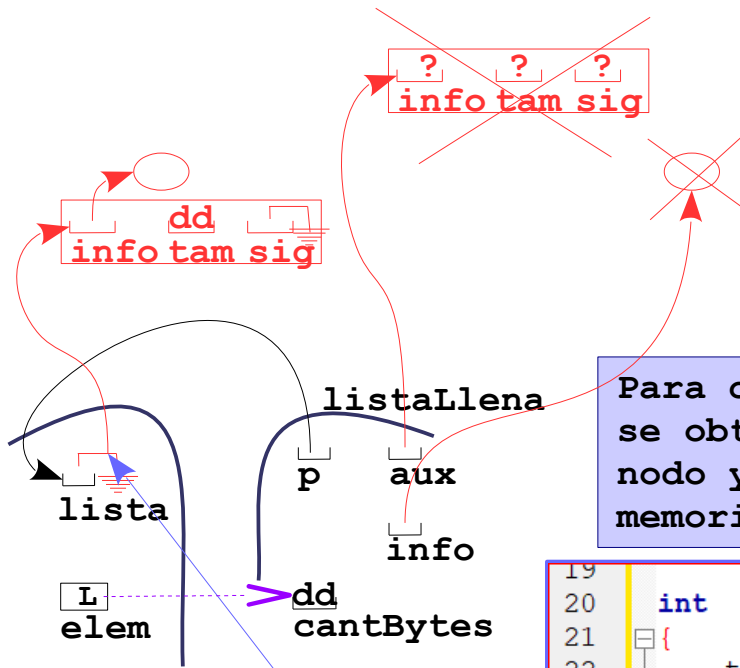
En cuanto a la primitiva que determina si la lista está vacía, es algo ya conocido, determina si lista (vista por medio de *p), tiene NULL.

```
14
15  int  listaVacia(const tLista *p)
16  {
17      return *p == NULL;
18  }
19
```

Tipos de Datos Abstractos

Tipo de dato Lista.

Primitiva: listaLlena



Para determinar si la lista está llena se obtiene memoria suficiente para un nodo y para la información, se libera la memoria y devuelve ...

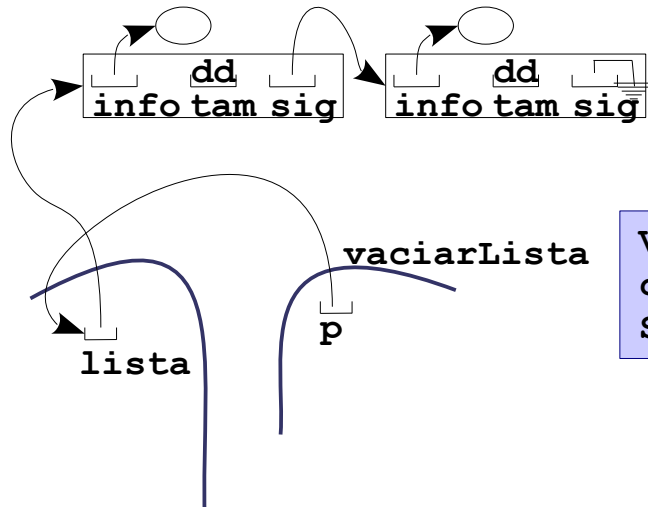
La lista puede o no estar vacía.
SE RECIBE LA LISTA (mediante un puntero), POR COMPATIBILIDAD CON MEMORIA ESTÁTICA, aunque no se use.

```
19
20  int  listaLlena(const tLista *p, unsigned cantBytes)
21  {
22      tNodo  *aux = (tNodo *)malloc(sizeof(tNodo));
23      void    *info = malloc(cantBytes);
24
25      free(aux);
26      free(info);
27      return aux == NULL || info == NULL;
28  }
29
```

Tipos de Datos Abstractos

Tipo de dato Lista.

Primitiva: vaciarLista



se comienza a ejecutar la función y determina que la lista no está vacía entrando en el ciclo repetitivo...

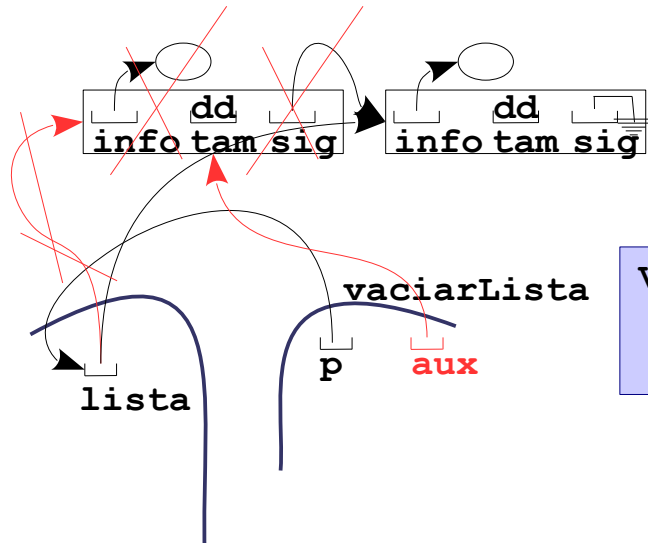
Vaciar la lista también es una primitiva conocida.
Suponga que hay nodos en la lista...

```
29  
30 void vaciarLista(tLista *p)  
31 {  
32     while(*p)  
33     {  
34         tNodo *aux = *p;  
35  
36         *p = aux->sig;  
37         free(aux->info);  
38         free(aux);  
39     }  
40 }  
41
```

Tipos de Datos Abstractos

Tipo de dato Lista.

Primitiva: vaciarLista (2)



Vaciar la lista ...

