Estructuras dinámicas lineales (i)

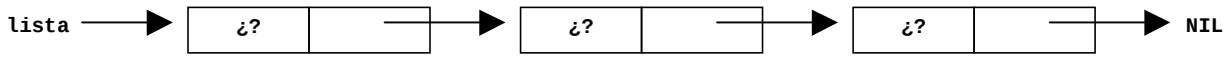
Introducción

En la lección anterior se explicaron los conceptos de variables dinámicas y puntero; vimos la forma en que se implementan dichas variables tanto en la notación algorítmica como en FORTRAN así como las peculiaridades de los POINTER en dicho lenguaje de programación.

Al final del capítulo se presentó un ejemplo sencillo de la utilización de variables dinámicas para la implementación de una estructura dinámica básica, en este caso una cola. En las próximas lecciones se profundizará en el estudio de estructuras dinámicas presentando las más habituales; comenzaremos en ésta con las más sencillas de todas: las estructuras dinámicas lineales.

Estructuras dinámicas lineales

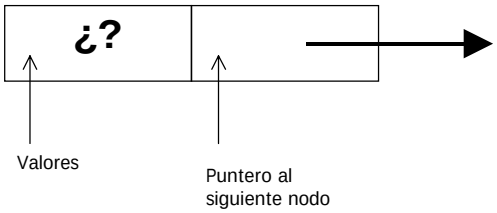
Decimos que una estructura de datos es dinámica si para su construcción se utilizan variables dinámicas, esto es, variables cuya creación y eliminación se lleva a cabo en tiempo de ejecución. La estructura de datos dinámica más sencilla posible consiste en una simple secuencia de variables enlazadas mediante punteros; esta estructura es, por razones obvias, lineal y recibe el nombre de “lista simplemente enlazada”.



Lista simplemente enlazada

Una lista es una secuencia de longitud variable de elementos del mismo tipo; los componentes de la lista (y en general de cualquier estructura dinámica) se denominan “nodos” y entre ellos existe una relación que permite pasar desde un nodo en particular al siguiente si es que existe; así, un tipo especial de lista es la lista vacía.

Los nodos se representan gráficamente de la forma siguiente:



Como se puede apreciar, un nodo tiene dos “zonas” de datos bien definidas, en una se almacenarán los datos de la estructura (enteros, reales, registros, etc.) mientras la otra será un puntero que indicará la dirección del siguiente elemento en la lista si existe o NIL para indicar que no hay siguiente elemento.

Además, para construir la lista es necesario un puntero externo a la lista que apunte al comienzo de la misma, es decir, el programa que va a utilizar la lista necesita un puntero a la “cabeza” de la lista; una vez se tiene dicho puntero a la cabeza (inicialmente nulo) es posible realizar sobre la lista las siguientes operaciones:

- Recorrerla.
- Buscar elementos.
- Insertar elementos.
- Eliminar elementos.
- Vaciarla.

En los siguientes apartados se explicarán las operaciones anteriores en la notación algorítmica ofreciendo algoritmos iterativos; en lecciones posteriores se implementarán los mismos algoritmos de forma recursiva así como en el lenguaje de programación FORTRAN.

Implementación de listas simplemente enlazadas en la notación algorítmica

Para implementar cualquier estructura dinámica es necesario hacer lo siguiente:

1. Determinar el tipo de datos que se va a almacenar.
2. Determinar la estructura del nodo que se va a utilizar.
3. Declarar un puntero en el programa principal que se utilizará para apuntar a la cabeza de la lista; dicho puntero inicialmente será nulo.

En el ejemplo que se mostrará a lo largo de este capítulo se creará una lista de números enteros; así pues, la estructura del nodo a emplear en dicha lista será la siguiente:

```
nodo = tupla
  numero ∈ entero
  siguiente ∈ puntero a nodo
fin tupla
```

Una vez se ha definido el tipo para los nodos de la lista ya es posible definir un puntero en el programa principal que apuntará a la cabeza de la futura lista, dicho puntero deberá ser inicialmente nulo:

```
cabeza ∈ puntero a nodo
cabeza ← NIL
```

Por último, para manipular la lista implementaremos las operaciones habituales:

- **Recorrer:** Esta operación será implementada como una acción puesto que no debe retornar ningún valor; recibirá como argumento la cabeza de la lista y mostrará por pantalla todos los elementos almacenados en la misma.
- **Buscar un elemento:** Esta operación será implementada como una función, recibirá como argumento un número entero y deberá retornar un puntero al primer nodo que tenga como valor el entero recibido o NIL si no lo encuentrar.
- **Insertar elemento:** La operación de inserción se implementará como una acción; como veremos más adelante esta acción define la naturaleza de la lista puesto que no es lo mismo insertar en la cabeza de la lista, en la cola o de forma ordenada. Recibirá como argumento un número entero.
- **Eliminar un elemento:** Esta operación recibirá un número entero como argumento y eliminará el primer nodo de la lista que tenga asignado dicho valor; será implementada como una función que retornará un valor lógico de manera que el usuario sepa si el elemento existía y fue eliminado o si, por el contrario, no existía.
- **Vaciar:** Esta operación tiene como fin eliminar todos los elementos de la lista, no recibe argumentos ni retorna ningún valor por lo que será implementada como una acción.

Todas las operaciones anteriores pueden implementarse tanto de forma iterativa como de forma recursiva; a continuación se proponen algoritmos para las mismas en su forma iterativa.