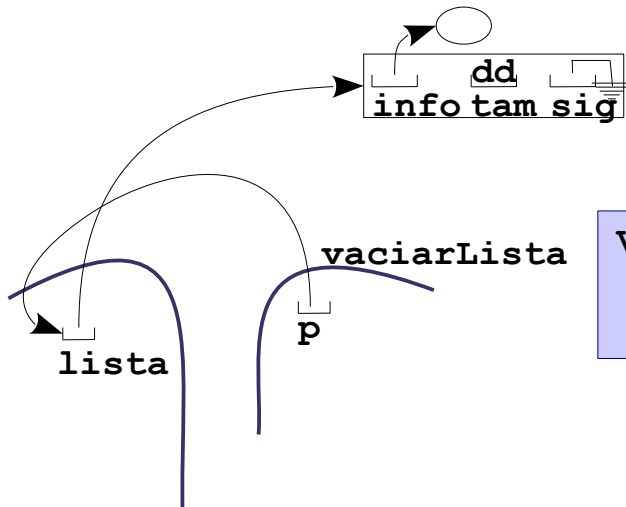
se crea la variable aux inicializada con la dirección del nodo apuntado por lista (*p). Hace que lista (*p) apunte al siguiente nodo. Libera la memoria de la información y luego la del nodo.

```
29
30 void vaciarLista(tLista *p)
31 {
32     while(*p)
33     {
34         tNodo *aux = *p;
35
36         *p = aux->sig;
37         free(aux->info);
38         free(aux);
39     }
40 }
41
```


Tipos de Datos Abstractos

Tipo de dato Lista.

Primitiva: vaciarLista (3)



vuelve a la condición del ciclo repetitivo, determina que hay nodo y entra en el ciclo.

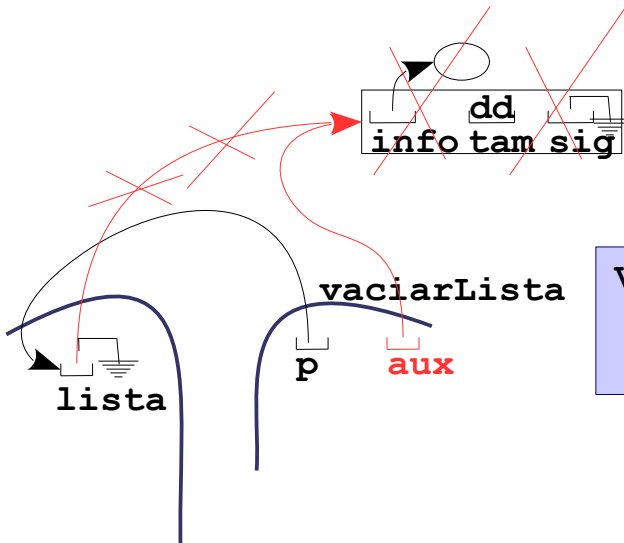
Vaciar la lista ...

```
29
30 void vaciarLista(tLista *p)
31 {
32     while(*p)
33     {
34         tNodo *aux = *p;
35
36         *p = aux->sig;
37         free(aux->info);
38         free(aux);
39     }
40 }
41
```

Tipos de Datos Abstractos

Tipo de dato Lista.

Primitiva: vaciarLista (4)



Vaciar la lista ...

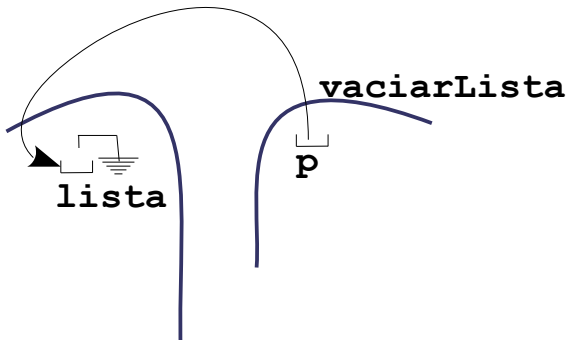
se crea la variable aux inicializada con la dirección del nodo apuntado por lista (*p). Hace que lista (*p) apunte al "siguiente nodo", como la dirección del "siguiente" es NULL, lista queda con NULL. Finalmente libera la memoria del nodo.

```
29
30 void vaciarLista(tLista *p)
31 {
32     while(*p)
33     {
34         tNodo *aux = *p;
35
36         *p = aux->sig;
37         free(aux->info);
38         free(aux);
39     }
40 }
41
```

Tipos de Datos Abstractos

Tipo de dato Lista.

Primitiva: vaciarLista (5)



Vaciar la lista ...

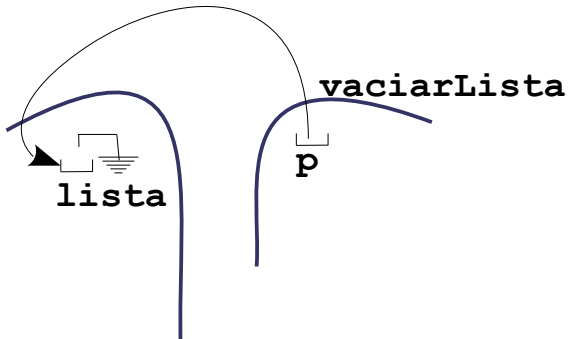
Al terminar de ejecutarse el bloque de código del ciclo repetitivo, vuelve a la condición. Evalúa `*p` y como la lista quedó vacía, la variable apuntada por `*p` (`lista`), tiene `NULL`, no sigue en el ciclo repetitivo.

```
29
30 void vaciarLista(tLista *p)
31 {
32     while (*p)
33     {
34         tNodo *aux = *p;
35
36         *p = aux->sig;
37         free(aux->info);
38         free(aux);
39     }
40 }
41
```

Tipos de Datos Abstractos

Tipo de dato Lista.

Primitiva: vaciarLista (6)



Vaciar la lista ...

Esto es lo que hubiera pasado si la lista hubiera estado vacía al invocar a la función.

```
29
30 void vaciarLista(tLista *p)
31 {
32     while(*p)
33     {
34         tNodo *aux = *p;
35
36         *p = aux->sig;
37         free(aux->info);
38         free(aux);
39     }
40 }
41
```

Tipos de Datos Abstractos

Tipo de dato Lista.

Primitiva: ponerAlComienzo

poner al comienzo de la lista, también es conocida, por su similitud con poner en pila

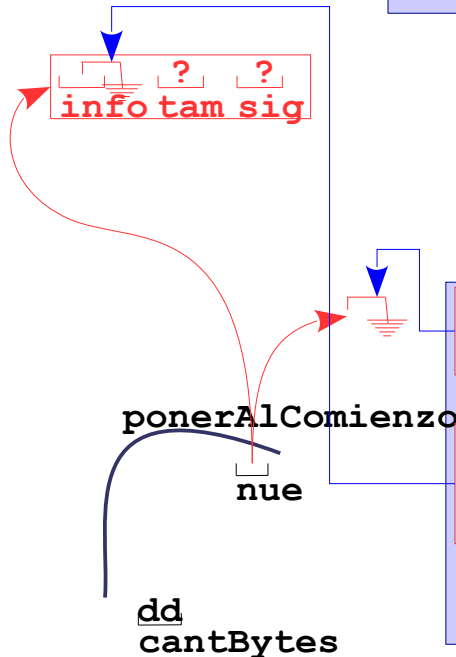
```
42  int  ponerAlComienzo(tLista *p, const void *d, unsigned cantBytes)
43  {
44      tNodo *nue;
45
46      if((nue = (tNodo *)malloc(sizeof(tNodo))) == NULL ||
47          (nue->info = malloc(cantBytes)) == NULL)
48      {
49          free(nue);
50          return 0;
51      }
52      memcpy(nue->info, d, cantBytes);
53      nue->tamInfo = cantBytes;
54      nue->sig = *p;
55      *p = nue;
56      return 1;
57  }
```

Veamos en detalle este trozo de código ...

Tipos de Datos Abstractos

Tipo de dato Lista.

Primitiva: ponerAlComienzo (2)



- si no hay memoria para el nodo
 - termina devolviendo FALSO
- si-no
 - si no hay memoria para el elemento
 - libera la memoria del nodo
 - termina devolviendo FALSO
 - fin-si
- fin-si

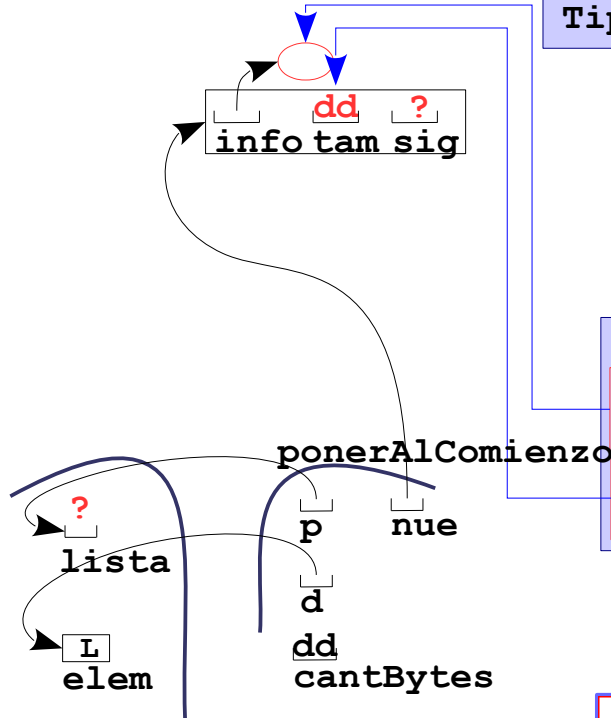
```
if((nue = (tNodo *)malloc(sizeof(tNodo))) == NULL ||
    (nue->info = malloc(cantBytes)) == NULL)
{
    free(nue);
    return 0;
}

memcpy(nue->info, d, cantBytes);
nue->tamInfo = cantBytes;
nue->sig = *p;
*p = nue;
```


Tipos de Datos Abstractos

Tipo de dato Lista.

Primitiva: ponerAlComienzo (4)



... hay memoria:

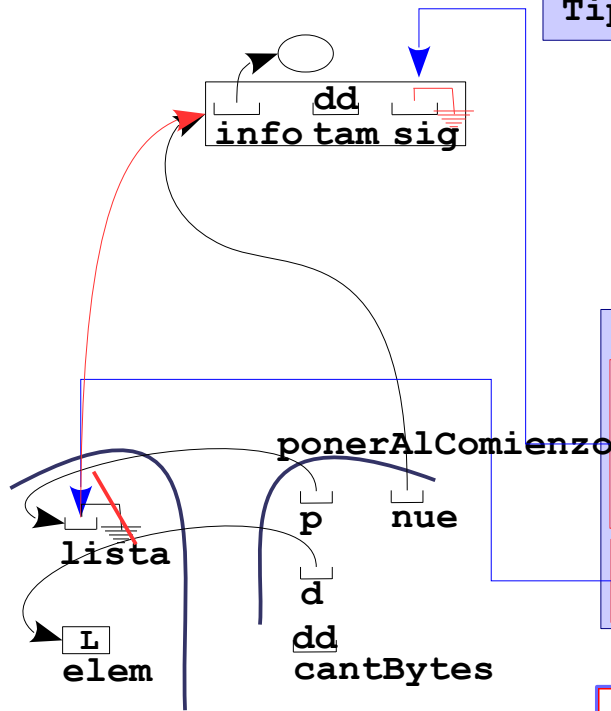
- copia el elemento en el espacio reservado.
- copia en el nodo la cantidad de bytes del elemento.

```
if((nue = (tNodo *)malloc(sizeof(tNodo))) == NULL ||  
    (nue->info = malloc(cantBytes)) == NULL)  
{  
    free(nue);  
    return 0;  
}  
memcpy(nue->info, d, cantBytes);  
nue->tamInfo = cantBytes;  
nue->sig = *p;  
*p = nue;
```


Tipos de Datos Abstractos

Tipo de dato Lista.

Primitiva: ponerAlComienzo (5)



Si la lista
estaba vacía

... con el nuevo nodo:

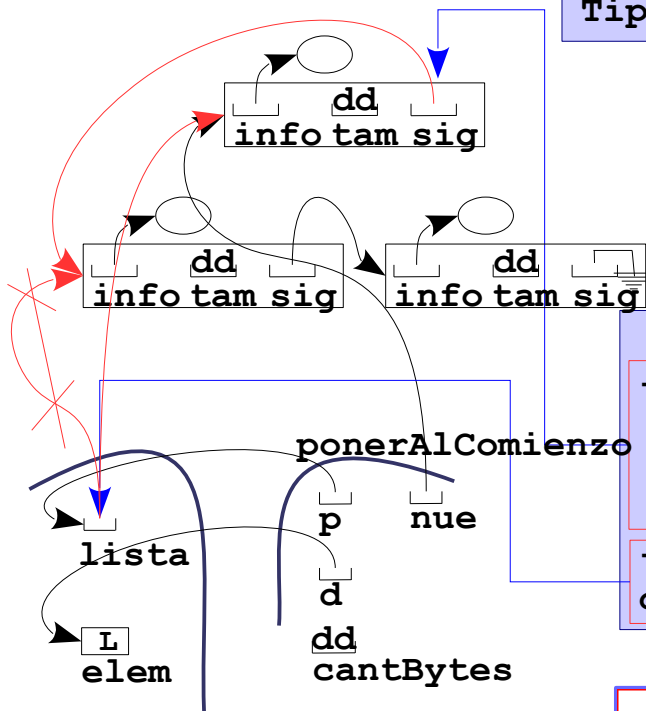
- al miembro sig le copia lo que apunta p (*p), es decir NULL si estaba vacía, o la dirección del primer nodo (que pasa a ser el segundo).
- a lo que apunta p (*p), le copia la dirección en que está el nuevo nodo

```
if((nue = (tNodo *)malloc(sizeof(tNodo))) == NULL ||  
    (nue->info = malloc(cantBytes)) == NULL)  
{  
    free(nue);  
    return 0;  
}  
memcpy(nue->info, d, cantBytes);  
nue->tamInfo = cantBytes;  
nue->sig = *p;  
*p = nue;
```