Operaciones sobre listas (simplemente enlazadas) implementadas iterativamente

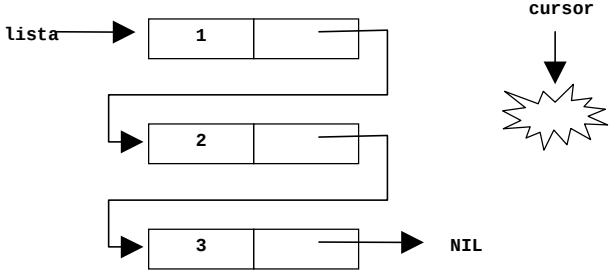
Recorrido de una lista simplemente enlazada

```
acción recorrer (cabezaLista ∈ puntero a nodo)
variables
  cursor ∈ puntero a nodo

inicio
  cursor ← cabezaLista

  mientras cursor ≠ NIL hacer
    escribir cursor↑.numero
    cursor ← cursor↑.siguiente
  fin mientras
fin acción
```

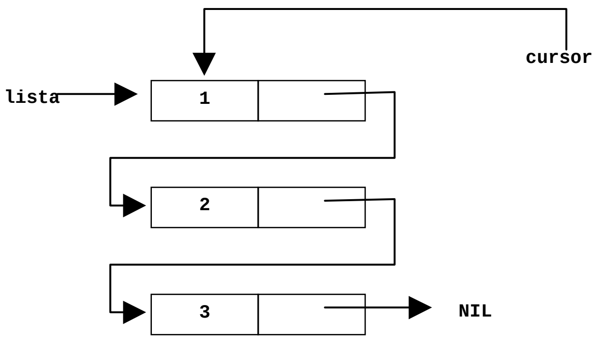
Como se puede observar, la acción recibe como argumento un puntero a la cabeza de la lista; sin embargo, no se utiliza dicho puntero para recorrer la estructura puesto que si se hiciera de esa forma, obviamente, perderíamos la cabeza de la lista y estaríamos "desmantelando" la estructura de datos.

En lugar de eso se utiliza una variable auxiliar, también de tipo puntero, `cursor`, que se utiliza para recorrer la lista. Inicialmente dicha variable se apunta hacia la cabeza de la lista y mientras apunte a "algo" distinto de NIL se va imprimiendo por pantalla el contenido del nodo apuntado y, una vez mostrado el dato, se le hace pasar al siguiente nodo.

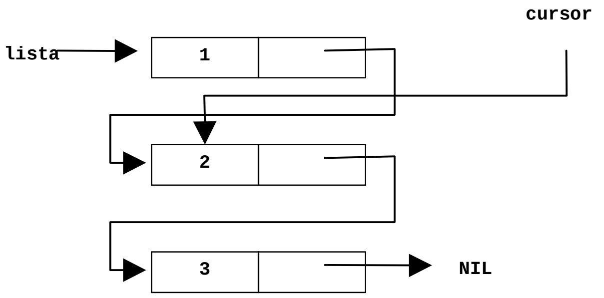
En la figura siguiente se muestra de forma gráfica la ejecución de esta acción para una lista que contuviera los enteros 1, 2 y 3.



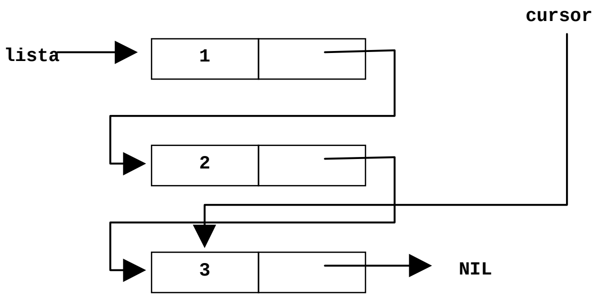

```
inicio
  cursor ← cabezaLista

  mientras cursor ≠ NIL hacer
    escribir cursor↑.numero
    cursor ← cursor↑.siguiente
  fin mientras
fin acción
```



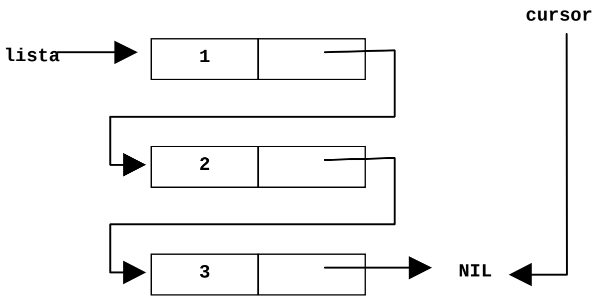

```
inicio
  cursor ← cabezaLista

  mientras cursor ≠ NIL hacer
    escribir cursor↑.numero
    cursor ← cursor↑.siguiente
  fin mientras
fin acción
```



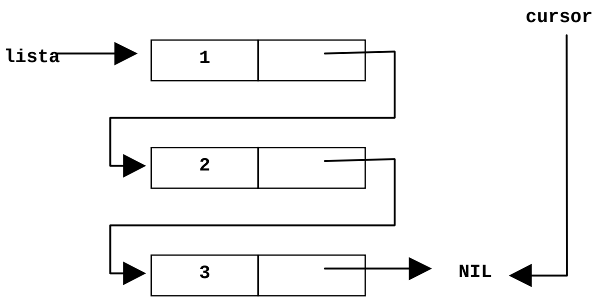

```
inicio
  cursor ← cabezaLista

  mientras cursor ≠ NIL hacer
    escribir cursor↑.numero
    cursor ← cursor↑.siguiente
  fin mientras
fin acción
```




```
inicio
  cursor ← cabezaLista

  mientras cursor ≠ NIL hacer
    escribir cursor↑.numero
    cursor ← cursor↑.siguiente
  fin mientras
fin acción
```




```
inicio
  cursor ← cabezaLista

  mientras cursor ≠ NIL hacer
    escribir cursor↑.numero
    cursor ← cursor↑.siguiente
  fin mientras
fin acción
```




```
inicio
  cursor ← cabezaLista

  mientras cursor ≠ NIL hacer
    escribir cursor↑.numero
    cursor ← cursor↑.siguiente
  fin mientras
fin acción
```

Búsqueda de un elemento en una lista simplemente enlazada

La operación de búsqueda se basa en la de recorrido; básicamente se trata de recorrer la lista hasta que se encuentre el elemento o llegar al final, retornando el cursor.