Tipos de Datos Abstractos

Tipo de dato Lista.

Primitiva: ponerAlComienzo (6)



Si la lista NO
estaba vacía

... con el nuevo nodo:

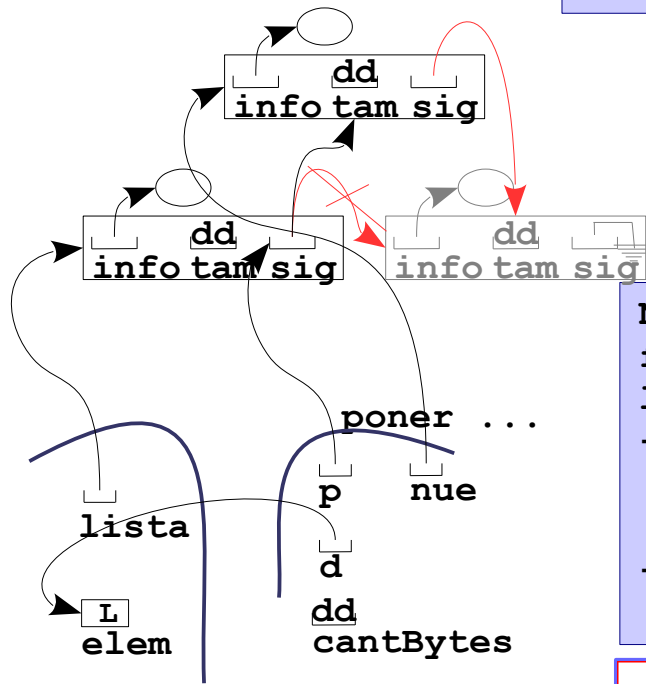
- al miembro sig le copia lo que apunta p (*p), es decir NULL si estaba vacía, o la dirección del primer nodo (que pasa a ser el segundo).
- a lo que apunta p (*p), le copia la dirección en que está el nuevo nodo

```
if((nue = (tNodo *)malloc(sizeof(tNodo))) == NULL ||  
    (nue->info = malloc(cantBytes)) == NULL)  
{  
    free(nue);  
    return 0;  
}  
memcpy(nue->info, d, cantBytes);  
nue->tamInfo = cantBytes;  
nue->sig = *p;  
*p = nue;
```

Tipos de Datos Abstractos

Tipo de dato Lista.

Primitiva: poner... (en general)



Poniendo al final o en medio de la lista

NOTA: si hubiera que poner un nodo al final o en una posición intermedia de la lista:

- al miembro sig del nuevo nodo con *p, le estaría copiando la dirección de un nodo (o NULL).
- a *p (miembro sig de un nodo), le copiará la dirección del nuevo nodo.

```
if((nue = (tNodo *)malloc(sizeof(tNodo))) == NULL ||
    (nue->info = malloc(cantBytes)) == NULL)
{
    free(nue);
    return 0;
}
memcpy(nue->info, d, cantBytes);
nue->tamInfo = cantBytes;
nue->sig = *p;
*p = nue;
```

Tipos de Datos Abstractos

Tipo de dato Lista.

Primitiva: sacarPrimeroLista

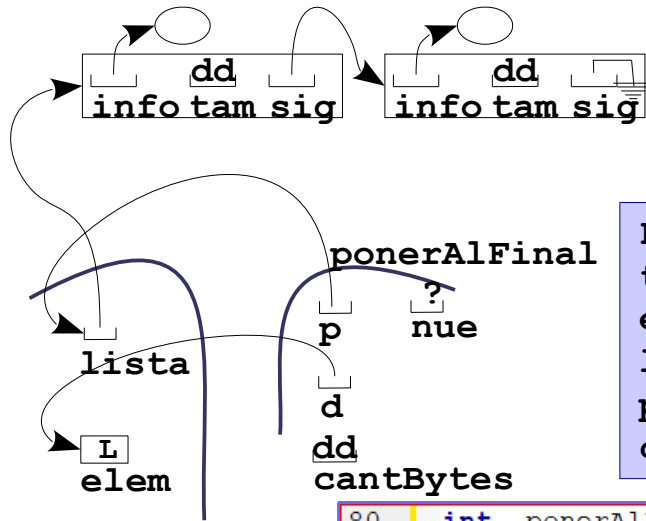
sacar el primero de la lista y ver el primero de la lista le deben resultar muy similares a las de pila y no tendrá mayor problema en resolverlas.

```
58
59  int  sacarPrimeroLista(tLista *p, void *d, unsigned cantBytes)
60  {
61      tNodo *aux = *p;
62
63      if(aux == NULL)
64          return 0;
65      *p = aux->sig;
66      memcpy(d, aux->info, minimo(cantBytes, aux->tamInfo));
67      free(aux->info);
68      free(aux);
69      return 1;
70  }
71
72  int  verPrimeroLista(const tLista *p, void *d, unsigned cantBytes)
73  {
74      if(*p == NULL)
75          return 0;
76      memcpy(d, (*p)->info, minimo(cantBytes, (*p)->tamInfo));
77      return 1;
78  }
79
```

Tipos de Datos Abstractos

Tipo de dato Lista.

Primitiva: ponerAlFinal



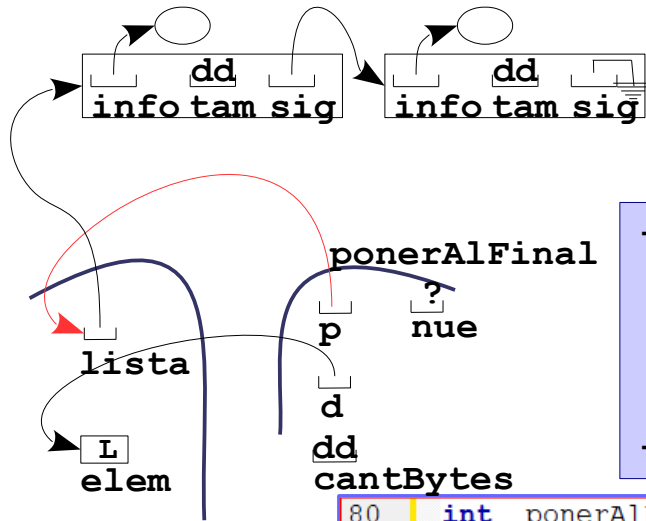
Para poner al final de la lista, la estrategia para codificarla consiste en entender que se recibe en p un puntero a la variable lista. "Mirando" lo apuntado por p (*p), se "ve" la variable lista, con lo cual se "sabe" si hay nodo o no.

```
80 int ponerAlFinal(tLista *p, const void *d, unsigned cantBytes)
81 {
82     tNodo *nue;
83
84     while(*p)
85         p = &(*p)->sig;
86     if((nue = (tNodo *)malloc(sizeof(tNodo))) == NULL) |
```

Tipos de Datos Abstractos

Tipo de dato Lista.

Primitiva: ponerAlFinal (2)



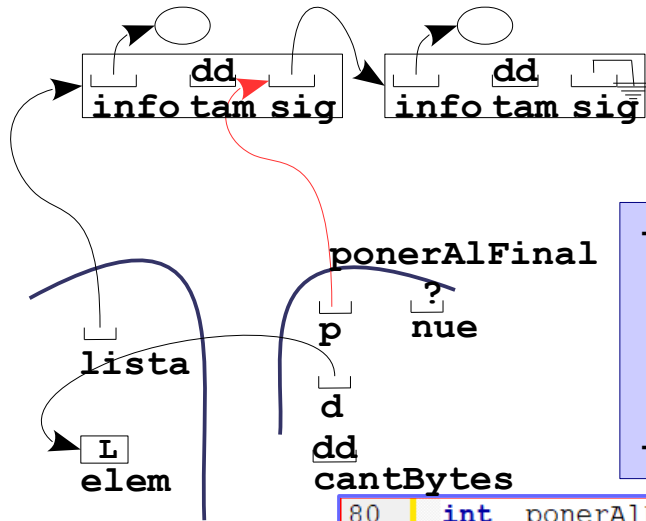
- mientras hay nodo
- toma en p la dirección de memoria en la que está el miembro sig de ese nodo (que es del mismo tipo de dato que la variable lista).
- fin-mientras.

```
80  int ponerAlFinal(tLista *p, const void *d, unsigned cantBytes)
81  {
82      tNodo *nue;
83
84      while(*p)
85          p = &(*p)->sig;
86      if((nue = (tNodo *)malloc(sizeof(tNodo))) == NULL, ||
```

Tipos de Datos Abstractos

Tipo de dato Lista.

Primitiva: ponerAlFinal (3)



- mientras hay nodo
- toma en p la dirección de memoria en la que está el miembro sig de ese nodo (que es del mismo tipo de dato que la variable lista). ←
- fin-mientras.

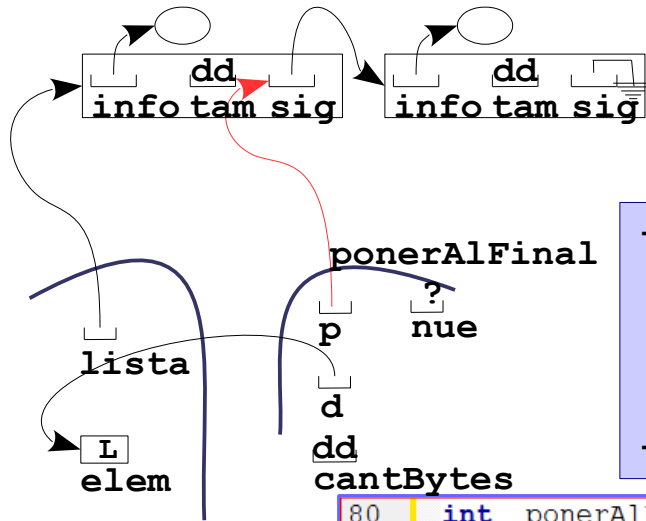
```
80 int ponerAlFinal(tLista *p, const void *d, unsigned cantBytes)
81 {
82     tNodo *nue;
83
84     while(*p)
85         p = &(*p)->sig; ←
86     if((nue = (tNodo *)malloc(sizeof(tNodo))) == NULL) ||
```

Deja de apuntar a la variable lista, para a continuación ver si hay un nodo.

Tipos de Datos Abstractos

Tipo de dato Lista.

Primitiva: ponerAlFinal (4)



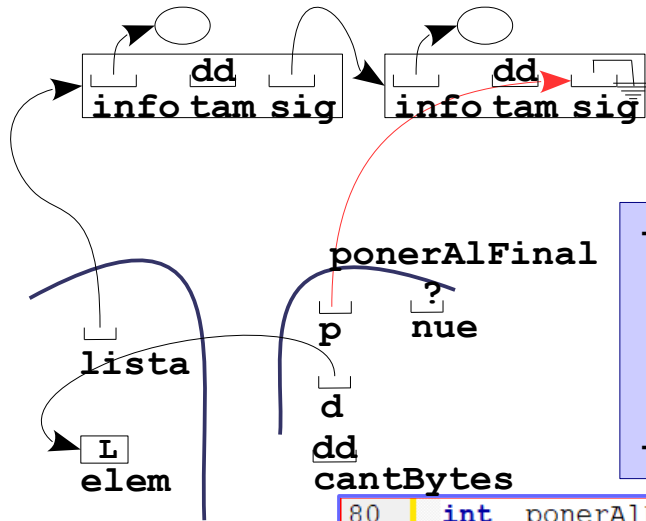
- mientras hay nodo
- toma en p la dirección de memoria en la que está el miembro sig de ese nodo (que es del mismo tipo de dato que la variable lista).
- fin-mientras.

```
80 int ponerAlFinal(tLista *p, const void *d, unsigned cantBytes)
81 {
82     tNodo *nue;
83
84     while(*p)
85         p = &(*p)->sig;
86     if((nue = (tNodo *)malloc(sizeof(tNodo))) == NULL, ||
```


Tipos de Datos Abstractos

Tipo de dato Lista.