Si el elemento ha sido encontrado el cursor apuntará a una posición de memoria y si no ha sido encontrado retornará NIL.

```
puntero a nodo función buscar (cabezaLista ∈ puntero a nodo, elemento ∈ entero)
variables
  cursor ∈ puntero a nodo

inicio
  cursor ← cabezaLista

  mientras cursor ≠ NIL y cursor↑.numero ≠ elemento hacer
    cursor ← cursor↑.siguiente
  fin mientras

  buscar ← cursor
fin acción
```

Inserción de un elemento en una lista simplemente enlazada

Como ya hemos dicho la operación de inserción marca la naturaleza de la lista. Si la inserción se hace al final de la lista se está implementando una cola, también denominada lista FIFO (*First In First Out*, “el primero que entra es el primero que sale”). Si la inserción se hace al principio de la lista (por la cabeza) se está implementando una pila, también denominada lista LIFO (*Last In First Out*, “el último que entra es el primero que sale”). Si la inserción se hace siguiendo un criterio de orden entre los nodos tendríamos una lista ordenada.

Independientemente del tipo de lista que se desee implementar lo que está claro es que a la hora de introducir un dato en una lista tendremos los siguientes elementos:

1. Un puntero a un nodo de la lista.
2. El dato a introducir.

Entonces habrá que decidir si el dato a introducir se inserta delante del nodo seleccionado o detrás; como veremos, el hecho de que la lista sea simplemente enlazada influye enormemente en los algoritmos de inserción. A continuación mostraremos una acción que insertará un elemento delante de un entero y otra que lo insertará detrás; cada acción recibirá dos enteros, el primero es el dato a insertar y el segundo es el dato respecto al cual se producirá la inserción.

Un caso especial se produce cuando se debe insertar en una lista vacía; el algoritmo para insertar en una lista vacía sería el siguiente:

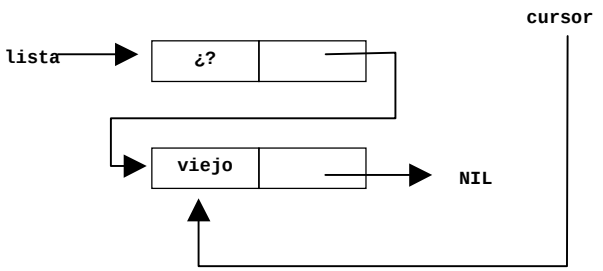
```
acción insertarVacía (cabezaLista ∈ puntero a nodo, dato ∈ entero)
inicio
  si cabezaLista = NIL entonces
    crear (cabezaLista)
    cabezaLista↑.siguiente ← NIL
    cabezaLista↑.numero ← dato
  fin si
fin acción
```

Inserción “por delante” de un elemento en una lista simplemente enlazada

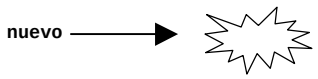
Al insertar un elemento nuevo en la lista tendremos que crear un nuevo nodo en la lista (recuérdese que se trata de estructuras dinámicas). Una vez se ha creado ese nuevo nodo habrá que “enlazar” la lista de forma adecuada y mantener el orden especificado; si la inserción del dato N debía realizarse “delante” del dato V, la lista deberá tener primero el nodo con el dato N seguido del nodo con el dato V. A continuación se muestra un algoritmo que busca el dato V y, en caso de que exista, procede a insertar “delante” el dato N (para simplificar el algoritmo no se contempla que la lista pueda estar vacía); posteriormente se muestran de forma gráfica los pasos que se llevan a cabo.

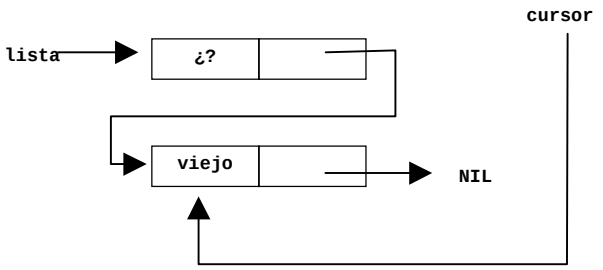
```
acción insertarDelante (cabezaLista ∈ puntero a nodo, dato_nuevo, dato_viejo ∈ entero)
variables
  cursor, nuevo ∈ puntero a nodo

inicio
  cursor ← cabezaLista

  mientras cursor ≠ NIL y cursor↑.numero ≠ dato_viejo hacer
    cursor ← cursor↑.siguiente
  fin mientras

  si cursor ≠ NIL entonces
    crear (nuevo)
    nuevo↑.siguiente ← cursor↑.siguiente
```

```
    cursor↑.siguiente ← nuevo
    nuevo↑.numero ← cursor↑.numero
    cursor↑.numero ← dato_nuevo
  fin si
fin acción
```

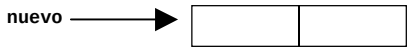


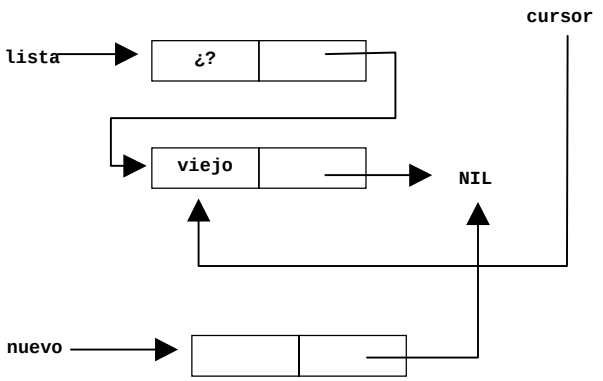

```
si cursor ≠ NIL entonces
  crear (nuevo)
  nuevo↑.siguiente ← cursor↑.siguiente
  cursor↑.siguiente ← nuevo
  nuevo↑.numero ← cursor↑.numero
  cursor↑.numero ← dato_nuevo
fin si
```



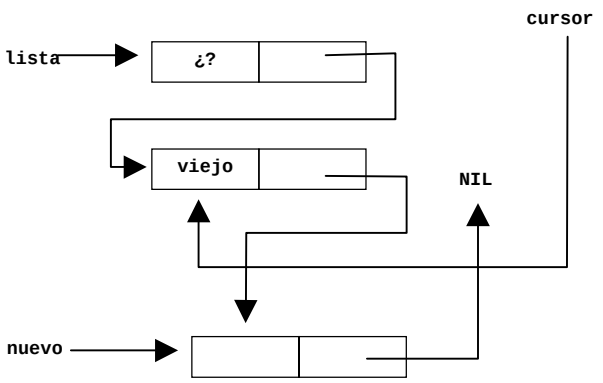



```
si cursor ≠ NIL entonces
  crear (nuevo)
  nuevo↑.siguiente ← cursor↑.siguiente
  cursor↑.siguiente ← nuevo
  nuevo↑.numero ← cursor↑.numero
  cursor↑.numero ← dato_nuevo
fin si
```

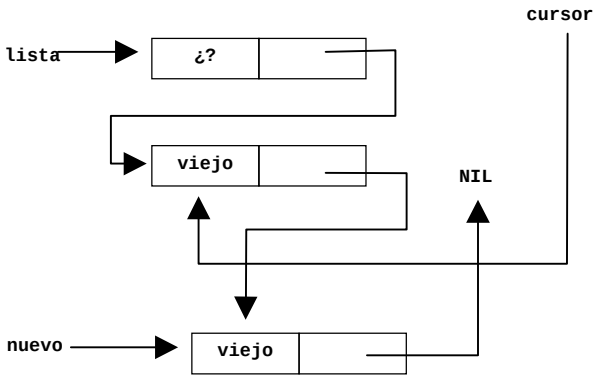




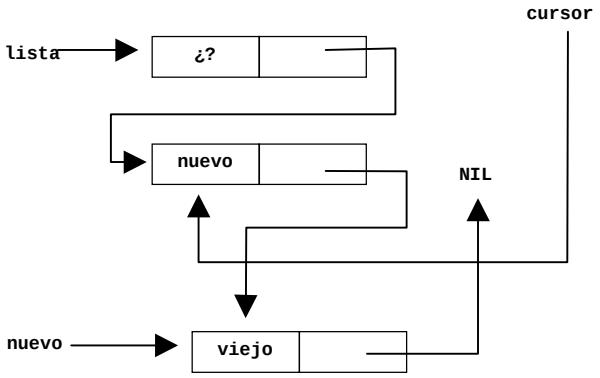

```
si cursor ≠ NIL entonces
  crear (nuevo)
  nuevo↑.siguiente ← cursor↑.siguiente
  cursor↑.siguiente ← nuevo
  nuevo↑.numero ← cursor↑.numero
  cursor↑.numero ← dato_nuevo
fin si
```



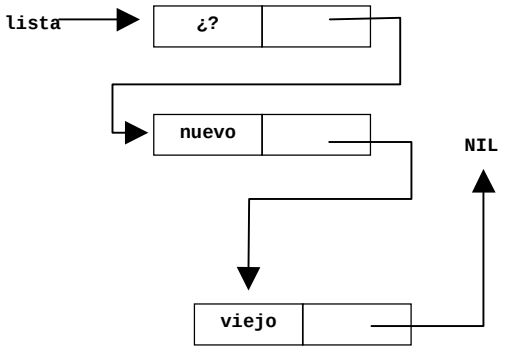

```
si cursor ≠ NIL entonces
  crear (nuevo)
  nuevo↑.siguiente ← cursor↑.siguiente
  cursor↑.siguiente ← nuevo
  nuevo↑.numero ← cursor↑.numero
  cursor↑.numero ← dato_nuevo
fin si
```




```
si cursor ≠ NIL entonces
  crear (nuevo)
  nuevo↑.siguiente ← cursor↑.siguiente
  cursor↑.siguiente ← nuevo
  nuevo↑.numero ← cursor↑.numero
  cursor↑.numero ← dato_nuevo
fin si
```




```
si cursor ≠ NIL entonces
  crear (nuevo)
  nuevo↑.siguiente ← cursor↑.siguiente
  cursor↑.siguiente ← nuevo
  nuevo↑.numero ← cursor↑.numero
  cursor↑.numero ← dato_nuevo
fin si
```



Al retornar de la acción las variables locales `cursor` y `nuevo` desaparecen y la lista queda como se muestra en la figura.

Inserción “por detrás” de un elemento en una lista simplemente enlazada

También aquí es necesario crear un nuevo nodo; sin embargo, en este caso la estructura de la lista hace innecesario el intercambio de valores entre el nodo “nuevo” y el “viejo”

```
acción insertarDetras (cabezaLista ∈ puntero a nodo, dato_nuevo, dato_viejo ∈ entero)
variables
  cursor, nuevo ∈ puntero a nodo

inicio
  cursor ← cabezaLista

  mientras cursor ≠ NIL y cursor↑.numero ≠ dato_viejo hacer
    cursor ← cursor↑.siguiente
  fin mientras

  si cursor ≠ NIL entonces
    crear (nuevo)
    nuevo↑.siguiente ← cursor↑.siguiente
    nuevo↑.numero ← dato_nuevo
    cursor↑.siguiente ← nuevo
  fin si
fin acción
```

Como se puede ver, la única diferencia entre este algoritmo y el anterior es la eliminación de la sentencia en que se asigna al nodo `nuevo` el `dato_viejo` y al `cursor` el `dato_nuevo`.

A continuación, basándonos en estos dos algoritmos (y en el de inserción en una lista vacía), se implementarán las acciones para insertar en una cola, en una pila y en una lista ordenada ascendentemente.

En el primer caso se tratará simplemente de una inserción “por detrás” con la particularidad de que siempre se hará a partir del último elemento de la lista; en el caso de una pila se utilizará una inserción “por delante” haciéndose siempre en la cabeza de la lista y en el caso de una inserción ordenada tiene que hacerse “por detrás” si el dato a introducir es mayor que cualquiera de los datos existentes en la lista y “por delante” si es menor o igual.