Primitiva: ponerAlFinal (5)



- mientras hay nodo
- toma en p la dirección de memoria en la que está el miembro sig de ese nodo (que es del mismo tipo de dato que la variable lista). ←
- fin-mientras.

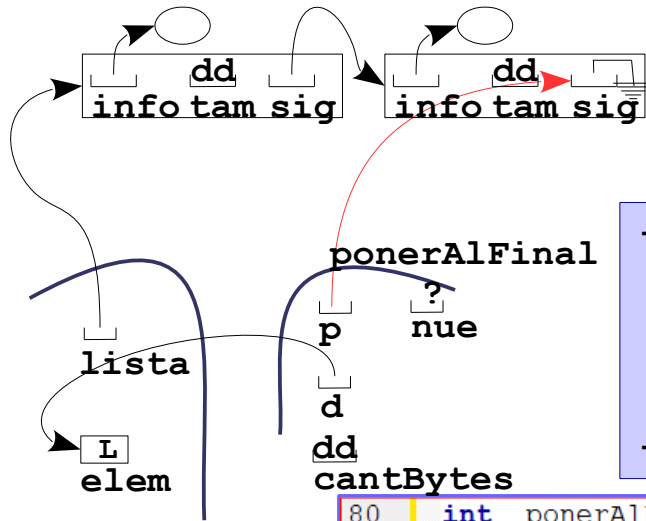
```
80 int ponerAlFinal(tLista *p, const void *d, unsigned cantBytes)
81 {
82     tNodo *nue;
83
84     while(*p)
85         p = &(*p)->sig; ←
86     if((nue = (tNodo *)malloc(sizeof(tNodo))) == NULL ||
```

Deja de apuntar a la variable lista, para ver si hay un nodo a continuación.

Tipos de Datos Abstractos

Tipo de dato Lista.

Primitiva: ponerAlFinal (6)



- mientras hay nodo
- toma en p la dirección de memoria en la que está el miembro sig de ese nodo (que es del mismo tipo de dato que la variable lista).
- fin-mientras.

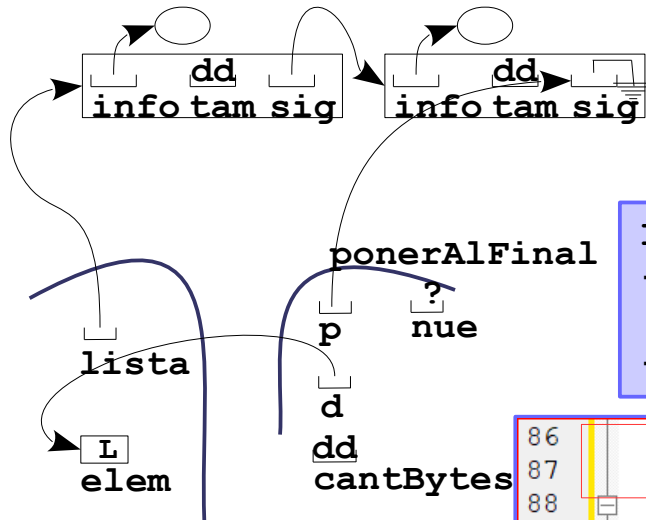
```
80 int ponerAlFinal(tLista *p, const void *d, unsigned cantBytes)
81 {
82     tNodo *nue;
83
84     while(*p)
85         p = &(*p)->sig;
86     if((nue = (tNodo *)malloc(sizeof(tNodo))) == NULL, ||
```

NO HAY NODO - SALE DEL CICLO REPETITIVO

Tipos de Datos Abstractos

Tipo de dato Lista.

Primitiva: ponerAlFinal (7)



Lo siguiente es conocido por todos.

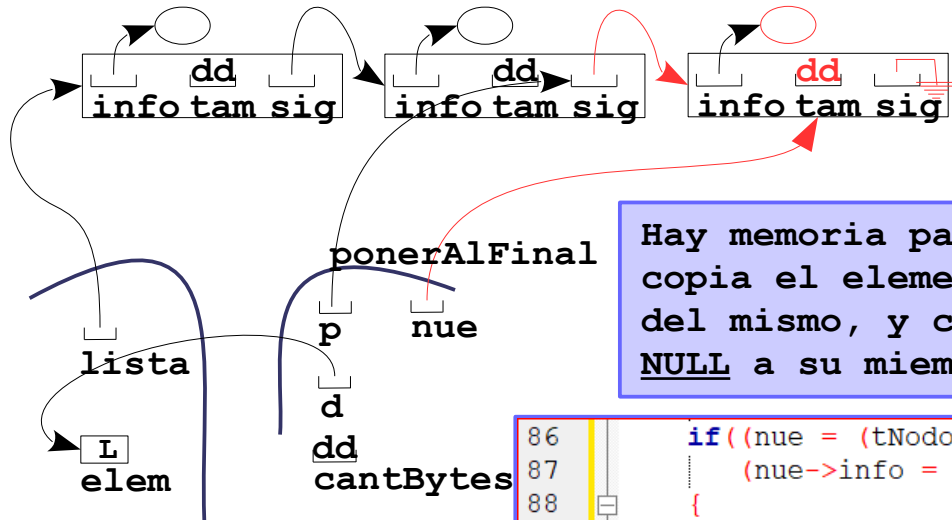
- si fracasa obteniendo memoria
- devuelve un indicador de falso
- fin-si

```
86     if((nue = (tNodo *)malloc(sizeof(tNodo))) == NULL ||  
87         (nue->info = malloc(cantBytes)) == NULL)  
88     {  
89         free(nue);  
90         return 0;  
91     }  
92     memcpy(nue->info, d, cantBytes);  
93     nue->tamInfo = cantBytes;  
94     nue->sig = NULL;  
95     *p = nue;  
96     return 1;  
97 }
```

Tipos de Datos Abstractos

Tipo de dato Lista.