Inserción en una cola

Como ya se ha dicho, la inserción en una cola precisa buscar el último elemento de la lista e insertar el nuevo elemento detrás del mismo.

```
acción insertarCola (cabezaLista ∈ puntero a nodo, dato_nuevo ∈ entero)
variables
  cursor, nuevo ∈ puntero a nodo

inicio
  if cabezaLista ≠ NIL entonces
    cursor ← cabezaLista

    mientras cursor↑.siguiente ≠ NIL hacer
      cursor ← cursor↑.siguiente
    fin mientras

    crear (nuevo)
    nuevo↑.siguiente ← cursor↑.siguiente
    nuevo↑.numero ← dato_nuevo
    cursor↑.siguiente ← nuevo
  si no
    crear (cabezaLista)
    cabezaLista↑.siguiente ← NIL
    cabezaLista↑.numero ← dato
  fin si
fin acción
```

Inserción en una pila

Como ya se ha dicho, la inserción en una pila consiste en insertar delante de la cabeza de la lista.

```
acción insertarPila (cabezaLista ∈ puntero a nodo, dato_nuevo ∈ entero)
variables
  nuevo ∈ puntero a nodo

inicio
  if cabezaLista ≠ NIL entonces
    crear (nuevo)
    nuevo↑.siguiente ← cabezaLista↑.siguiente
    cabezaLista↑.siguiente ← nuevo
    nuevo↑.numero ← cabezaLista↑.numero
    cabezaLista↑.numero ← dato_nuevo
  si no
    crear (cabezaLista)
    cabezaLista↑.siguiente ← NIL
    cabezaLista↑.numero ← dato
  fin si
fin acción
```

Inserción en una lista ordenada ascendentemente

```
acción insertarOrdenado (cabezaLista ∈ puntero a nodo, dato ∈ entero)
variables
  nuevo ∈ puntero a nodo

inicio
  si cabezaLista ≠ NIL entonces
    cursor ← cabezaLista

    mientras cursor↑.siguiente≠NIL y cursor↑.numero < dato hacer
      cursor ← cursor↑.siguiente
    fin mientras

    si cursor↑.numero≥dato entonces
      crear (nuevo)
      nuevo↑.siguiente ← cursor↑.siguiente
      cursor↑.siguiente ← nuevo
      nuevo↑.numero ← cursor↑.numero
      cursor↑.numero ← dato
    si no
      crear (nuevo)
      nuevo↑.siguiente ← cursor↑.siguiente
```

```
      nuevo↑.numero ← dato
      cursor↑.siguiente ← nuevo
    fin si
  si no
    crear (cabezaLista)
    cabezaLista↑.siguiente ← NIL
    cabezaLista↑.numero ← dato
  fin si
fin acción
```

Eliminar un elemento de una lista simplemente enlazada

El algoritmo básico de eliminación simplemente debe recorrer la lista hasta encontrar el dato a eliminar, enlazar la lista de forma adecuada y destruir el nodo sobrante.

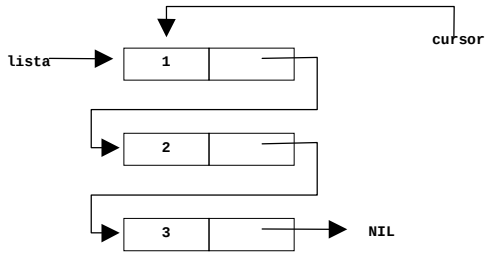
```
acción eliminarElemento (cabezaLista ∈ puntero a nodo, dato ∈ entero)
variables
  anterior,cursor ∈ puntero a nodo

inicio
  si cabezaLista≠NIL entonces
    cursor ← cabezaLista

    mientras cursor↑.siguiente≠NIL y cursor↑.numero≠dato hacer
      anterior ← cursor
      cursor ← cursor↑.siguiente
    fin mientras

    si cursor↑.numero=dato entonces
      si cursor≠cabezaLista entonces
        anterior↑.siguiente ← cursor↑.siguiente
        destruir(cursor)
      si no
        cursor ← cabezaLista↑.siguiente
        cabezaLista↑.numero ← cursor↑.numero
        cabezaLista↑.siguiente ← cursor↑.siguiente
        destruir(cursor)
      fin si
    fin si
  fin si
fin acción
```

A continuación se muestran de forma gráfica tres casos típicos de eliminación de elementos en una lista; el primero consistirá en borrar el último elemento de la lista, el segundo en borrar un elemento cualquiera y el último en la eliminación de la cabeza de la lista.

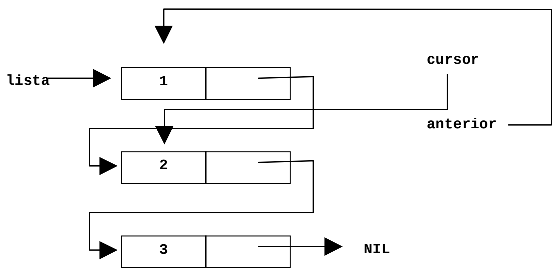
Eliminar el último elemento de una lista

En este ejemplo vamos a eliminar el último nodo de la lista que contiene los números ‘1’, ‘2’ y ‘3’; para ello se invocaría la acción `eliminarElemento` de la forma siguiente:

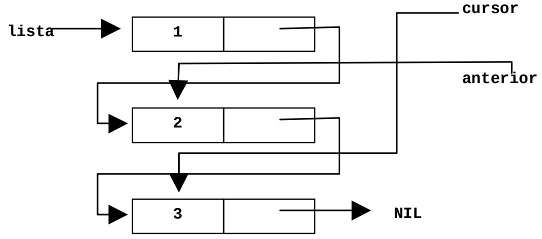
```
llamar eliminarElemento (lista,3)
```



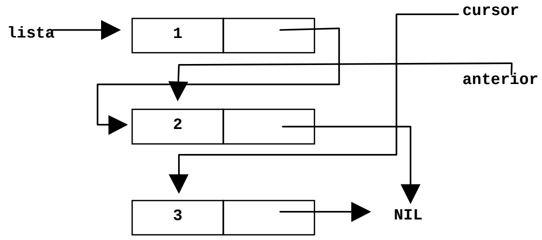

```
si cabezaLista≠NIL entonces
  cursor ← cabezaLista

  mientras cursor↑.siguiente≠NIL y cursor↑.numero≠dato hacer
    anterior ← cursor
    cursor ← cursor↑.siguiente
  fin mientras

  si cursor↑.numero=dato entonces
    si cursor≠cabezaLista entonces
      anterior↑.siguiente ← cursor↑.siguiente
      destruir(cursor)
    si no
      cursor ← cabezaLista↑.siguiente
      cabezaLista↑.numero ← cursor↑.numero
      cabezaLista↑.siguiente ← cursor↑.siguiente
      destruir(cursor)
    fin si
  fin si
fin si
```