Primitiva: ponerAlFinal (8)



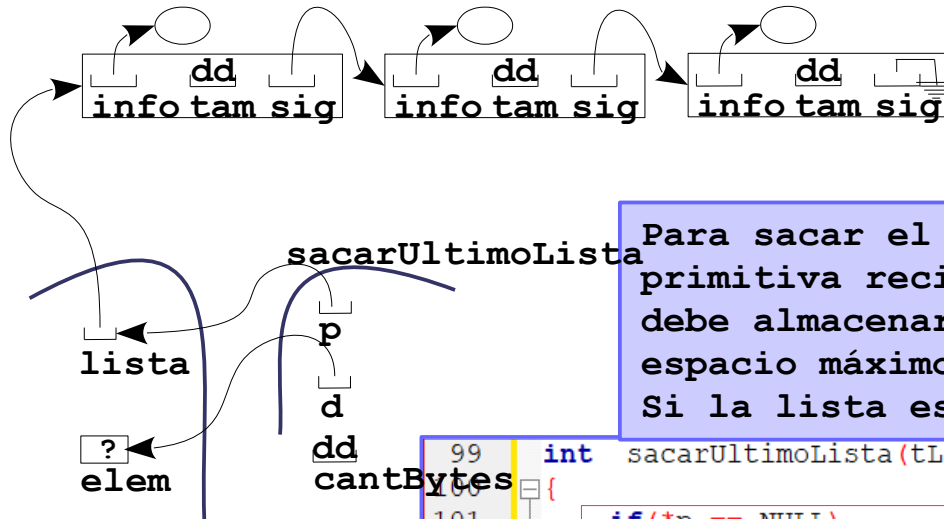
Hay memoria para el nuevo nodo...
copia el elemento, la cantidad de bytes
del mismo, y como es el último le asigna
NULL a su miembro sig y a *p lo enlaza.

```
86     if((nue = (tNodo *)malloc(sizeof(tNodo))) == NULL ||  
87         (nue->info = malloc(cantBytes)) == NULL)  
88     {  
89         free(nue);  
90         return 0;  
91     }  
92     memcpy(nue->info, d, cantBytes);  
93     nue->tamInfo = cantBytes;  
94     nue->sig = NULL;  
95     *p = nue;  
96     return 1;  
97 }
```

Tipos de Datos Abstractos

Tipo de dato Lista.

Primitiva: sacarUltimoLista



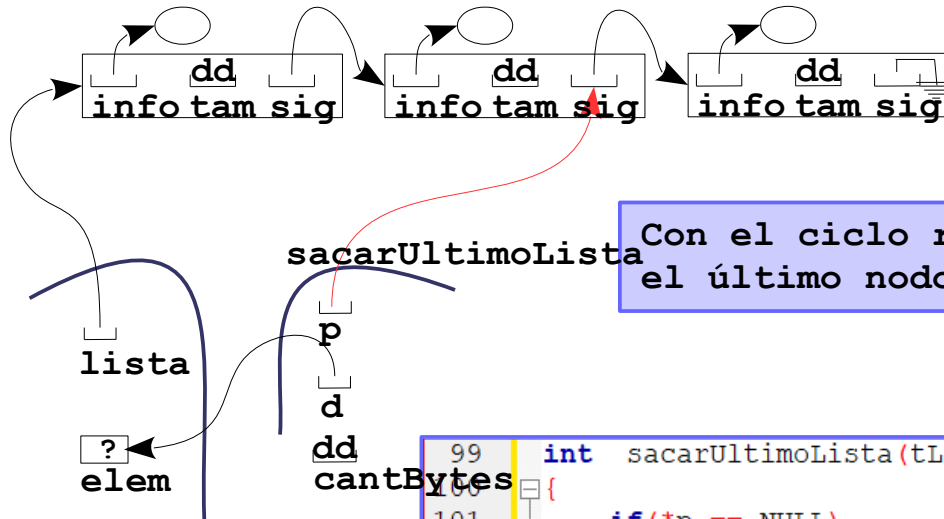
Para sacar el último de la lista, la primitiva recibe la lista (`p:E/S`), dónde debe almacenar la información (`d:S`), el espacio máximo del destino (`cantBytes:E`) Si la lista esá vacía, sale sin más.

```
99  int sacarUltimoLista(tLista *p, void *d, unsigned cantBytes)
100  {
101      if(*p == NULL)
102          return 0;
103      while((*p)->sig)
104          p = &(*p)->sig;
105      memcpy(d, (*p)->info, minimo(cantBytes, (*p)->tamInfo));
106      free((*p)->info);
107      free(*p);
108      *p = NULL;
109      return 1;
110  }
```

Tipos de Datos Abstractos

Tipo de dato Lista.

Primitiva: sacarUltimoLista (2)



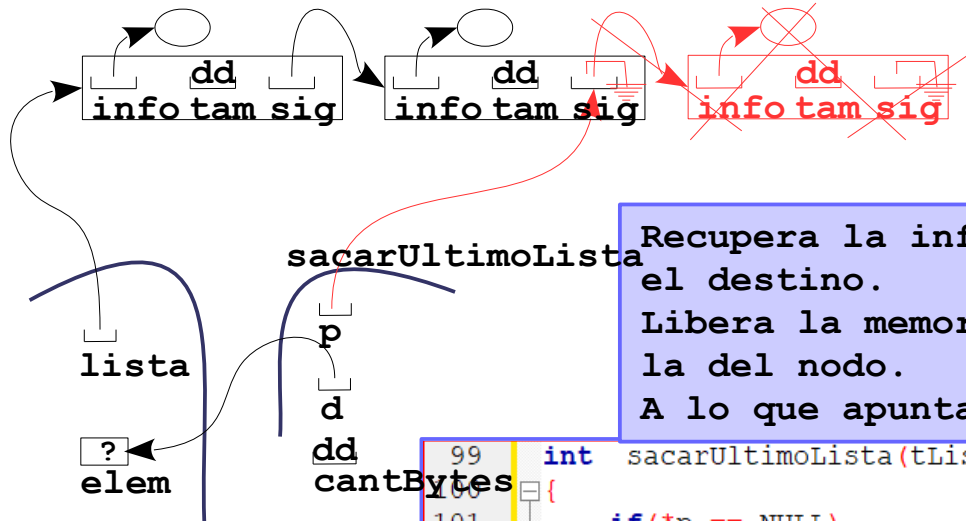
Con el ciclo repetitivo, se va a "ver" el último nodo (no tiene siguiente).

```
99  int sacarUltimoLista(tLista *p, void *d, unsigned cantBytes)
100  {
101      if(*p == NULL)
102          return 0;
103      while((*p)->sig)
104          p = &(*p)->sig;
105      memcpy(d, (*p)->info, minimo(cantBytes, (*p)->tamInfo));
106      free((*p)->info);
107      free(*p);
108      *p = NULL;
109      return 1;
110 }
```

Tipos de Datos Abstractos

Tipo de dato Lista.