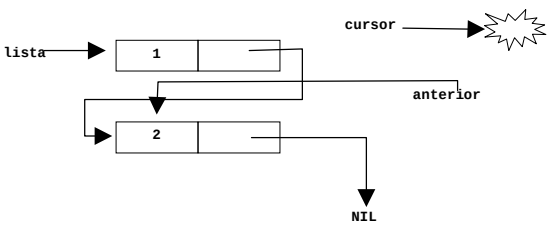

```
si cabezaLista≠NIL entonces
  cursor ← cabezaLista

  mientras cursor↑.siguiente≠NIL y cursor↑.numero≠dato hacer
    anterior ← cursor
    cursor ← cursor↑.siguiente
  fin mientras

  si cursor↑.numero=dato entonces
    si cursor≠cabezaLista entonces
      anterior↑.siguiente ← cursor↑.siguiente
      destruir(cursor)
    si no
      cursor ← cabezaLista↑.siguiente
      cabezaLista↑.numero ← cursor↑.numero
      cabezaLista↑.siguiente ← cursor↑.siguiente
      destruir(cursor)
    fin si
  fin si
fin si
```



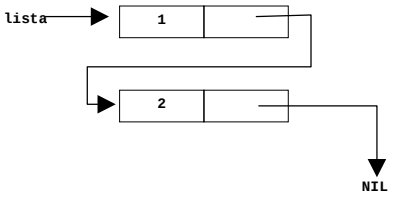

```
si cabezaLista≠NIL entonces
  cursor ← cabezaLista

  mientras cursor↑.siguiente≠NIL y cursor↑.numero≠dato hacer
    anterior ← cursor
    cursor ← cursor↑.siguiente
  fin mientras

  si cursor↑.numero=dato entonces
    si cursor≠cabezaLista entonces
      anterior↑.siguiente ← cursor↑.siguiente
      destruir(cursor)
    si no
      cursor ← cabezaLista↑.siguiente
      cabezaLista↑.numero ← cursor↑.numero
      cabezaLista↑.siguiente ← cursor↑.siguiente
      destruir(cursor)
    fin si
  fin si
fin si
```




```
si cabezaLista≠NIL entonces
  cursor ← cabezaLista

  mientras cursor↑.siguiente≠NIL y cursor↑.numero≠dato hacer
    anterior ← cursor
    cursor ← cursor↑.siguiente
  fin mientras

  si cursor↑.numero=dato entonces
    si cursor≠cabezaLista entonces
      anterior↑.siguiente ← cursor↑.siguiente
      destruir(cursor)
    si no
      cursor ← cabezaLista↑.siguiente
      cabezaLista↑.numero ← cursor↑.numero
      cabezaLista↑.siguiente ← cursor↑.siguiente
      destruir(cursor)
    fin si
  fin si
fin si
```




```
si cabezaLista≠NIL entonces
  cursor ← cabezaLista

  mientras cursor↑.siguiente≠NIL y cursor↑.numero≠dato hacer
    anterior ← cursor
    cursor ← cursor↑.siguiente
  fin mientras

  si cursor↑.numero=dato entonces
    si cursor≠cabezaLista entonces
      anterior↑.siguiente ← cursor↑.siguiente
      destruir(cursor)
    si no
      cursor ← cabezaLista↑.siguiente
      cabezaLista↑.numero ← cursor↑.numero
      cabezaLista↑.siguiente ← cursor↑.siguiente
      destruir(cursor)
    fin si
  fin si
fin si
```

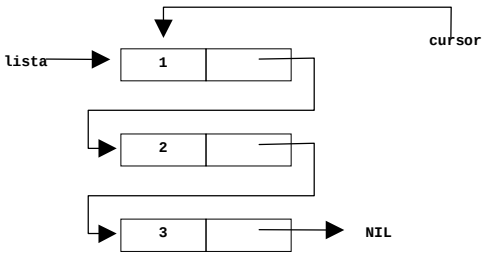


Al retornar al programa principal la lista queda como se muestra en la figura.

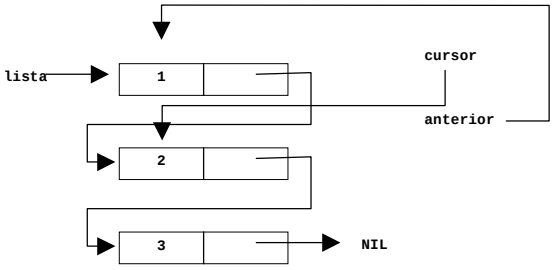
Eliminar un elemento cualquiera de una lista

En este ejemplo vamos a eliminar el segundo nodo de la lista que contiene los números ‘1’, ‘2’ y ‘3’; para ello se invocaría la acción `eliminarElemento` de la forma siguiente:

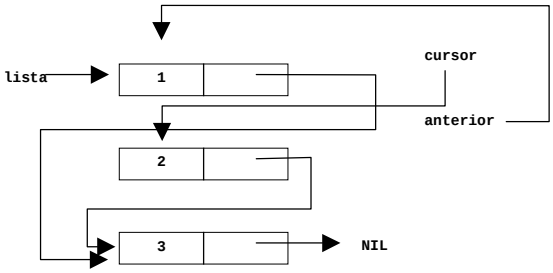
```
llamar eliminarElemento (lista,2)
```