Primitiva: sacarUltimoLista (3)



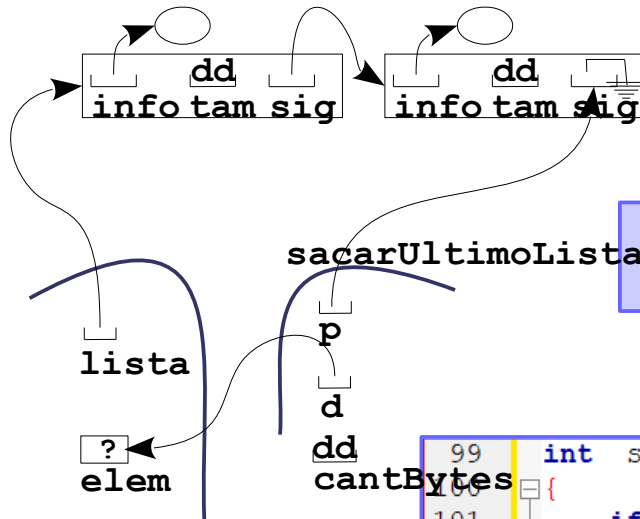
Recupera la información, sin desbordar el destino.
Libera la memoria del elemento y luego la del nodo.
A lo que apunta p (*p), le asigna NULL.

```
99  int sacarUltimoLista(tLista *p, void *d, unsigned cantBytes)
100  {
101      if(*p == NULL)
102          return 0;
103      while((*p)->sig)
104          p = &(*p)->sig;
105      memcpy(d, (*p)->info, minimo(cantBytes, (*p)->tamInfo));
106      free((*p)->info);
107      free(*p);
108      *p = NULL;
109      return 1;
110  }
```

Tipos de Datos Abstractos

Tipo de dato Lista.

Primitiva: sacarUltimoLista (4)



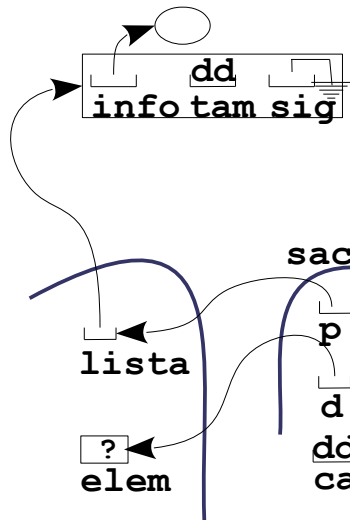
Termina su cometido devolviendo un indicador de éxito (1);

```
99  int  sacarUltimoLista(tLista *p, void *d, unsigned cantBytes)
100  {
101      if(*p == NULL)
102          return 0;
103      while((*p)->sig)
104          p = &(*p)->sig;
105      memcpy(d, (*p)->info, minimo(cantBytes, (*p)->tamInfo));
106      free((*p)->info);
107      free(*p);
108      *p = NULL;
109      return 1;
110  }
```


Tipos de Datos Abstractos

Tipo de dato Lista.

Primitiva: sacarUltimoLista (5)



sacarUltimoLista

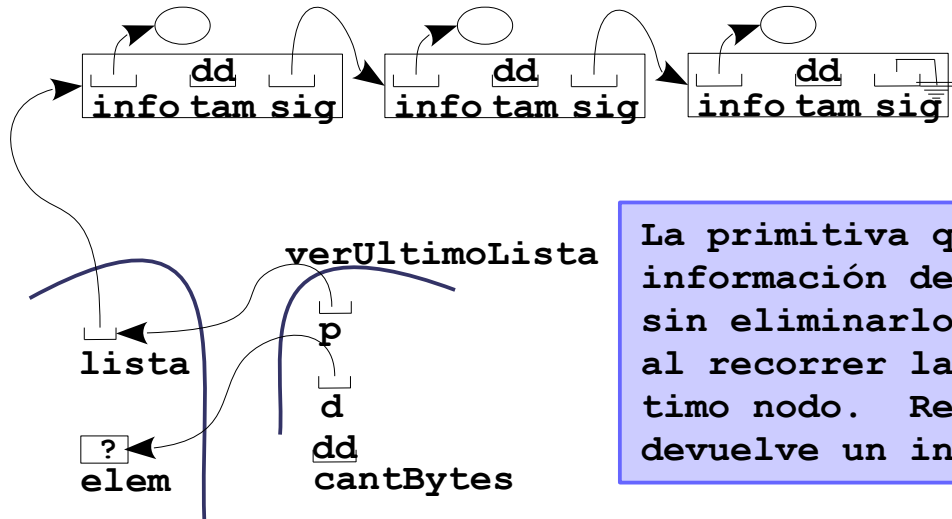
Cuando sólo hay un nodo, al invocar nuevamente a la primitiva, quedará el nodo eliminado y la variable lista con NULL.

```
99  int sacarUltimoLista(tLista *p, void *d, unsigned cantBytes)
100  {
101      if(*p == NULL)
102          return 0;
103      while((*p)->sig)
104          p = &(*p)->sig;
105      memcpy(d, (*p)->info, minimo(cantBytes, (*p)->tamInfo));
106      free((*p)->info);
107      free(*p);
108      *p = NULL;
109      return 1;
110  }
```

Tipos de Datos Abstractos

Tipo de dato Lista.

Primitiva: verUltimoLista



La primitiva que permite recuperar la información del último nodo de la lista, sin eliminarlo, procede de modo análogo al recorrer la lista hasta "ver" el último nodo. Recupera su información y devuelve un indicador de éxito.

```
112 int verUltimoLista(const tLista *p, void *d, unsigned cantBytes)
113 {
114     if(*p == NULL)
115         return 0;
116     while((*p)->sig)
117         p = &(*p)->sig;
118     memcpy(d, (*p)->info, minimo(cantBytes, (*p)->tamInfo));
119     return 1;
120 }
```

Tipos de Datos Abstractos

Tipo de dato Lista.