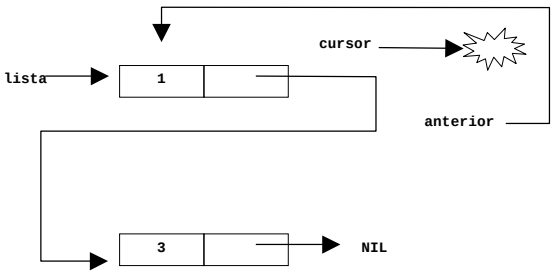

```
si cabezaLista≠NIL entonces
  cursor ← cabezaLista

  mientras cursor↑.siguiente≠NIL y cursor↑.numero≠dato hacer
    anterior ← cursor
    cursor ← cursor↑.siguiente
  fin mientras

  si cursor↑.numero=dato entonces
    si cursor≠cabezaLista entonces
      anterior↑.siguiente ← cursor↑.siguiente
      destruir(cursor)
    si no
      cursor ← cabezaLista↑.siguiente
      cabezaLista↑.numero ← cursor↑.numero
      cabezaLista↑.siguiente ← cursor↑.siguiente
      destruir(cursor)
    fin si
  fin si
fin si
```



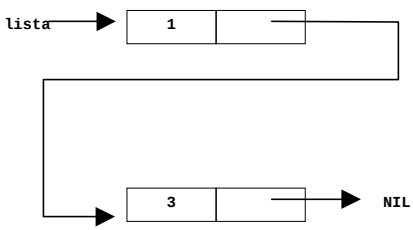

```
si cabezaLista≠NIL entonces
  cursor ← cabezaLista

  mientras cursor↑.siguiente≠NIL y cursor↑.numero≠dato hacer
    anterior ← cursor
    cursor ← cursor↑.siguiente
  fin mientras

  si cursor↑.numero=dato entonces
    si cursor≠cabezaLista entonces
      anterior↑.siguiente ← cursor↑.siguiente
      destruir(cursor)
    si no
      cursor ← cabezaLista↑.siguiente
      cabezaLista↑.numero ← cursor↑.numero
      cabezaLista↑.siguiente ← cursor↑.siguiente
      destruir(cursor)
    fin si
  fin si
fin si
```




```
si cabezaLista≠NIL entonces
  cursor ← cabezaLista

  mientras cursor↑.siguiente≠NIL y cursor↑.numero≠dato hacer
    anterior ← cursor
    cursor ← cursor↑.siguiente
  fin mientras

  si cursor↑.numero=dato entonces
    si cursor≠cabezaLista entonces
      anterior↑.siguiente ← cursor↑.siguiente
      destruir(cursor)
    si no
      cursor ← cabezaLista↑.siguiente
      cabezaLista↑.numero ← cursor↑.numero
      cabezaLista↑.siguiente ← cursor↑.siguiente
      destruir(cursor)
    fin si
  fin si
fin si
```




```
si cabezaLista≠NIL entonces
  cursor ← cabezaLista

  mientras cursor↑.siguiente≠NIL y cursor↑.numero≠dato hacer
    anterior ← cursor
    cursor ← cursor↑.siguiente
  fin mientras

  si cursor↑.numero=dato entonces
    si cursor≠cabezaLista entonces
      anterior↑.siguiente ← cursor↑.siguiente
      destruir(cursor)
    si no
      cursor ← cabezaLista↑.siguiente
      cabezaLista↑.numero ← cursor↑.numero
      cabezaLista↑.siguiente ← cursor↑.siguiente
      destruir(cursor)
    fin si
  fin si
fin si
```

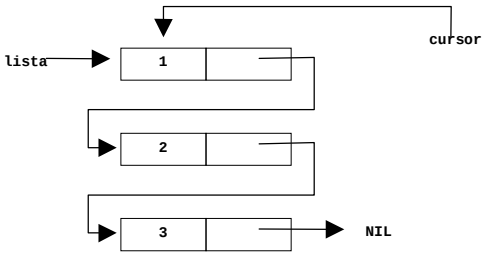


Al retornar al programa principal la lista queda como se muestra en la figura.

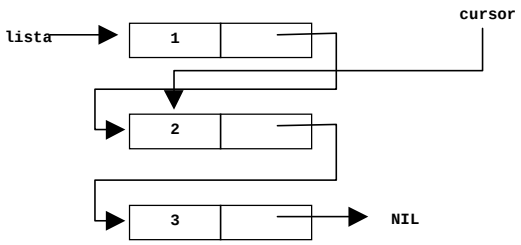
Eliminar la cabeza de una lista

En este ejemplo vamos a eliminar la cabeza de una lista que contiene los números ‘1’, ‘2’ y ‘3’; para ello se invocaría la acción `eliminarElemento` de la forma siguiente:

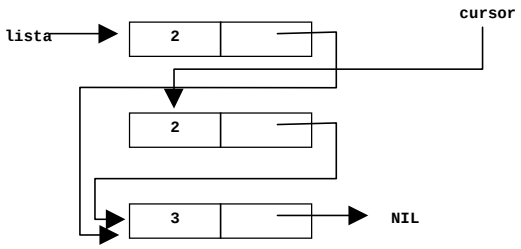
```
llamar eliminarElemento (lista,1)
```



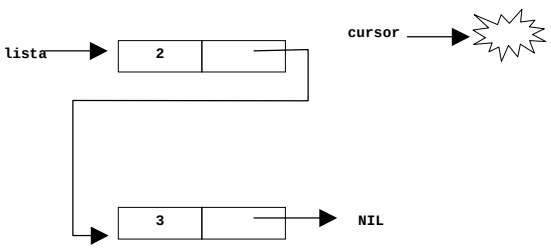

```
si cabezaLista≠NIL entonces
  cursor ← cabezaLista

  mientras cursor↑.siguiente≠NIL y cursor↑.numero≠dato hacer
    anterior ← cursor
    cursor ← cursor↑.siguiente
  fin mientras

  si cursor↑.numero=dato entonces
    si cursor≠cabezaLista entonces
      anterior↑.siguiente ← cursor↑.siguiente
      destruir(cursor)
    si no
      cursor ← cabezaLista↑.siguiente
      cabezaLista↑.numero ← cursor↑.numero
      cabezaLista↑.siguiente ← cursor↑.siguiente
      destruir(cursor)
    fin si
  fin si
fin si
```



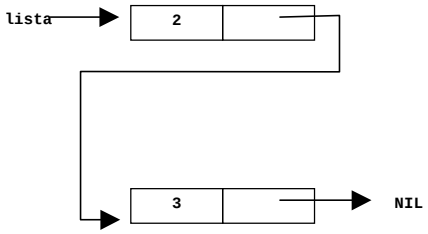

```
si cabezaLista≠NIL entonces
  cursor ← cabezaLista

  mientras cursor↑.siguiente≠NIL y cursor↑.numero≠dato hacer
    anterior ← cursor
    cursor ← cursor↑.siguiente
  fin mientras

  si cursor↑.numero=dato entonces
    si cursor≠cabezaLista entonces
      anterior↑.siguiente ← cursor↑.siguiente
      destruir(cursor)
    si no
      cursor ← cabezaLista↑.siguiente
      cabezaLista↑.numero ← cursor↑.numero
      cabezaLista↑.siguiente ← cursor↑.siguiente
      destruir(cursor)
    fin si
  fin si
fin si
```




```
si cabezaLista≠NIL entonces
  cursor ← cabezaLista

  mientras cursor↑.siguiente≠NIL y cursor↑.numero≠dato hacer
    anterior ← cursor
    cursor ← cursor↑.siguiente
  fin mientras

  si cursor↑.numero=dato entonces
    si cursor≠cabezaLista entonces
      anterior↑.siguiente ← cursor↑.siguiente
      destruir(cursor)
    si no
      cursor ← cabezaLista↑.siguiente
      cabezaLista↑.numero ← cursor↑.numero
      cabezaLista↑.siguiente ← cursor↑.siguiente
      destruir(cursor)
    fin si
  fin si
fin si
```




```
si cabezaLista≠NIL entonces
  cursor ← cabezaLista

  mientras cursor↑.siguiente≠NIL y cursor↑.numero≠dato hacer
    anterior ← cursor
    cursor ← cursor↑.siguiente
  fin mientras

  si cursor↑.numero=dato entonces
    si cursor≠cabezaLista entonces
      anterior↑.siguiente ← cursor↑.siguiente
      destruir(cursor)
    si no
      cursor ← cabezaLista↑.siguiente
      cabezaLista↑.numero ← cursor↑.numero
      cabezaLista↑.siguiente ← cursor↑.siguiente
      destruir(cursor)
    fin si
  fin si
fin si
```



Al retornar al programa principal la lista queda como se muestra en la figura.

Extracción de elementos de pilas y colas

En caso de que la lista sea una pila o una cola no se suele utilizar la operación de eliminación sino una variación de la misma que permite “extraer” un elemento de la estructura de datos. La extracción siempre elimina el valor que se encuentra en la cabeza de la lista y, además, retorna su valor al programa principal; por esa razón se implementa como una función, una posible implementación de la misma se muestra a continuación:

```
entero función extraerElemento (cabezaLista ∈ puntero a nodo)
variables
  cursor ∈ puntero a nodo

inicio
  si cabezaLista≠NIL entonces
    extraerElemento ← cabezaLista↑.numero

    cursor ← cabezaLista↑.siguiente

    cabezaLista↑.numero ← cursor↑.numero
    cabezaLista↑.siguiente ← cursor↑.siguiente

    destruir(cursor)
  fin si
fin acción
```

Vaciado de una lista simplemente enlazada

```
acción vaciarLista (cabezaLista ∈ puntero a nodo)
variables
  cursor ∈ puntero a nodo

inicio
  cursor ← cabezaLista^.siguiente

  mientras cursor ≠ NIL hacer
    cabezaLista^.siguiente ← cursor^.siguiente
    destruir(cursor)
    cursor ← cabezaLista^.siguiente
  fin mientras

  destruir(cabezaLista)
fin acción
```

Para vaciar una lista simplemente enlazada basta con eliminar el nodo que se encuentra en la cabeza de la lista de forma repetida hasta que la lista quede vacía. A la izquierda se muestra una posible implementación.

Resumen

1. Una estructura de datos es dinámica si para su construcción se utilizan variables dinámicas, es decir, variables cuya creación y eliminación se realiza en tiempo de ejecución mediante el empleo de punteros.
2. La estructura dinámica más sencilla consiste en una secuencia de elementos con un solo enlace entre ellos; esta estructura se denomina lista simplemente enlazada y cada elemento recibe el nombre de nodo.
3. Las operaciones más comunes en una lista simplemente enlazada son las siguientes:
   - Recorrido de la lista.
   - Búsqueda de un elemento en la lista.
   - Inserción de elementos en la lista.
   - Supresión de elementos de la lista.
   - Vaciado de la lista.
4. Las operaciones de recorrido, inserción, supresión y vaciado se implementan, generalmente, como acciones mientras que la búsqueda se suele implementar como una función que retorna un puntero.
5. Todas las operaciones sobre listas pueden implementarse tanto de forma iterativa como recursiva.
6. La naturaleza exacta de una lista viene marcada por la forma en que se comporta el algoritmo de inserción. Así, existen listas en las que se inserta siempre al final (y se extrae por la cabeza), se denominan colas o listas FIFO (*First In First Out*); listas en las que se inserta siempre al principio (y se extrae por la cabeza), denominadas pilas o listas LIFO (*Last In First Out*) y listas en las que se inserta según una relación de orden entre los nodos.
7. Todos estos métodos de inserción emplean dos posibles técnicas: insertar “por delante” de un nodo dado o insertar “por detrás” del nodo.
8. La operación de supresión de elementos de una lista, por su parte, tiene dos casos básicos: eliminar la cabeza de la lista y eliminar un nodo cualquiera.
9. En el caso de pilas y colas no se emplea la eliminación de elementos sino la “extracción”, esta operación se implementa como una función que siempre elimina el nodo de la cabeza de la lista y retorna su valor al programa principal.
10. Para vaciar una lista tan sólo hay que eliminar de forma sistemática los nodos que se encuentran en la cabeza de la lista hasta que ésta apunta a NIL.